

Francesco Ricci · Lior Rokach · Bracha Shapira ·
Paul B. Kantor
Editors

Recommender Systems Handbook

 Springer

Editors

Francesco Ricci
Free University of Bozen-Bolzano
Faculty of Computer Science
Piazza Domenicani 3
39100 Bolzano
Italy
fricci@unibz.it

Lior Rokach
Ben-Gurion University of the
Negev
Dept. Information Systems
Engineering
84105 Beer-Sheva
Israel
liorrk@bgu.ac.il

Bracha Shapira
Ben-Gurion University of the
Negev
Dept. Information Systems
Engineering
Beer-Sheva
Israel
bshapira@bgu.ac.il

Paul B. Kantor
Rutgers University
School of Communication,
Information & Library Studies
Huntington Street 4
08901-1071 New Brunswick
New Jersey
SCILS Bldg.
USA
kantor@scils.rutgers.edu

ISBN 978-0-387-85819-7 e-ISBN 978-0-387-85820-3
DOI 10.1007/978-0-387-85820-3
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010937590

© Springer Science+Business Media, LLC 2011

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden. The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Recommender Systems are software tools and techniques providing suggestions for items to be of use to a user. The suggestions provided are aimed at supporting their users in various decision-making processes, such as what items to buy, what music to listen, or what news to read. Recommender systems have proven to be valuable means for online users to cope with the information overload and have become one of the most powerful and popular tools in electronic commerce. Correspondingly, various techniques for recommendation generation have been proposed and during the last decade, many of them have also been successfully deployed in commercial environments.

Development of recommender systems is a multi-disciplinary effort which involves experts from various fields such as Artificial intelligence, Human Computer Interaction, Information Technology, Data Mining, Statistics, Adaptive User Interfaces, Decision Support Systems, Marketing, or Consumer Behavior. *Recommender Systems Handbook: A Complete Guide for Research Scientists and Practitioners* aims to impose a degree of order upon this diversity by presenting a coherent and unified repository of recommender systems' major concepts, theories, methodologies, trends, challenges and applications. This is the first comprehensive book which is dedicated entirely to the field of recommender systems and covers several aspects of the major techniques. Its informative, factual pages will provide researchers, students and practitioners in industry with a comprehensive, yet concise and convenient reference source to recommender systems. The book describes in detail the classical methods, as well as extensions and novel approaches that were recently introduced. The book consists of five parts: techniques, applications and evaluation of recommender systems, interacting with recommender systems, recommender systems and communities, and advanced algorithms. The first part presents the most popular and fundamental techniques used nowadays for building recommender systems, such as collaborative filtering, content-based filtering, data mining methods and context-aware methods. The second part starts by surveying techniques and approaches that have been used to evaluate the quality of the recommendations. Then deals with the practical aspects of designing recommender systems, it describes design and implementation consideration, setting guidelines for the selection of the

more suitable algorithms. The section continues considering aspects that may affect the design and finally, it discusses methods, challenges and measures to be applied for the evaluation of the developed systems. The third part includes papers dealing with a number of issues related to the presentation, browsing, explanation and visualization of the recommendations, and techniques that make the recommendation process more structured and conversational.

The fourth part is fully dedicated to a rather new topic, which is however rooted in the core idea of a collaborative recommender, i.e., exploiting user generated content of various types to build new types and more credible recommendations.

Finally the last section collects a few papers on some advanced topics, such as the exploitation of active learning principles to guide the acquisition of new knowledge, techniques suitable for making a recommender system robust against attacks of malicious users, and recommender systems that aggregate multiple types of user feedbacks and preferences to build more reliable recommendations.

We would like to thank all authors for their valuable contributions. We would like to express gratitude for all reviewers that generously gave comments on drafts or counsel otherwise. We would like to express our special thanks to Susan Lagerstrom-Fife and staff members of Springer for their kind cooperation throughout the production of this book. Finally, we wish this handbook will contribute to the growth of this subject, we wish to the novices a fruitful learning path, and to those more experts a compelling application of the ideas discussed in this handbook and a fruitful development of this challenging research area.

Francesco Ricci
Lior Rokach
Bracha Shapira
Paul B. Kantor

May 2010

Contents

1	Introduction to Recommender Systems Handbook	1
	Francesco Ricci, Lior Rokach and Bracha Shapira	
1.1	Introduction	1
1.2	Recommender Systems Function	4
1.3	Data and Knowledge Sources	7
1.4	Recommendation Techniques	10
1.5	Application and Evaluation	14
1.6	Recommender Systems and Human Computer Interaction	17
1.6.1	Trust, Explanations and Persuasiveness	18
1.6.2	Conversational Systems	19
1.6.3	Visualization	21
1.7	Recommender Systems as a Multi-Disciplinary Field	21
1.8	Emerging Topics and Challenges	23
1.8.1	Emerging Topics Discussed in the Handbook	23
1.8.2	Challenges	26
	References	29

Part I Basic Techniques

2	Data Mining Methods for Recommender Systems	39
	Xavier Amatriain, Alejandro Jaimes, Nuria Oliver, and Josep M. Pujol	
2.1	Introduction	39
2.2	Data Preprocessing	40
2.2.1	Similarity Measures	41
2.2.2	Sampling	42
2.2.3	Reducing Dimensionality	44
2.2.4	Denoising	47
2.3	Classification	48
2.3.1	Nearest Neighbors	48
2.3.2	Decision Trees	50
2.3.3	Ruled-based Classifiers	51

2.3.4	Bayesian Classifiers	52
2.3.5	Artificial Neural Networks	54
2.3.6	Support Vector Machines	56
2.3.7	Ensembles of Classifiers	58
2.3.8	Evaluating Classifiers	59
2.4	Cluster Analysis	61
2.4.1	k -Means	62
2.4.2	Alternatives to k -means	63
2.5	Association Rule Mining	64
2.6	Conclusions	66
	References	67
3	Content-based Recommender Systems: State of the Art and Trends	73
	Pasquale Lops, Marco de Gemmis and Giovanni Semeraro	
3.1	Introduction	74
3.2	Basics of Content-based Recommender Systems	75
3.2.1	A High Level Architecture of Content-based Systems	75
3.2.2	Advantages and Drawbacks of Content-based Filtering	78
3.3	State of the Art of Content-based Recommender Systems	79
3.3.1	Item Representation	80
3.3.2	Methods for Learning User Profiles	90
3.4	Trends and Future Research	94
3.4.1	The Role of User Generated Content in the Recommendation Process	94
3.4.2	Beyond Over-specialization: Serendipity	96
3.5	Conclusions	99
	References	100
4	A Comprehensive Survey of Neighborhood-based Recommendation Methods	107
	Christian Desrosiers and George Karypis	
4.1	Introduction	107
4.1.1	Formal Definition of the Problem	108
4.1.2	Overview of Recommendation Approaches	110
4.1.3	Advantages of Neighborhood Approaches	112
4.1.4	Objectives and Outline	113
4.2	Neighborhood-based Recommendation	114
4.2.1	User-based Rating Prediction	115
4.2.2	User-based Classification	116
4.2.3	Regression VS Classification	117
4.2.4	Item-based Recommendation	117
4.2.5	User-based VS Item-based Recommendation	118
4.3	Components of Neighborhood Methods	120
4.3.1	Rating Normalization	121
4.3.2	Similarity Weight Computation	124
4.3.3	Neighborhood Selection	129

- 4.4 Advanced Techniques 131
 - 4.4.1 Dimensionality Reduction Methods 132
 - 4.4.2 Graph-based Methods 135
- 4.5 Conclusion 139
- References 140

- 5 Advances in Collaborative Filtering 145**

Yehuda Koren and Robert Bell

 - 5.1 Introduction 145
 - 5.2 Preliminaries 147
 - 5.2.1 Baseline predictors 148
 - 5.2.2 The Netflix data 149
 - 5.2.3 Implicit feedback 150
 - 5.3 Matrix factorization models 151
 - 5.3.1 SVD 151
 - 5.3.2 SVD++ 153
 - 5.3.3 Time-aware factor model 154
 - 5.3.4 Comparison 159
 - 5.3.5 Summary 160
 - 5.4 Neighborhood models 161
 - 5.4.1 Similarity measures 162
 - 5.4.2 Similarity-based interpolation 163
 - 5.4.3 Jointly derived interpolation weights 165
 - 5.4.4 Summary 168
 - 5.5 Enriching neighborhood models 168
 - 5.5.1 A global neighborhood model 169
 - 5.5.2 A factorized neighborhood model 173
 - 5.5.3 Temporal dynamics at neighborhood models 180
 - 5.5.4 Summary 182
 - 5.6 Between neighborhood and factorization 182
 - References 184

- 6 Developing Constraint-based Recommenders 187**

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach and Markus Zanker

 - 6.1 Introduction 187
 - 6.2 Development of Recommender Knowledge Bases 191
 - 6.3 User Guidance in Recommendation Processes 194
 - 6.4 Calculating Recommendations 203
 - 6.5 Experiences from Projects and Case Studies 205
 - 6.6 Future Research Issues 207
 - 6.7 Summary 212
 - References 212

7 Context-Aware Recommender Systems 217
 Gediminas Adomavicius and Alexander Tuzhilin

7.1 Introduction and Motivation 218

7.2 Context in Recommender Systems 219

7.2.1 What is Context? 219

7.2.2 Modeling Contextual Information in
 Recommender Systems 223

7.2.3 Obtaining Contextual Information 228

7.3 Paradigms for Incorporating Context in Recommender Systems . . 230

7.3.1 Contextual Pre-Filtering 233

7.3.2 Contextual Post-Filtering 237

7.3.3 Contextual Modeling 238

7.4 Combining Multiple Approaches 243

7.4.1 Case Study of Combining Multiple Pre-Filters:
 Algorithms 244

7.4.2 Case Study of Combining Multiple Pre-Filters:
 Experimental Results 245

7.5 Additional Issues in Context-Aware Recommender Systems 247

7.6 Conclusions 249

References 250

Part II Applications and Evaluation of RSs

8 Evaluating Recommendation Systems 257
 Guy Shani and Asela Gunawardana

8.1 Introduction 258

8.2 Experimental Settings 260

8.2.1 Offline Experiments 261

8.2.2 User Studies 263

8.2.3 Online Evaluation 266

8.2.4 Drawing Reliable Conclusions 267

8.3 Recommendation System Properties 271

8.3.1 User Preference 272

8.3.2 Prediction Accuracy 273

8.3.3 Coverage 281

8.3.4 Confidence 283

8.3.5 Trust 285

8.3.6 Novelty 285

8.3.7 Serendipity 286

8.3.8 Diversity 288

8.3.9 Utility 289

8.3.10 Risk 290

8.3.11 Robustness 290

8.3.12 Privacy 291

8.3.13 Adaptivity 292

- 8.3.14 Scalability 293
- 8.4 Conclusion 293
- References 294
- 9 A Recommender System for an IPTV Service Provider: a Real Large-Scale Production Environment. 299**
Riccardo Bambini, Paolo Cremonesi and Roberto Turrin
 - 9.1 Introduction 299
 - 9.2 IPTV Architecture 301
 - 9.2.1 IPTV Search Problems 302
 - 9.3 Recommender System Architecture 303
 - 9.3.1 Data Collection 304
 - 9.3.2 Batch and Real-Time Stages 306
 - 9.4 Recommender Algorithms 308
 - 9.4.1 Overview of Recommender Algorithms 308
 - 9.4.2 LSA Content-Based Algorithm 311
 - 9.4.3 Item-based Collaborative Algorithm 314
 - 9.4.4 Dimensionality-Reduction-Based Collaborative Algorithm 316
 - 9.5 Recommender Services 318
 - 9.6 System Evaluation 319
 - 9.6.1 Off-Line Analysis 321
 - 9.6.2 On-line Analysis 325
 - 9.7 Conclusions 329
 - References 329
- 10 How to Get the Recommender Out of the Lab? 333**
J rome Picault, Myriam Rib re, David Bonnefoy and Kevin Mercer
 - 10.1 Introduction 334
 - 10.2 Designing Real-World Recommender Systems 334
 - 10.3 Understanding the Recommender Environment 335
 - 10.3.1 Application Model 335
 - 10.3.2 User Model 340
 - 10.3.3 Data Model 344
 - 10.3.4 A Method for Using Environment Models 349
 - 10.4 Understanding the Recommender Validation Steps in an Iterative Design Process 350
 - 10.4.1 Validation of the Algorithms 350
 - 10.4.2 Validation of the Recommendations 351
 - 10.5 Use Case: a Semantic News Recommendation System 355
 - 10.5.1 Context: MESH Project 356
 - 10.5.2 Environmental Models in MESH 357
 - 10.5.3 In Practice: Iterative Instantiations of Models 361
 - 10.6 Conclusion 362
 - References 362

11	Matching Recommendation Technologies and Domains	367
	Robin Burke and Maryam Ramezani	
11.1	Introduction	367
11.2	Related Work	368
11.3	Knowledge Sources	368
11.3.1	Recommendation types	370
11.4	Domain	372
11.4.1	Heterogeneity	372
11.4.2	Risk	373
11.4.3	Churn	373
11.4.4	Interaction Style	374
11.4.5	Preference stability	374
11.4.6	Scrutability	375
11.5	Knowledge Sources	375
11.5.1	Social Knowledge	375
11.5.2	Individual	376
11.5.3	Content	377
11.6	Mapping Domains to Technologies	378
11.6.1	Algorithms	380
11.6.2	Sample Recommendation Domains	381
11.7	Conclusion	382
	References	382
12	Recommender Systems in Technology Enhanced Learning	387
	Nikos Manouselis, Hendrik Drachsler, Riina Vuorikari, Hans Hummel and Rob Koper	
12.1	Introduction	388
12.2	Background	389
12.3	Related Work	392
12.4	Survey of TEL Recommender Systems	399
12.5	Evaluation of TEL Recommenders	404
12.6	Conclusions and further work	408
	References	409
Part III Interacting with Recommender Systems		
13	On the Evolution of Critiquing Recommenders	419
	Lorraine McGinty and James Reilly	
13.1	Introduction	419
13.2	The Early Days: Critiquing Systems/Recognised Benefits	420
13.3	Representation & Retrieval Challenges for Critiquing Systems ...	422
13.3.1	Approaches to Critique Representation	422
13.3.2	Retrieval Challenges in Critique-Based Recommenders ..	430
13.4	Interfacing Considerations Across Critiquing Platforms	438
13.4.1	Scaling to Alternate Critiquing Platforms	438
13.4.2	Direct Manipulation Interfaces vs Restricted User Control	440

- 13.4.3 Supporting Explanation, Confidence & Trust 441
- 13.4.4 Visualisation, Adaptivity, and Partitioned Dynamicity . . . 443
- 13.4.5 Respecting Multi-cultural Usability Differences 445
- 13.5 Evaluating Critiquing: Resources, Methodologies and Criteria 445
 - 13.5.1 Resources & Methodologies 446
 - 13.5.2 Evaluation Criteria 446
- 13.6 Conclusion / Open Challenges & Opportunities 448
- References 449

14 Creating More Credible and Persuasive Recommender Systems: The Influence of Source Characteristics on Recommender

System Evaluations 455
 Kyung-Hyan Yoo and Ulrike Gretzel

- 14.1 Introduction 455
- 14.2 Recommender Systems as Social Actors 456
- 14.3 Source Credibility 457
 - 14.3.1 Trustworthiness 458
 - 14.3.2 Expertise 458
 - 14.3.3 Influences on Source Credibility 458
- 14.4 Source Characteristics Studied in Human-Human Interactions 459
 - 14.4.1 Similarity 459
 - 14.4.2 Likeability 460
 - 14.4.3 Symbols of Authority 460
 - 14.4.4 Styles of Speech 461
 - 14.4.5 Physical Attractiveness 461
 - 14.4.6 Humor 461
- 14.5 Source Characteristics in Human-Computer Interactions 462
- 14.6 Source Characteristics in Human-Recommender System Interactions 463
 - 14.6.1 Recommender system type 463
 - 14.6.2 Input characteristics 464
 - 14.6.3 Process characteristics 465
 - 14.6.4 Output characteristics 465
 - 14.6.5 Characteristics of embodied agents 467
- 14.7 Discussion 468
- 14.8 Implications 468
- 14.9 Directions for future research 470
- References 471

15 Designing and Evaluating Explanations for Recommender Systems 479

Nava Tintarev and Judith Masthoff

- 15.1 Introduction 479
- 15.2 Guidelines 481
- 15.3 Explanations in Expert Systems 481
- 15.4 Defining Goals 482
 - 15.4.1 Explain How the System Works: Transparency 483

- 15.4.2 Allow Users to Tell the System it is Wrong: Scrutability 485
- 15.4.3 Increase Users’ Confidence in the System: Trust 485
- 15.4.4 Convince Users to Try or Buy: Persuasiveness 487
- 15.4.5 Help Users Make Good Decisions: Effectiveness 488
- 15.4.6 Help Users Make Decisions Faster: Efficiency 490
- 15.4.7 Make the use of the system enjoyable: Satisfaction 491
- 15.5 Evaluating the Impact of Explanations on the Recommender System 492
 - 15.5.1 Accuracy Metrics 493
 - 15.5.2 Learning Rate 493
 - 15.5.3 Coverage 494
 - 15.5.4 Acceptance 494
- 15.6 Designing the Presentation and Interaction with Recommendations 495
 - 15.6.1 Presenting Recommendations 495
 - 15.6.2 Interacting with the Recommender System 496
- 15.7 Explanation Styles 497
 - 15.7.1 Collaborative-Based Style Explanations 500
 - 15.7.2 Content-Based Style Explanation 501
 - 15.7.3 Case-Based Reasoning (CBR) Style Explanations 503
 - 15.7.4 Knowledge and Utility-Based Style Explanations 504
 - 15.7.5 Demographic Style Explanations 505
- 15.8 Summary and future directions 505
- References 507

16 Usability Guidelines for Product Recommenders Based on Example Critiquing Research 511

Pearl Pu, Boi Faltings, Li Chen, Jiyong Zhang and Paolo Viappiani

- 16.1 Introduction 512
- 16.2 Preliminaries 513
 - 16.2.1 Interaction Model 513
 - 16.2.2 Utility-Based Recommenders 515
 - 16.2.3 The Accuracy, Confidence, Effort Framework 517
 - 16.2.4 Organization of this Chapter 518
- 16.3 Related Work 518
 - 16.3.1 Types of Recommenders 518
 - 16.3.2 Rating-based Systems 519
 - 16.3.3 Case-based Systems 519
 - 16.3.4 Utility-based Systems 519
 - 16.3.5 Critiquing-based Systems 520
 - 16.3.6 Other Design Guidelines 520
- 16.4 Initial Preference Elicitation 521
- 16.5 Stimulating Preference Expression with Examples 525
 - 16.5.1 How Many Examples to Show 527
 - 16.5.2 What Examples to Show 527
- 16.6 Preference Revision 530

- 16.6.1 Preference Conflicts and Partial Satisfaction 531
- 16.6.2 Tradeoff Assistance 532
- 16.7 Display Strategies 534
 - 16.7.1 Recommending One Item at a Time 534
 - 16.7.2 Recommending K best Items 535
 - 16.7.3 Explanation Interfaces 536
- 16.8 A Model for Rationalizing the Guidelines 537
- 16.9 Conclusion 541
- References 541

17 Map Based Visualization of Product Catalogs 547

Martijn Kagie, Michiel van Wezel and Patrick J.F. Groenen

- 17.1 Introduction 547
- 17.2 Methods for Map Based Visualization 549
 - 17.2.1 Self-Organizing Maps 550
 - 17.2.2 Treemaps 551
 - 17.2.3 Multidimensional Scaling 553
 - 17.2.4 Nonlinear Principal Components Analysis 553
- 17.3 Product Catalog Maps 554
 - 17.3.1 Multidimensional Scaling 555
 - 17.3.2 Nonlinear Principal Components Analysis 558
- 17.4 Determining Attribute Weights using Clickstream Analysis 559
 - 17.4.1 Poisson Regression Model 560
 - 17.4.2 Handling Missing Values 560
 - 17.4.3 Choosing Weights Using Poisson Regression 561
 - 17.4.4 Stepwise Poisson Regression Model 562
- 17.5 Graphical Shopping Interface 562
- 17.6 E-Commerce Applications 563
 - 17.6.1 MDS Based Product Catalog Map Using Attribute Weights 564
 - 17.6.2 NL-PCA Based Product Catalog Map 568
 - 17.6.3 Graphical Shopping Interface 570
- 17.7 Conclusions and Outlook 573
- References 574

Part IV Recommender Systems and Communities

18 Communities, Collaboration, and Recommender Systems in Personalized Web Search 579

Barry Smyth, Maurice Coyle and Peter Briggs

- 18.1 Introduction 579
- 18.2 A Brief History of Web Search 581
- 18.3 The Future of Web Search 583
 - 18.3.1 Personalized Web Search 584
 - 18.3.2 Collaborative Information Retrieval 588
 - 18.3.3 Towards Social Search 590

- 18.4 Case-Study 1 - Community-Based Web Search 591
 - 18.4.1 Repetition and Regularity in Search Communities 592
 - 18.4.2 The Collaborative Web Search System 593
 - 18.4.3 Evaluation 596
 - 18.4.4 Discussion 598
- 18.5 Case-Study 2 - Web Search. Shared..... 598
 - 18.5.1 The HeyStaks System 599
 - 18.5.2 The HeyStaks Recommendation Engine 602
 - 18.5.3 Evaluation 604
 - 18.5.4 Discussion 607
- 18.6 Conclusions 607
- References 609

- 19 Social Tagging Recommender Systems 615**
 Leandro Balby Marinho, Alexandros Nanopoulos, Lars Schmidt-Thieme, Robert Jäschke, Andreas Hotho, Gerd Stumme and Panagiotis Symeonidis
 - 19.1 Introduction 616
 - 19.2 Social Tagging Recommenders Systems 617
 - 19.2.1 Folksonomy 618
 - 19.2.2 The Traditional Recommender Systems Paradigm 619
 - 19.2.3 Multi-mode Recommendations 620
 - 19.3 Real World Social Tagging Recommender Systems 621
 - 19.3.1 What are the Challenges? 621
 - 19.3.2 BibSonomy as Study Case 622
 - 19.3.3 Tag Acquisition 624
 - 19.4 Recommendation Algorithms for Social Tagging Systems 626
 - 19.4.1 Collaborative Filtering 626
 - 19.4.2 Recommendation based on Ranking 630
 - 19.4.3 Content-Based Social Tagging RS..... 634
 - 19.4.4 Evaluation Protocols and Metrics 637
 - 19.5 Comparison of Algorithms 639
 - 19.6 Conclusions and Research Directions..... 640
 - References 642

- 20 Trust and Recommendations 645**
 Patricia Victor, Martine De Cock, and Chris Cornelis
 - 20.1 Introduction 645
 - 20.2 Computational Trust..... 647
 - 20.2.1 Trust Representation 648
 - 20.2.2 Trust Computation..... 650
 - 20.3 Trust-Enhanced Recommender Systems 655
 - 20.3.1 Motivation 656
 - 20.3.2 State of the Art..... 658
 - 20.3.3 Empirical Comparison 664
 - 20.4 Recent Developments and Open Challenges 670

20.5 Conclusions 672
 References 672

21 Group Recommender Systems: Combining Individual Models 677

Judith Masthoff
 21.1 Introduction 677
 21.2 Usage Scenarios and Classification of Group Recommenders 679
 21.2.1 Interactive Television 679
 21.2.2 Ambient Intelligence 679
 21.2.3 Scenarios Underlying Related Work 680
 21.2.4 A Classification of Group Recommenders 681
 21.3 Aggregation Strategies 682
 21.3.1 Overview of Aggregation Strategies 682
 21.3.2 Aggregation Strategies Used in Related Work 683
 21.3.3 Which Strategy Performs Best 685
 21.4 Impact of Sequence Order 686
 21.5 Modelling Affective State 688
 21.5.1 Modelling an Individual’s Satisfaction on its Own 689
 21.5.2 Effects of the Group on an Individual’s Satisfaction 690
 21.6 Using Affective State inside Aggregation Strategies 691
 21.7 Applying Group Recommendation to Individual Users 693
 21.7.1 Multiple Criteria 693
 21.7.2 Cold-Start Problem 695
 21.7.3 Virtual Group Members 697
 21.8 Conclusions and Challenges 697
 21.8.1 Main Issues Raised 697
 21.8.2 Caveat: Group Modelling 698
 21.8.3 Challenges 698
 References 701

Part V Advanced Algorithms

22 Aggregation of Preferences in Recommender Systems 705

Gleb Beliakov, Tomasa Calvo and Simon James
 22.1 Introduction 705
 22.2 Types of Aggregation in Recommender Systems 706
 22.2.1 Aggregation of Preferences in CF 708
 22.2.2 Aggregation of Features in CB and
 UB Recommendation 708
 22.2.3 Profile Construction for CB, UB 709
 22.2.4 Item and User Similarity and Neighborhood Formation .. 709
 22.2.5 Connectives in Case-Based Reasoning for RS 711
 22.2.6 Weighted Hybrid Systems 711
 22.3 Review of Aggregation Functions 712
 22.3.1 Definitions and Properties 712
 22.3.2 Aggregation Families 716
 22.4 Construction of Aggregation Functions 722

22.4.1	Data Collection and Preprocessing	722
22.4.2	Desired Properties, Semantics and Interpretation	724
22.4.3	Complexity and the Understanding of Function Behavior	725
22.4.4	Weight and Parameter Determination	726
22.5	Sophisticated Aggregation Procedures in Recommender Systems: Tailoring for Specific Applications	726
22.6	Conclusions	731
22.7	Further Reading	732
	References	733
23	Active Learning in Recommender Systems	735
	Neil Rubens, Dain Kaplan, and Masashi Sugiyama	
23.1	Introduction	735
23.1.1	Objectives of Active Learning in Recommender Systems	737
23.1.2	An Illustrative Example	738
23.1.3	Types of Active Learning	739
23.2	Properties of Data Points	740
23.2.1	Other Considerations	741
23.3	Active Learning in Recommender Systems	742
23.3.1	Method Summary Matrix	742
23.4	Active Learning Formulation	742
23.5	Uncertainty-based Active Learning	746
23.5.1	Output Uncertainty	746
23.5.2	Decision Boundary Uncertainty	748
23.5.3	Model Uncertainty	749
23.6	Error-based Active Learning	751
23.6.1	Instance-based Methods	752
23.6.2	Model-based	754
23.7	Ensemble-based Active Learning	756
23.7.1	Models-based	756
23.7.2	Candidates-based	757
23.8	Conversation-based Active Learning	760
23.8.1	Case-based Critique	761
23.8.2	Diversity-based	761
23.8.3	Query Editing-based	762
23.9	Computational Considerations	762
23.10	Discussion	763
	References	764
24	Multi-Criteria Recommender Systems	769
	Gediminas Adomavicius, Nikos Manouselis and YoungOk Kwon	
24.1	Introduction	769
24.2	Recommendation as a Multi-Criteria Decision Making Problem	771
24.2.1	Object of Decision	772
24.2.2	Family of Criteria	773

- 24.2.3 Global Preference Model 774
- 24.2.4 Decision Support Process 775
- 24.3 MCDM Framework for Recommender Systems:
Lessons Learned 776
- 24.4 Multi-Criteria Rating Recommendation 780
 - 24.4.1 Traditional single-rating recommendation problem 781
 - 24.4.2 Extending traditional recommender systems to include
multi-criteria ratings 782
- 24.5 Survey of Algorithms for Multi-Criteria Rating Recommenders... 783
 - 24.5.1 Engaging Multi-Criteria Ratings during Prediction 784
 - 24.5.2 Engaging Multi-Criteria Ratings
during Recommendation 791
- 24.6 Discussion and Future Work 795
- 24.7 Conclusions 797
- References 798
- 25 Robust Collaborative Recommendation 805**
Robin Burke, Michael P. O’Mahony and Neil J. Hurley
- 25.1 Introduction 805
- 25.2 Defining the Problem 807
 - 25.2.1 An Example Attack 809
- 25.3 Characterising Attacks 810
 - 25.3.1 Basic Attacks 810
 - 25.3.2 Low-knowledge attacks 811
 - 25.3.3 Nuke Attack Models 812
 - 25.3.4 Informed Attack Models 813
- 25.4 Measuring Robustness 814
 - 25.4.1 Evaluation Metrics 815
 - 25.4.2 Push Attacks 816
 - 25.4.3 Nuke Attacks 818
 - 25.4.4 Informed Attacks 819
 - 25.4.5 Attack impact 820
- 25.5 Attack Detection 820
 - 25.5.1 Evaluation Metrics 821
 - 25.5.2 Single Profile Detection 822
 - 25.5.3 Group Profile Detection 824
 - 25.5.4 Detection findings 827
- 25.6 Robust Algorithms 828
 - 25.6.1 Model-based Recommendation 828
 - 25.6.2 Robust Matrix Factorisation (RMF) 829
 - 25.6.3 Other Robust Recommendation Algorithms 830
 - 25.6.4 The Influence Limiter and Trust-based
Recommendation 831
- 25.7 Conclusion 832
- References 833
- Index 837**

List of Contributors

Gediminas Adomavicius
Department of Information and Decision Sciences
Carlson School of Management, University of Minnesota, Minneapolis, MN 55455,
USA
e-mail: gedas@umn.edu

Xavier Amatriain
Telefonica Research, Via Augusta, 122, Barcelona 08021, Spain
e-mail: xar@tid.es

Riccardo Bambini
Fastweb, via Francesco Caracciolo 51, Milano, Italy
e-mail: riccardo.bambini@fastweb.it

Gleb Beliakov
School of Information Technology, Deakin University, 221 Burwood Hwy,
Burwood 3125, Australia,
e-mail: gleb@deakin.edu.au

Robert Bell
AT&T Labs – Research
e-mail: rbell@research.att.com

David Bonnefoy
Pearltrees,
e-mail: david.bonnefoy@pearltrees.com

Peter Briggs
CLARITY: Centre for Sensor Web Technologies, School of Computer Science &
Informatics, University College Dublin, Ireland,
e-mail: Peter.Briggs@ucd.ie

Robin Burke
Center for Web Intelligence, School of Computer Science, Telecommunication and

Information Systems, DePaul University, Chicago, Illinois, USA

e-mail: rburke@cs.depaul.edu

Tomasa Calvo

Departamento de Ciencias de la Computación, Universidad de Alcalá
28871-Alcalá de Henares (Madrid), Spain.

e-mail: tomasa.calvo@uah.es

Li Chen

Human Computer Interaction Group, School of Computer and Communication
Sciences,
Swiss Federal Institute of Technology in Lausanne (EPFL), CH-1015, Lausanne,
Switzerland

e-mail: li.chen@epfl.ch

Martine De Cock

Institute of Technology, University of Washington Tacoma, 1900 Pacific Ave,
Tacoma, WA, USA (on leave from Ghent University)

e-mail: mdecock@u.washington.edu

Chris Cornelis

Dept. of Applied Mathematics and Computer Science, Ghent University, Krijgslaan
281 (S9), 9000 Gent, Belgium

e-mail: Patricia.Victor@ugent.be

Maurice Coyle

CLARITY: Centre for Sensor Web Technologies, School of Computer Science &
Informatics, University College Dublin, Ireland,

e-mail: Maurice.Coyle@ucd.ie

Paolo Cremonesi

Politecnico di Milano, p.zza Leonardo da Vinci 32, Milano, Italy Neptuny, via
Durando 10, Milano, Italy

e-mail: paolo.cremonesi@polimi.it

Christian Desrosiers

Department of Software Engineering and IT, École de Technologie Supérieure,
Montreal, Canada

e-mail: christian.desrosiers@etsmtl.ca

Hendrik Drachslers

Centre for Learning Sciences and Technologies (CELSTEC), Open Universiteit
Nederland

e-mail: hendrik.drachslers@ou.nl

Boi Faltings

Artificial Intelligence Laboratory, School of Computer and Communication
Sciences

Swiss Federal Institute of Technology in Lausanne (EPFL), CH-1015, Lausanne,
Switzerland

e-mail: boi.faltings@epfl.ch

Alexander Felfernig
Graz University of Technology
e-mail: alexander.felfernig@ist.tugraz.at

Gerhard Friedrich
University Klagenfurt
e-mail: gerhard.friedrich@uni-klu.ac.at

Marco de Gemmis
Department of Computer Science, University of Bari “Aldo Moro”, Via E. Orabona,
4, Bari (Italy)
e-mail: degemmis@di.uniba.it

Ulrike Gretzel
Texas A&M University, 2261 TAMU, College Station, TX, USA,
e-mail: ugretzel@tamu.edu

Patrick J.F. Groenen
Econometric Institute, Erasmus University Rotterdam, The Netherlands,
e-mail: groenen@ese.eur.nl

Asela Gunawardana
Microsoft Research, One Microsoft Way, Redmond, WA,
e-mail: aselag@microsoft.com

Andreas Hotho
Knowledge & Data Engineering Group (KDE), University of Kassel, Wilhelmshö,
her Allee 73, 34121 Kassel, Germany,
e-mail: hotho@cs.uni-kassel.de

Hans Hummel
Centre for Learning Sciences and Technologies (CELSTEC), Open Universiteit
Nederland
e-mail: hans.hummel@ou.nl

Neil J. Hurley
School of Computer Science and Informatics, University College Dublin, Ireland
e-mail: neil.hurley@ucd.ie

Robert Jäschke
Knowledge & Data Engineering Group (KDE), University of Kassel, Wilhelmshö
her Allee 73, 34121 Kassel, Germany,
e-mail: jaeschke@cs.uni-kassel.de

Alejandro Jaimes
Yahoo! Research, Av.Diagonal, 177, Barcelona 08018, Spain
e-mail: ajaimes@yahoo-inc.com

Simon James
School of Information Technology, Deakin University, 221 Burwood Hwy,
Burwood 3125, Australia,
e-mail: sjames@deakin.edu.au

Dietmar Jannach
TU Dortmund
e-mail: dietmar.jannach@tu-dortmund.de

Martijn Kagie
Econometric Institute, Erasmus University Rotterdam, The Netherlands,
e-mail: martijn@kagie.net

Dain Kaplan
Tokyo Institute of Technology, Tokyo, Japan
e-mail: dain@cl.cs.titech.ac.jp

George Karypis
Department of Computer Science & Engineering, University of Minnesota,
Minneapolis, USA
e-mail: karypis@cs.umn.edu

Rob Koper
Centre for Learning Sciences and Technologies (CELSTEC), Open Universiteit
Nederland
e-mail: rob.koper@ou.nl

Yehuda Koren
Yahoo! Research,
e-mail: yehuda@yahoo-inc.com

YoungOk Kwon
Department of Information and Decision Sciences
Carlson School of Management, University of Minnesota, Minneapolis, MN 55455,
USA
e-mail: kwonx052@umn.edu

Pasquale Lops
Department of Computer Science, University of Bari "Aldo Moro", Via E. Orabona,
4, Bari (Italy)
e-mail: lops@di.uniba.it

Nikos Manouselis
Greek Research and Technology Network (GRNET S.A.)
56 Messogeion Av., 115 27, Athens, Greece
e-mail: nikosm@gnet.gr

Leandro Balby Marinho
Information Systems and Machine Learning Lab (ISMLL), University of
Hildesheim, Marienburger Platz 22, 31141 Hildesheim, Germany,
e-mail: marinho@ismll.uni-hildesheim.de

Judith Masthoff

University of Aberdeen, AB24 3UE Aberdeen UK,

e-mail: j.masthoff@abdn.ac.uk

Lorraine McGinty

UCD School of Computer Science and Informatics, University College Dublin,
Dublin 4, Ireland.

e-mail: lorraine.mcginty@ucd.ie

Kevin Mercer

Loughborough University,

e-mail: K.C.Mercer@lboro.ac.uk

Alexandros Nanopoulos

Information Systems and Machine Learning Lab (ISMLL), University of
Hildesheim, Marienburger Platz 22, 31141 Hildesheim, Germany,

e-mail: nanopoulos@ismll.uni-hildesheim.de

Michael P. O'Mahony

CLARITY: Centre for Sensor Web Technologies, School of Computer Science and
Informatics, University College Dublin, Ireland

e-mail: michael.p.omahony@ucd.ie

Nuria Oliver

Telefonica Research, Via Augusta, 122, Barcelona 08021, Spain

e-mail: nuriao@tid.es

Jérôme Picault

Alcatel-Lucent Bell Labs,

e-mail: jerome.picault@alcatel-lucent.com

Pearl Pu

Human Computer Interaction Group, School of Computer and Communication
Sciences,

Swiss Federal Institute of Technology in Lausanne (EPFL), CH-1015, Lausanne,
Switzerland

e-mail: pearl.pu, li.chen, jiyong.zhang@epfl.ch

Josep M. Pujol

Telefonica Research, Via Augusta, 122, Barcelona 08021, Spain

e-mail: jmpps@tid.es

Maryam Ramezani

Center for Web Intelligence, College of Computing and Digital Media, 243 S.
Wabash Ave., DePaul University, Chicago, Illinois, USA

e-mail: mramezani@depaul.edu

James Reilly

Google Inc., 5 Cambridge Center, Cambridge, MA 02142, United States.

e-mail: jamesreilly@google.com

Myriam Ribière
Alcatel-Lucent Bell Labs,
e-mail: myriam.ribiere@alcatel-lucent.com

Francesco Ricci
Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
e-mail: fricci@unibz.it

Lior Rokach
Department of Information Systems Engineering, Ben-Gurion University of the
Negev, Israel
e-mail: liorrk@bgu.ac.il

Neil Rubens
University of Electro-Communications, Tokyo, Japan,
e-mail: rubens@hrstc.org

Lars Schmidt-Thieme
Information Systems and Machine Learning Lab (ISMLL), University of
Hildesheim, Marienburger Platz 22, 31141 Hildesheim, Germany,
e-mail: schmidt-thieme@ismll.uni-hildesheim.de

Giovanni Semeraro
Department of Computer Science, University of Bari “Aldo Moro”, Via E. Orabona,
4, Bari (Italy)
e-mail: semeraro@di.uniba.it

Guy Shani
Department of Information Systems Engineering, Ben-Gurion University of the
Negev, Beer-Sheva, Israel
e-mail: shanigu@bgu.ac.il

Bracha Shapira
Department of Information Systems Engineering, Ben-Gurion University of the
Negev, Israel
e-mail: bshapira@bgu.ac.il

Barry Smyth
CLARITY: Centre for Sensor Web Technologies, School of Computer Science &
Informatics, University College Dublin, Ireland,
e-mail: Barry.Smyth@ucd.ie

Gerd Stumme
Knowledge & Data Engineering Group (KDE), University of Kassel, Wilhelmshö
her Allee 73, 34121 Kassel, Germany,
e-mail: stumme@cs.uni-kassel.de

Masashi Sugiyama
Tokyo Institute of Technology, Tokyo, Japan
e-mail: sugi@cs.titech.ac.jp

Panagiotis Symeonidis

Department of Informatics, Aristotle University, 54124 Thessaloniki, Greece,
e-mail: symeon@csd.auth.gr

Nava Tintarev

University of Aberdeen, Aberdeen, U.K,
e-mail: n.tintarev@abdn.ac.uk

Roberto Turrin

Politecnico di Milano, p.zza Leonardo da Vinci 32, Milano, Italy Neptun, via
Durando 10, Milano, Italy
e-mail: roberto.turrin@polimi.it

Alexander Tuzhilin

Department of Information, Operations and Management Sciences
Stern School of Business, New York University
e-mail: atuzhili@stern.nyu.edu

Paolo Viappiani

Department of Computer Science, University of Toronto, 6 King's College Road,
M5S3G4, Toronto, ON, CANADA
e-mail: paolo.viappiani@gmail.com

Patricia Victor

Dept. of Applied Mathematics and Computer Science, Ghent University, Krijgslaan
281 (S9), 9000 Gent, Belgium
e-mail: Chris.Cornelis@ugent.be

Riina Vuorikari

European Schoolnet (EUN), 24, Rue Paul Emile Janson, 1050 Brussels, Belgium
e-mail: riina.vuorikari@eun.org

Michiel van Wezel

Econometric Institute, Erasmus University Rotterdam, The Netherlands,
e-mail: mvanwezel@acm.org

Kyung-Hyan Yoo

William Paterson University, Communication Department, 300 Pompton Road,
Wayne, NJ, USA,
e-mail: toinette75@gmail.com

Markus Zanker

University Klagenfurt
e-mail: markus.zanker@uni-klu.ac.at

Jiyong Zhang

Human Computer Interaction Group, School of Computer and Communication
Sciences,
Swiss Federal Institute of Technology in Lausanne (EPFL), CH-1015, Lausanne,
Switzerland
e-mail: jiyong.zhang@epfl.ch

Chapter 1

Introduction to Recommender Systems Handbook

Francesco Ricci, Lior Rokach and Bracha Shapira

Abstract Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user. In this introductory chapter we briefly discuss basic RS ideas and concepts. Our main goal is to delineate, in a coherent and structured way, the chapters included in this handbook and to help the reader navigate the extremely rich and detailed content that the handbook offers.

1.1 Introduction

Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [60, 85, 25]. The suggestions relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read.

“Item” is the general term used to denote what the system recommends to users. A RS normally focuses on a specific type of item (e.g., CDs, or news) and accordingly its design, its graphical user interface, and the core recommendation technique used to generate the recommendations are all customized to provide useful and effective suggestions for that specific type of item.

RSs are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially overwhelming number of alter-

Francesco Ricci

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy e-mail: fricci@unibz.it

Lior Rokach

Department of Information Systems Engineering, Ben-Gurion University of the Negev, Israel e-mail: liorrk@bgu.ac.il

Bracha Shapira

Department of Information Systems Engineering, Ben-Gurion University of the Negev, Israel e-mail: bshapira@bgu.ac.il

native items that a Web site, for example, may offer [85]. A case in point is a book recommender system that assists users to select a book to read. In the popular Web site, Amazon.com, the site employs a RS to personalize the online store for each customer [47]. Since recommendations are usually personalized, different users or user groups receive diverse suggestions. In addition there are also non-personalized recommendations. These are much simpler to generate and are normally featured in magazines or newspapers. Typical examples include the top ten selections of books, CDs etc. While they may be useful and effective in certain situations, these types of non-personalized recommendations are not typically addressed by RS research.

In their simplest form, personalized recommendations are offered as ranked lists of items. In performing this ranking, RSs try to predict what the most suitable products or services are, based on the user's preferences and constraints. In order to complete such a computational task, RSs collect from users their preferences, which are either explicitly expressed, e.g., as ratings for products, or are inferred by interpreting user actions. For instance, a RS may consider the navigation to a particular product page as an implicit sign of preference for the items shown on that page.

RSs development initiated from a rather simple observation: individuals often rely on recommendations provided by others in making routine, daily decisions [60, 70]. For example it is common to rely on what one's peers recommend when selecting a book to read; employers count on recommendation letters in their recruiting decisions; and when selecting a movie to watch, individuals tend to read and rely on the movie reviews that a film critic has written and which appear in the newspaper they read.

In seeking to mimic this behavior, the first RSs applied algorithms to leverage recommendations produced by a community of users to deliver recommendations to an active user, i.e., a user looking for suggestions. The recommendations were for items that similar users (those with similar tastes) had liked. This approach is termed collaborative-filtering and its rationale is that if the active user agreed in the past with some users, then the other recommendations coming from these similar users should be relevant as well and of interest to the active user.

As e-commerce Web sites began to develop, a pressing need emerged for providing recommendations derived from filtering the whole range of available alternatives. Users were finding it very difficult to arrive at the most appropriate choices from the immense variety of items (products and services) that these Web sites were offering.

The explosive growth and variety of information available on the Web and the rapid introduction of new e-business services (buying products, product comparison, auction, etc.) frequently overwhelmed users, leading them to make poor decisions. The availability of choices, instead of producing a benefit, started to decrease users' well-being. It was understood that while choice is good, more choice is not always better. Indeed, choice, with its implications of freedom, autonomy, and self-determination can become excessive, creating a sense that freedom may come to be regarded as a kind of misery-inducing tyranny [96].

RSs have proved in recent years to be a valuable means for coping with the information overload problem. Ultimately a RS addresses this phenomenon by pointing

a user towards new, not-yet-experienced items that may be relevant to the users current task. Upon a user's request, which can be articulated, depending on the recommendation approach, by the user's context and need, RSs generate recommendations using various types of knowledge and data about users, the available items, and previous transactions stored in customized databases. The user can then browse the recommendations. She may accept them or not and may provide, immediately or at a next stage, an implicit or explicit feedback. All these user actions and feedbacks can be stored in the recommender database and may be used for generating new recommendations in the next user-system interactions.

As noted above, the study of recommender systems is relatively new compared to research into other classical information system tools and techniques (e.g., databases or search engines). Recommender systems emerged as an independent research area in the mid-1990s [35, 60, 70, 7]. In recent years, the interest in recommender systems has dramatically increased, as the following facts indicate:

1. Recommender systems play an important role in such highly rated Internet sites as Amazon.com, YouTube, Netflix, Yahoo, Tripadvisor, Last.fm, and IMDb. Moreover many media companies are now developing and deploying RSs as part of the services they provide to their subscribers. For example Netflix, the online movie rental service, awarded a million dollar prize to the team that first succeeded in improving substantially the performance of its recommender system [54].
2. There are dedicated conferences and workshops related to the field. We refer specifically to ACM Recommender Systems (RecSys), established in 2007 and now the premier annual event in recommender technology research and applications. In addition, sessions dedicated to RSs are frequently included in the more traditional conferences in the area of data bases, information systems and adaptive systems. Among these conferences are worth mentioning ACM SIGIR Special Interest Group on Information Retrieval (SIGIR), User Modeling, Adaptation and Personalization (UMAP), and ACM's Special Interest Group on Management Of Data (SIGMOD).
3. At institutions of higher education around the world, undergraduate and graduate courses are now dedicated entirely to RSs; tutorials on RSs are very popular at computer science conferences; and recently a book introducing RSs techniques was published [48].
4. There have been several special issues in academic journals covering research and developments in the RS field. Among the journals that have dedicated issues to RS are: AI Communications (2008); IEEE Intelligent Systems (2007); International Journal of Electronic Commerce (2006); International Journal of Computer Science and Applications (2006); ACM Transactions on Computer-Human Interaction (2005); and ACM Transactions on Information Systems (2004).

In this introductory chapter we briefly discuss basic RS ideas and concepts. Our main goal is not much to present a self-contained comprehensive introduction or survey on RSs but rather to delineate, in a coherent and structured way, the chapters

included in this handbook and to help the reader navigate the extremely rich and detailed content that the handbook offers.

The handbook is divided into five sections: techniques; applications and evaluation of RSs; interacting with RSs; RSs and communities; and advanced algorithms.

The first section presents the techniques most popularly used today for building RSs, such as collaborative filtering; content-based, data mining methods; and context-aware methods.

The second section surveys techniques and approaches that have been utilized to evaluate the quality of the recommendations. It also deals with the practical aspects of designing recommender systems; describes design and implementation considerations; and sets guidelines for selecting the more suitable algorithms. The section also considers aspects that may affect RS design (domain, device, users, etc.). Finally, it discusses methods, challenges and measures to be applied in evaluating the developed systems.

The third section includes papers dealing with a number of issues related to how recommendations are presented, browsed, explained and visualized. The techniques that make the recommendation process more structured and conversational are discussed here.

The fourth section is fully dedicated to a rather new topic, exploiting user-generated content (UGC) of various types (tags, search queries, trust evaluations, etc.) to generate innovative types of recommendations and more credible ones. Despite its relative newness, this topic is essentially rooted in the core idea of a collaborative recommender,

The last selection presents papers on various advanced topics, such as: the exploitation of active learning principles to guide the acquisition of new knowledge; suitable techniques for protecting a recommender system against attacks of malicious users; and RSs that aggregate multiple types of user feedbacks and preferences to build more reliable recommendations.

1.2 Recommender Systems Function

In the previous section we defined RSs as software tools and techniques providing users with suggestions for items a user may wish to utilize. Now we want to refine this definition illustrating a range of possible roles that a RS can play. First of all, we must distinguish between the role played by the RS on behalf of the service provider from that of the user of the RS. For instance, a travel recommender system is typically introduced by a travel intermediary (e.g., Expedia.com) or a destination management organization (e.g., Visitfinland.com) to increase its turnover (Expedia), i.e., sell more hotel rooms, or to increase the number of tourists to the destination [86]. Whereas, the user's primary motivations for accessing the two systems is to find a suitable hotel and interesting events/attractions when visiting a destination.

In fact, there are various reasons as to why service providers may want to exploit this technology:

- *Increase the number of items sold.* This is probably the most important function for a commercial RS, i.e., to be able to sell an additional set of items compared to those usually sold without any kind of recommendation. This goal is achieved because the recommended items are likely to suit the user's needs and wants. Presumably the user will recognize this after having tried several recommendations¹. Non-commercial applications have similar goals, even if there is no cost for the user that is associated with selecting an item. For instance, a content network aims at increasing the number of news items read on its site.
In general, we can say that from the service provider's point of view, the primary goal for introducing a RS is to increase the conversion rate, i.e., the number of users that accept the recommendation and consume an item, compared to the number of simple visitors that just browse through the information.
- *Sell more diverse items.* Another major function of a RS is to enable the user to select items that might be hard to find without a precise recommendation. For instance, in a movie RS such as Netflix, the service provider is interested in renting all the DVDs in the catalogue, not just the most popular ones. This could be difficult without a RS since the service provider cannot afford the risk of advertising movies that are not likely to suit a particular user's taste. Therefore, a RS suggests or advertises unpopular movies to the right users
- *Increase the user satisfaction.* A well designed RS can also improve the experience of the user with the site or the application. The user will find the recommendations interesting, relevant and, with a properly designed human-computer interaction, she will also enjoy using the system. The combination of effective, i.e., accurate, recommendations and a usable interface will increase the user's subjective evaluation of the system. This in turn will increase system usage and the likelihood that the recommendations will be accepted.
- *Increase user fidelity.* A user should be loyal to a Web site which, when visited, recognizes the old customer and treats him as a valuable visitor. This is a normal feature of a RS since many RSs compute recommendations, leveraging the information acquired from the user in previous interactions, e.g., her ratings of items. Consequently, the longer the user interacts with the site, the more refined her user model becomes, i.e., the system representation of the user's preferences, and the more the recommender output can be effectively customized to match the user's preferences.
- *Better understand what the user wants.* Another important function of a RS, which can be leveraged to many other applications, is the description of the user's preferences, either collected explicitly or predicted by the system. The service provider may then decide to re-use this knowledge for a number of other goals such as improving the management of the item's stock or production. For instance, in the travel domain, destination management organizations can decide to advertise a specific region to new customer sectors or advertise a particular

¹ This issue, convincing the user to accept a recommendation, is discussed again when we explain the difference between predicting the user interest in an item and the likelihood that the user will select the recommended item.

type of promotional message derived by analyzing the data collected by the RS (transactions of the users).

We mentioned above some important motivations as to why e-service providers introduce RSs. But users also may want a RS, if it will effectively support their tasks or goals. Consequently a RS must balance the needs of these two players and offer a service that is valuable to both.

Herlocker et al. [25], in a paper that has become a classical reference in this field, define eleven popular tasks that a RS can assist in implementing. Some may be considered as the main or core tasks that are normally associated with a RS, i.e., to offer suggestions for items that may be useful to a user. Others might be considered as more “opportunistic” ways to exploit a RS. As a matter of fact, this task differentiation is very similar to what happens with a search engine. Its primary function is to locate documents that are relevant to the user’s information need, but it can also be used to check the importance of a Web page (looking at the position of the page in the result list of a query) or to discover the various usages of a word in a collection of documents.

- *Find Some Good Items*: Recommend to a user some items as a ranked list along with predictions of how much the user would like them (e.g., on a one- to five-star scale). This is the main recommendation task that many commercial systems address (see, for instance, Chapter 9). Some systems do not show the predicted rating.
- *Find all good items*: Recommend all the items that can satisfy some user needs. In such cases it is insufficient to just find some good items. This is especially true when the number of items is relatively small or when the RS is mission-critical, such as in medical or financial applications. In these situations, in addition to the benefit derived from carefully examining all the possibilities, the user may also benefit from the RS ranking of these items or from additional explanations that the RS generates.
- *Annotation in context*: Given an existing context, e.g., a list of items, emphasize some of them depending on the user’s long-term preferences. For example, a TV recommender system might annotate which TV shows displayed in the electronic program guide (EPG) are worth watching (Chapter 18 provides interesting examples of this task).
- *Recommend a sequence*: Instead of focusing on the generation of a single recommendation, the idea is to recommend a sequence of items that is pleasing as a whole. Typical examples include recommending a TV series; a book on RSs after having recommended a book on data mining; or a compilation of musical tracks [99], [39].
- *Recommend a bundle*: Suggest a group of items that fits well together. For instance a travel plan may be composed of various attractions, destinations, and accommodation services that are located in a delimited area. From the point of view of the user these various alternatives can be considered and selected as a single travel destination [87].

- *Just browsing*: In this task, the user browses the catalog without any imminent intention of purchasing an item. The task of the recommender is to help the user to browse the items that are more likely to fall within the scope of the user's interests for that specific browsing session. This is a task that has been also supported by adaptive hypermedia techniques [23].
- *Find credible recommender*: Some users do not trust recommender systems thus they play with them to see how good they are in making recommendations. Hence, some system may also offer specific functions to let the users test its behavior in addition to those just required for obtaining recommendations.
- *Improve the profile*: This relates to the capability of the user to provide (input) information to the recommender system about what he likes and dislikes. This is a fundamental task that is strictly necessary to provide personalized recommendations. If the system has no specific knowledge about the active user then it can only provide him with the same recommendations that would be delivered to an "average" user.
- *Express self*: Some users may not care about the recommendations at all. Rather, what it is important to them is that they be allowed to contribute with their ratings and express their opinions and beliefs. The user satisfaction for that activity can still act as a leverage for holding the user tightly to the application (as we mentioned above in discussing the service provider's motivations).
- *Help others*: Some users are happy to contribute with information, e.g., their evaluation of items (ratings), because they believe that the community benefits from their contribution. This could be a major motivation for entering information into a recommender system that is not used routinely. For instance, with a car RS, a user, who has already bought her new car is aware that the rating entered in the system is more likely to be useful for other users rather than for the next time she will buy a car.
- *Influence others*: In Web-based RSs, there are users whose main goal is to explicitly influence other users into purchasing particular products. As a matter of fact, there are also some malicious users that may use the system just to promote or penalize certain items (see Chapter 25).

As these various points indicate, the role of a RS within an information system can be quite diverse. This diversity calls for the exploitation of a range of different knowledge sources and techniques and in the next two sections we discuss the data a RS manages and the core technique used to identify the right recommendations.

1.3 Data and Knowledge Sources

RSs are information processing systems that actively gather various kinds of data in order to build their recommendations. Data is primarily about the items to suggest and the users who will receive these recommendations. But, since the data and knowledge sources available for recommender systems can be very diverse, ultimately, whether they can be exploited or not depends on the recommendation

technique (see also section 1.4). This will become clearer in the various chapters included in this handbook (see in particular Chapter 11).

In general, there are recommendation techniques that are knowledge poor, i.e., they use very simple and basic data, such as user ratings/evaluations for items (Chapters 5, 4). Other techniques are much more knowledge dependent, e.g., using ontological descriptions of the users or the items (Chapter 3), or constraints (Chapter 6), or social relations and activities of the users (Chapter 19). In any case, as a general classification, data used by RSs refers to three kinds of objects: items, users, and transactions, i.e., relations between users and items.

Items. Items are the objects that are recommended. Items may be characterized by their complexity and their value or utility. The value of an item may be positive if the item is useful for the user, or negative if the item is not appropriate and the user made a wrong decision when selecting it. We note that when a user is acquiring an item she will always incur in a cost, which includes the cognitive cost of searching for the item and the real monetary cost eventually paid for the item.

For instance, the designer of a news RS must take into account the complexity of a news item, i.e., its structure, the textual representation, and the time-dependent importance of any news item. But, at the same time, the RS designer must understand that even if the user is not paying for reading news, there is always a cognitive cost associated to searching and reading news items. If a selected item is relevant for the user this cost is dominated by the benefit of having acquired a useful information, whereas if the item is not relevant the net value of that item for the user, and its recommendation, is negative. In other domains, e.g., cars, or financial investments, the true monetary cost of the items becomes an important element to consider when selecting the most appropriate recommendation approach.

Items with low complexity and value are: news, Web pages, books, CDs, movies. Items with larger complexity and value are: digital cameras, mobile phones, PCs, etc. The most complex items that have been considered are insurance policies, financial investments, travels, jobs [72].

RSs, according to their core technology, can use a range of properties and features of the items. For example in a movie recommender system, the genre (such as comedy, thriller, etc.), as well as the director, and actors can be used to describe a movie and to learn how the utility of an item depends on its features. Items can be represented using various information and representation approaches, e.g., in a minimalist way as a single id code, or in a richer form, as a set of attributes, but even as a concept in an ontological representation of the domain (Chapter 3).

Users. Users of a RS, as mentioned above, may have very diverse goals and characteristics. In order to personalize the recommendations and the human-computer interaction, RSs exploit a range of information about the users. This information can be structured in various ways and again the selection of what information to model depends on the recommendation technique.

For instance, in collaborative filtering, users are modeled as a simple list containing the ratings provided by the user for some items. In a demographic RS, socio-demographic attributes such as age, gender, profession, and education, are used. User data is said to constitute the user model [21, 32]. The user model profiles the

user, i.e., encodes her preferences and needs. Various user modeling approaches have been used and, in a certain sense, a RS can be viewed as a tool that generates recommendations by building and exploiting user models [19, 20]. Since no personalization is possible without a convenient user model, unless the recommendation is non-personalized, as in the top-10 selection, the user model will always play a central role. For instance, considering again a collaborative filtering approach, the user is either profiled directly by its ratings to items or, using these ratings, the system derives a vector of factor values, where users differ in how each factor weights in their model (Chapters 5 and 4).

Users can also be described by their behavior pattern data, for example, site browsing patterns (in a Web-based recommender system) [107], or travel search patterns (in a travel recommender system) [60]. Moreover, user data may include relations between users such as the trust level of these relations between users (Chapter 20). A RS might utilize this information to recommend items to users that were preferred by similar or trusted users.

Transactions. We generically refer to a transaction as a recorded interaction between a user and the RS. Transactions are log-like data that store important information generated during the human-computer interaction and which are useful for the recommendation generation algorithm that the system is using. For instance, a transaction log may contain a reference to the item selected by the user and a description of the context (e.g., the user goal/query) for that particular recommendation. If available, that transaction may also include an explicit feedback the user has provided, such as the rating for the selected item.

In fact, ratings are the most popular form of transaction data that a RS collects. These ratings may be collected explicitly or implicitly. In the explicit collection of ratings, the user is asked to provide her opinion about an item on a rating scale. According to [93], ratings can take on a variety of forms:

- Numerical ratings such as the 1-5 stars provided in the book recommender associated with Amazon.com.
- Ordinal ratings, such as “strongly agree, agree, neutral, disagree, strongly disagree” where the user is asked to select the term that best indicates her opinion regarding an item (usually via questionnaire).
- Binary ratings that model choices in which the user is simply asked to decide if a certain item is good or bad.
- Unary ratings can indicate that a user has observed or purchased an item, or otherwise rated the item positively. In such cases, the absence of a rating indicates that we have no information relating the user to the item (perhaps she purchased the item somewhere else).

Another form of user evaluation consists of tags associated by the user with the items the system presents. For instance, in MovieLens RS (<http://movielens.umn.edu>) tags represent how MovieLens users feel about a movie, e.g.: “too long”, or “acting”. Chapter 19 focuses on these types of transactions.

In transactions collecting implicit ratings, the system aims to infer the users opinion based on the user’s actions. For example, if a user enters the keyword “Yoga” at

Amazon.com she will be provided with a long list of books. In return, the user may click on a certain book on the list in order to receive additional information. At this point, the system may infer that the user is somewhat interested in that book.

In conversational systems, i.e., systems that support an interactive process, the transaction model is more refined. In these systems user requests alternate with system actions (see Chapter 13). That is, the user may request a recommendation and the system may produce a suggestion list. But it can also request additional user preferences to provide the user with better results. Here, in the transaction model, the system collects the various requests-responses, and may eventually learn to modify its interaction strategy by observing the outcome of the recommendation process [60].

1.4 Recommendation Techniques

In order to implement its core function, identifying the useful items for the user, a RS must *predict* that an item is worth recommending. In order to do this, the system must be able to predict the utility of some of them, or at least compare the utility of some items, and then decide what items to recommend based on this comparison. The prediction step may not be explicit in the recommendation algorithm but we can still apply this unifying model to describe the general role of a RS. Here our goal is to provide the reader with a unifying perspective rather than an account of all the different recommendation approaches that will be illustrated in this handbook.

To illustrate the prediction step of a RS, consider, for instance, a simple, non-personalized, recommendation algorithm that recommends just the most popular songs. The rationale for using this approach is that in absence of more precise information about the user's preferences, a popular song, i.e., something that is liked (high utility) by many users, will also be probably liked by a generic user, at least more than another randomly selected song. Hence the utility of these popular songs is predicted to be reasonably high for this generic user.

This view of the core recommendation computation as the prediction of the utility of an item for a user has been suggested in [3]. They model this degree of utility of the user u for the item i as a (real valued) function $R(u, i)$, as is normally done in collaborative filtering by considering the ratings of users for items. Then the fundamental task of a collaborative filtering RS is to predict the value of R over pairs of users and items, i.e., to compute $\hat{R}(u, i)$, where we denote with \hat{R} the estimation, computed by the RS, of the true function R . Consequently, having computed this prediction for the active user u on a set of items, i.e., $\hat{R}(u, i_1), \dots, \hat{R}(u, i_N)$ the system will recommend the items i_{j_1}, \dots, i_{j_K} ($K \leq N$) with the largest predicted utility. K is typically a small number, i.e., much smaller than the cardinality of the item data set or the items on which a user utility prediction can be computed, i.e., RSs "filter" the items that are recommended to users.

As mentioned above, some recommender systems do not fully estimate the utility before making a recommendation but they may apply some heuristics to hypothe-

size that an item is of use to a user. This is typical, for instance, in knowledge-based systems. These utility predictions are computed with specific algorithms (see below) and use various kind of knowledge about users, items, and the utility function itself (see section 1.3) [25]. For instance, the system may assume that the utility function is Boolean and therefore it will just determine whether an item is or is not useful for the user. Consequently, assuming that there is some available knowledge (possibly none) about the user who is requesting the recommendation, knowledge about items, and other users who received recommendations, the system will leverage this knowledge with an appropriate algorithm to generate various utility predictions and hence recommendations [25].

It is also important to note that sometimes the user utility for an item is observed to depend on other variables, which we generically call “contextual” [1]. For instance, the utility of an item for a user can be influenced by the domain knowledge of the user (e.g., expert vs. beginning users of a digital camera), or can depend on the time when the recommendation is requested. Or the user may be more interested in items (e.g., a restaurant) closer to his current location. Consequently, the recommendations must be adapted to these specific additional details and as a result it becomes harder and harder to correctly estimate what the right recommendations are.

This handbook presents several different types of recommender systems that vary in terms of the addressed domain, the knowledge used, but especially in regard to the recommendation algorithm, i.e., how the prediction of the utility of a recommendation, as mentioned at the beginning of this section, is made. Other differences relate to how the recommendations are finally assembled and presented to the user in response to user requests. These aspects are also discussed later in this introduction.

To provide a first overview of the different types of RSs, we want to quote a taxonomy provided by [25] that has become a classical way of distinguishing between recommender systems and referring to them. [25] distinguishes between six different classes of recommendation approaches:

Content-based: The system learns to recommend items that are similar to the ones that the user liked in the past. The similarity of items is calculated based on the features associated with the compared items. For example, if a user has positively rated a movie that belongs to the comedy genre, then the system can learn to recommend other movies from this genre. Chapter 3 provides an overview of content-based recommender systems, imposing some order among the extensive and diverse aspects involved in their design and implementation. It presents the basic concepts and terminology of content-based RSs, their high level architecture, and their main advantages and drawbacks. The chapter then surveys state-of-the-art systems that have been adopted in several application domains. The survey encompasses a thorough description of both classical and advanced techniques for representing items and user profiles. Finally, it discusses trends and future research which might lead towards the next generation of recommender systems.

Collaborative filtering: The simplest and original implementation of this approach [93] recommends to the active user the items that other users with similar tastes liked in the past. The similarity in taste of two users is calculated based on

the similarity in the rating history of the users. This is the reason why [94] refers to collaborative filtering as “people-to-people correlation.” Collaborative filtering is considered to be the most popular and widely implemented technique in RS.

Chapter 4 presents a comprehensive survey of neighborhood-based methods for collaborative filtering. Neighborhood methods focus on relationships between items or, alternatively, between users. An item-item approach models the preference of a user to an item based on ratings of similar items by the same user. Nearest-neighbors methods enjoy considerable popularity due to their simplicity, efficiency, and their ability to produce accurate and personalized recommendations. The authors will address the essential decisions that are required when implementing a neighborhood-based recommender system and provide practical information on how to make such decisions.

Finally, the chapter deals with problems of data sparsity and limited coverage, often observed in large commercial recommender systems. A few solutions to overcome these problems are presented.

Chapter 5 presents several recent extensions available for building CF recommenders. Specifically, the authors discuss latent factor models, such as matrix factorization (e.g., Singular Value Decomposition, SVD). These methods transform both items and users to the same latent factor space. The latent space is then used to explain ratings by characterizing both products and users in term of factors automatically inferred from user feedback. The authors elucidate how SVD can handle additional features of the data, including implicit feedback and temporal information. They also describe techniques to address shortcomings of neighborhood techniques by suggesting more rigorous formulations using global optimization techniques. Utilizing such techniques makes it possible to lift the limit on neighborhood size and to address implicit feedback and temporal dynamics. The resulting accuracy is close to that of matrix factorization models, while offering a number of practical advantages.

Demographic: This type of system recommends items based on the demographic profile of the user. The assumption is that different recommendations should be generated for different demographic niches. Many Web sites adopt simple and effective personalization solutions based on demographics. For example, users are dispatched to particular Web sites based on their language or country. Or suggestions may be customized according to the age of the user. While these approaches have been quite popular in the marketing literature, there has been relatively little proper RS research into demographic systems [59].

Knowledge-based: Knowledge-based systems recommend items based on specific domain knowledge about how certain item features meet users needs and preferences and, ultimately, how the item is useful for the user. Notable knowledge-based recommender systems are case-based [22, 87]. In these systems a similarity function estimates how much the user needs (problem description) match the recommendations (solutions of the problem). Here the similarity score can be directly interpreted as the utility of the recommendation for the user.

Constraint-based systems are another type of knowledge-based RSs (Chapter 6). In terms of used knowledge, both systems are similar: user requirements are col-

lected; repairs for inconsistent requirements are automatically proposed in situations where no solutions could be found; and recommendation results are explained. The major difference lies in the way solutions are calculated. Case-based recommenders determine recommendations on the basis of similarity metrics whereas constraint-based recommenders predominantly exploit predefined knowledge bases that contain explicit rules about how to relate customer requirements with item features.

Knowledge-based systems tend to work better than others at the beginning of their deployment but if they are not equipped with learning components they may be surpassed by other shallow methods that can exploit the logs of the human/computer interaction (as in CF).

Community-based: This type of system recommends items based on the preferences of the users' friends. This technique follows the epigram "Tell me who your friends are, and I will tell you who you are". [8, 14]. Evidence suggests that people tend to rely more on recommendations from their friends than on recommendations from similar but anonymous individuals [103]. This observation, combined with the growing popularity of open social networks, is generating a rising interest in community-based systems or, as or as they usually referred to, social recommender systems [34]. This type of RSs models and acquires information about the social relations of the users and the preferences of the user's friends. The recommendation is based on ratings that were provided by the user's friends. In fact these RSs are following the rise of social-networks and enable a simple and comprehensive acquisition of data related to the social relations of the users.

The research in this area is still in its early phase and results about the systems performance are mixed. For example, [34, 64] report that overall, social-network-based recommendations are no more accurate than those derived from traditional CF approaches, except in special cases, such as when user ratings of a specific item are highly varied (i.e. controversial items) or for cold-start situations, i.e., where the users did not provide enough ratings to compute similarity to other users. Others have showed that in some cases social-network data yields better recommendations than profile similarity data [37] and that adding social network data to traditional CF improves recommendation results [36]. The chapter 20 provides a survey of the findings in this field and analyzes current results.

Hybrid recommender systems: These RSs are based on the combination of the above mentioned techniques. A hybrid system combining techniques A and B tries to use the advantages of A to fix the disadvantages of B. For instance, CF methods suffer from new-item problems, i.e., they cannot recommend items that have no ratings. This does not limit content-based approaches since the prediction for new items is based on their description (features) that are typically easily available. Given two (or more) basic RSs techniques, several ways have been proposed for combining them to create a new hybrid system (see [25] for the precise descriptions).

As we have already mentioned, the context of the user when she is seeking a recommendation can be used to better personalize the output of the system. For example, in a temporal context, vacation recommendations in winter should be very different from those provided in summer. Or a restaurant recommendation for a

Saturday evening with your friends should be different from that suggested for a workday lunch with co-workers.

Chapter 7 presents the general notion of context and how it can be modeled in RSs. Discussing the possibilities of combining several context-aware recommendation techniques into a single unified approach, the authors also provide a case study of one such combined approach.

Three different algorithmic paradigms for incorporating contextual information into the recommendation process are discussed: reduction-based (pre-filtering), contextual post filtering, and context modeling. In reduction-based (pre-filtering) methods, only the information that matches the current usage context, e.g., the ratings for items evaluated in the same context, are used to compute the recommendations. In contextual post filtering, the recommendation algorithm ignores the context information. The output of the algorithm is filtered/adjusted to include only the recommendations that are relevant in the target context. In the contextual modeling, the more sophisticated of the three approaches, context data is explicitly used in the prediction model.

1.5 Application and Evaluation

Recommender system research is being conducted with a strong emphasis on practice and commercial applications, since, aside from its theoretical contribution, is generally aimed at practically improving commercial RSs. Thus, RS research involves practical aspects that apply to the implementation of these systems. These aspects are relevant to different stages in the life cycle of a RS, namely, the design of the system, its implementation and its maintenance and enhancement during system operation.

The aspects that apply to the design stage include factors that might affect the choice of the algorithm. The first factor to consider, the application's domain, has a major effect on the algorithmic approach that should be taken. [72] provide a taxonomy of RSs and classify existing RS applications to specific application domains. Based on these specific application domains, we define more general classes of domains for the most common recommender systems applications:

- Entertainment - recommendations for movies, music, and IPTV.
- Content - personalized newspapers, recommendation for documents, recommendations of Web pages, e-learning applications, and e-mail filters.
- E-commerce - recommendations for consumers of products to buy such as books, cameras, PCs etc.
- Services - recommendations of travel services, recommendation of experts for consultation, recommendation of houses to rent, or matchmaking services.

As recommender systems become more popular, interest is aroused in the potential advantages in new applications, such as recommending friends or tweets to

follow as in www.tweeter.com. Hence, the above list cannot cover all the application domains that are now being addressed by RS techniques; it gives only an initial description of the various types of application domains.

The developer of a RS for a certain application domain should understand the specific facets of the domain, its requirements, application challenges and limitations. Only after analyzing these factors one could be able to select the optimal recommender algorithm and to design an effective human-computer interaction.

Chapter 11 of this handbook provides guidelines for matching the application domain to the recommendation technique. Burke and Ramezani in their chapter provide a new classification of recommender systems. Unlike former classifications of RSs (such as [25, 94, 3, 7]), Burke and Ramezani take an AI-centric approach, and focus on the knowledge sources required for different recommendation approaches, and the constraints related to them as a primer guideline to choosing the algorithm. The chapter discusses the applicability of various recommendation techniques for different types of problems and suggests decision-making guidelines in selecting these techniques.

The chapter explicitly aims at system implementers as “recommenders” for the right recommendation approach. The authors describe the knowledge sources that are available to a recommender systems in different domains and identify what knowledge sources are required for each recommendation technique. This implies that the design of a recommender system should first emphasize the analysis of the available sources of knowledge, and then decide about the algorithm accordingly.

Another example of the need to adjust the recommender approach to the domain is described in Chapter 12, which deals with recommender systems for technology-enhanced learning (TEL). TEL, which generally covers technologies that support all forms of teaching and learning activities, aims at designing, developing and testing new methods and technologies to enhance learning practices of both individuals and organizations. TEL may benefit greatly from integrating recommender systems technology to personalize the learning process and adjust it to the user’s former knowledge, abilities and preferences. The chapter presents the particular requirements of RSs for TEL; the user tasks that are supported in TEL settings; and how these tasks compare to typical user tasks in other RSs. For example, one particular user task for TEL – “find novel resources” – attempts to recommend only new or novel items. Or, to cite another example, – “find new pathways” – is concerned with recommending alternative pathways through the learning resources. The chapter presents an analysis of the filtering approaches that could be useful for TEL along with a survey of existing TEL systems illustrating the recommendation techniques that have been deployed in these systems.

Chapter 10 discusses practical aspects of RS development and aims at providing practical guidelines to the design, implementation and evaluation of personalized systems. Besides the prediction algorithm, many other factors need to be considered when designing a RS. Chapter 10 lists some of these elements: the type of target users and their context; the devices that they would use; the role of the recommendation within the application; the goal of the recommendation; and, as mentioned previously, the data that is available.

The authors propose to build a model of the environment based on three dimensions: system users; the characteristics of the data; and the overall application. The recommender system design will be based on this model. The authors illustrate their guidelines and the model on a news recommendation system that they have developed.

Another important issue related to the practical side of RS deployment is the necessity of evaluating them. Evaluation is required at different stages of the systems life cycle for various purposes [25, 1]. At design time, evaluation is required to verify the selection of the appropriate recommender approach. In the design phase, evaluation should be implemented off-line and the recommendation algorithms are compared with user interactions. The off-line evaluation consists of running several algorithms on the same datasets of user interactions (e.g., ratings) and comparing their performance. This type of evaluation is usually conducted on existing public benchmark data if appropriate data is available, or, otherwise, on collected data. The design of the off-line experiments should follow known experiment design practices [11] in order to ensure reliable results.

Evaluation is also required after the system has been launched. The algorithms might be very accurate in solving the core recommendation problem, i.e., predicting user ratings, but for some other reason the system may not be accepted by users, e.g., because the performance of the system is not as expected. At this stage it is usually beneficial to perform on-line evaluation with real users of the system and analyze system logs in order to enhance system performance. In addition, most of the algorithms include parameters, such as weights thresholds, the number of neighbors, etc., requiring constant adjustment and calibration.

Another type of evaluation is a focused user study that can be conducted when the on-line evaluation is not feasible or too risky. In this type of evaluation, a controlled experiment is planned where a small group of users are asked to perform different tasks with various versions of the system. It is then possible to analyze the users performance and to distribute questionnaires so that users may report on their experience. In such experiments it is possible to collect both quantitative and qualitative information about the systems.

Evaluation is also discussed in Chapter 12 in the context of TEL systems. The authors provide a detailed analysis of the evaluation methods and tools that can be employed for evaluating TEL recommendation techniques against a set of criteria that are proposed for each of the selected components (e.g., user model, domain model, recommendation strategy and algorithm).

Chapter 8 details three types of experiments that can be conducted in order to evaluate recommender systems. It presents their advantages and disadvantages, and defines guidelines for choosing the methods for evaluation them. Unlike existing discussions of evaluation in the literature that usually speaks about the accuracy of an algorithms prediction [25] and related measures, this chapter is unique in its approach to the evaluation discussion since it focuses on property-directed evaluation. It provides a large set of properties (other than accuracy) that are relevant to the systems success. For each of the properties, the appropriate type of experiment and

relevant measures are suggested. Among the list of properties are: coverage, cold start, confidence, trust, novelty, risk, and serendipity.

When discussing the practical aspects of RSs, it may be beneficial to analyze real system implementations. The idea is to test theoretically intuitive assumptions in order to determine if they work in practice. The major problem that one must face in this case comes from the fact that the owners of commercial RSs are generally unwilling to reveal their practices and there are only relatively few opportunities for such cooperation.

Chapter 9 reports on such an opportunity and describes the operation of a real RS, illustrating the practical aspects that apply to the implementation stage of the RS development and its evaluation. This description focuses on the integration of a RS into the production environment of Fastweb, one of the largest European IP Television (IPTV) providers. The chapter describes the requirements and considerations, including scaling and accuracy, that led to the choice of the recommender algorithms. It also describes the off-line and on-line evaluations that took place and illustrates how the system is adjusted accordingly.

1.6 Recommender Systems and Human Computer Interaction

As we have illustrated in previous sections, researchers have chiefly been concerned with designing a range of technical solutions, leveraging various sources of knowledge to achieve better predictions about what is liked and how much by the target user. The underlying assumption behind this research activity is that just presenting these correct recommendations, i.e., the best options, should be enough. In other words, the recommendations should speak for themselves, and the user should definitely accept the recommendations if they are correct. This is clearly an overly simplified account of the recommendation problem and it is not so easy to deliver recommendations.

In practice, users need recommendations because they do not have enough knowledge to make an autonomous decision. Consequently, it may not be easy for them to evaluate the proposed recommendation. Hence, various researchers have tried to understand the factors that lead to the acceptance of a recommendation by a given user [105, 30, 24, 97, 33].

[105] was among the first to point out that the effectiveness of a RS is dependent on factors that go beyond the quality of the prediction algorithm. In fact, the recommender must also convince users to try (or read, buy, listen, watch) the recommended items. This, of course, depends on the individual characteristics of the selected items and therefore on the recommendation algorithm. The process also depends, however, on the particular human/computer interaction supported by the system when the items are presented, compared, and explained. [105] found that from a users perspective, an effective recommender system must inspire trust in the system; it must have a system logic that is at least somewhat transparent; it should point users towards new, not-yet-experienced items; it should provide details

about recommended items, including pictures and community ratings; and finally, it should present ways to refine recommendations.

[105] and other similarly oriented researchers do not diminish the importance of the recommendation algorithm, but claim that its effectiveness should not be evaluated only in terms of the accuracy of the prediction, i.e., with standard and popular IR metrics, such as MAE (Mean Absolute Error), precision, or NDCG (Normalized Discounted Cumulative Gain) (see also Chapters 8 5, 9). Other dimensions should be measured that relate to the acceptance of the recommender system and its recommendations. These ideas have been remarkably well presented and discussed also by [33]. In that work the authors propose user-centric directions for evaluating recommender systems, including: the similarity of recommendation lists, recommendation serendipity, and the importance of user needs and expectations in a recommender.

Following the remarks made in [105], let us introduce some important points raised by HCI research that are further discussed in this handbook.

1.6.1 Trust, Explanations and Persuasiveness

First of all let us focus on trust. There are two different notions of trust that are discussed in this handbook: trust about the other users of the recommender and trust about a system's recommendations.

Chapter 20 focuses on the first notion and considers a class of recommender systems termed "social recommender systems". These systems attempt to generate more useful recommendations derived from information about user profiles and relationships between users that nowadays can be found virtually everywhere; e.g. in social networking sites such as Facebook, LinkedIn and MySpace. Since trust-based recommender systems mainly exploit the trust relationships found in these social networking sites to build new recommendation algorithms (e.g., [34]), they still operate on the core rating prediction problem but use trust relationships. The main claimed advantage is that users will be aware of the nature of the recommendations, i.e., how they have been identified, and will tend to place greater trust in these recommendations. In other words, the mutual trust of users can be exploited also for increasing the trust in the system.

Trust in system recommendations is discussed in Chapter 15. In this chapter the main scope is actually the role of explanations in RSs and trust emerges as one out of seven roles that can be played by explanations in RSs. These roles are: transparency - explaining how the system works; scrutability - allowing users to tell the system it is wrong [50]; trust - increasing user confidence in the system; effectiveness - helping users make good decisions; persuasiveness - convincing users to try or buy; efficiency - helping users make decisions faster; and satisfaction - increasing the ease of use or enjoyment.

This chapter also illustrates a range of approaches for building explanations. In the collaborative filtering style, i.e., the explanation is of the form "Other users similar to you liked this item". In content-based style explanations, the item's attributes

which most affected the item to be recommended to the user are illustrated. For example, in a movie recommendation, an explanation may be of the form “This movie was recommended because it stars Bruce Willis who you seem to like”, or “Item X was recommended because of features A and B which are shared by items Y and Z, which you rated highly”. In case-based style explanations, the system refers to items that are similar to the recommended one, for example, “The item was recommended because you said you own item X” or “These items are recommended based on your most recently viewed items”. And finally, in knowledge-based style explanations, the system explains the differences between the recommended item and another item and how it serves the user’s goal: “This room has an ocean view and is larger than the previous recommended room, which will make it more romantic as you requested”.

Moving back to trust, we see that it serves as a means of obtaining the main goal of the recommender, i.e., to convince the user to accept the recommendations and try out one of the recommended items. This issue is ultimately related to the persuasiveness of the full RS, i.e., how the various elements of the RS, including what and how an item is recommended, actually operate during the human/computer interaction. This topic is discussed in the Chapter 14. Here the authors stress that a recommendation is seen as credible advice and is actually taken into account not only because of the user’s perceptions of the recommendation but also due to the fundamental role of the system which is perceived as an advice-giver. Indeed, the literature about persuasion suggests that people are likely to accept recommendations from credible sources and we therefore conclude that the credibility of the RS is vital to increasing the likelihood of recommendation acceptance. Hence, the authors discuss how the credibility of RSs can be enhanced, providing a synopsis of credibility-related research.

1.6.2 Conversational Systems

Another severe limitation of many algorithmic approaches to RSs is due to the fact that these algorithms have been designed to collect all the input data only once. They then terminate their job by returning their recommendations. In many cases, this model is not effective since users may not be fully aware of their preferences until they have interacted to a certain extent with the system and roughly understand the range of alternatives. Or they may want to browse several alternative options before being convinced that some of the recommendations may suit them. There is also the possibility that the system may be initially wrong in its suggestions and the user may be willing to provide additional information that can fix these problems, and eventually obtain some better recommendations.

These aspects have been stressed and tackled by researchers engaged in following a line of research that is commonly known as “conversational RSs” [27, 110, 67, 60]. Conversational RSs use a diverse range of techniques for rating prediction or ranking. However, they all try to support an interactive process where both the user and

the system may query or provide information to the other partner. The critical issue here is how to design the dialogue, i.e., the conversational strategy and what actions the user and the system must perform in the various stages of the interaction. The supported dialogue must be effective, i.e., the user should terminate the conversation with a solution of the task (e.g., book a flight) and in a quick way (small number of conversational steps). In this handbook two chapters deal with this important topic.

Chapter 13 provides a comprehensive account of the research conducted in critiquing-based systems. Critiquing-based interfaces, or dialogue models, given an initial set of user preferences (e.g., preferred values for some item features) present to the user recommended items and support the user in formulating “critiques”, such as “Show me more like item A, but cheaper”.

Critiquing-based systems have attracted great interest in domains where there is a need for more sophisticated and interactive decision/recommendation support systems, such as in travel applications [88, 32, 100], or computer systems [82, 83]. Critiquing-based systems were initially designed as effective approaches to user preference elicitation problems, but have now become important for some additional motivations or applications, such as group recommendations, mixed-initiative recommendations, adaptive user interface, recommendation explanation, mobile recommenders.

Another approach related to conversational systems is preference-based [67]. Preference-based are similar to critiquing-based approaches since they present upfront the user with some recommendations, which are not considered to be the best but then let the user express preferences about some items. This additional information is used to refine the system representation of the user’s preferences (user model) enabling the system to generate new and better recommendations.

Chapter 16 surveys these novel methods and systems focusing on three facets of the user-system interaction of such preference-based recommenders: initial preference elicitation; preference revision; and presentation of recommendation results. This chapter derives from the analysis of some systems as a collection of usability guidelines that can be applied in a wide and scalable way. Moreover, to select the guidelines, the authors do not focus on accuracy alone, but take into account that humans have limited cognitive resources and are not likely to achieve a high level of accuracy if the required effort is excessive. They identify and select methods that produce high recommendation accuracy involving an effort level that users are willing to make.

Previously mentioned approaches (critiquing- and preference-based) have been mostly applied to case-based reasoning systems [22], where the retrieval component is based on a similarity metric. In such cases, a query can always retrieve and rank all the products contained in the catalogue since a product is always, to some extent, similar to a probe product (query). If the query language supports other constraints (e.g. equality or range constraints) the query may fail to return a product satisfying the query [47, 71, 31]. In this case several techniques have been proposed for repairing the query by relaxing the minimum amount of constraints to make it satisfiable. This topic is also covered in a chapter dedicated to constraint-based RSs (Chapter 6).

1.6.3 Visualization

We have highlighted so far some HCI issues that have been tackled in RS research and which are discussed in this handbook. In summary, we have noted that how the system presents and visualizes the computed recommendation is obviously a critical factor for the acceptance of the recommendations and the RS.

Presentation and explanation techniques are not easily separable; a good presentation technique is also capable of explaining recommendations but also in motivating the user to make further requests, including requests for explanations. One common aspect in the technologies presented so far is the fact that recommendations are presented as a list of items. The length of this list can vary but the output of the core recommendation algorithm is normally a ranked list and this has been always exploited in the presentation.

In this handbook we include a chapter that illustrates a presentation approach that deviates from this paradigm. In Chapter 17 the authors observe that much information is lost in the ranked list visualization approach, since two products, both of which match the user query or the user model, can differ from each other based on a completely different set of product characteristics. If one is using a two dimensional, map-based visualization of the recommendations, it is possible to retain part of this information. In the map, one can position, in a restricted area of the map, recommendations that are similar to each other. This chapter presents two approaches for building this two-dimensional map of the recommendations and discusses its advantages and disadvantages.

1.7 Recommender Systems as a Multi-Disciplinary Field

Designing and developing RSs is a multi-disciplinary effort that has benefited from results obtained in various computer science fields especially machine learning and data mining, information retrieval, and human-computer interaction. This is also clear in the chapters included in this handbook and the discussion presented above. Here we want to briefly address these relationships.

Machine learning and data mining, subfields of artificial intelligence, allow a computer to learn to optimally perform a certain task using examples, data or past experiences [109]. For example, data mining can be used to learn from transaction data that customers who bought “Da Vinci Code” also bought “The Five People You Meet in Heaven”. Consequently, recommendations can be constructed using the information provided by these associations.

Many RSs are centered around the use of various machine learning and data mining algorithms to predict user evaluations for items, or for learning how to correctly rank items for a user. Chapter 2 of this handbook provides an overview of the main data mining techniques used in the context of RSs preprocessing methods, such as: sampling or dimensionality reduction; classification techniques, such as Bayesian

networks and support vector machines; clustering techniques such as k-means algorithm; and association rules.

Other chapters that illustrate and exemplify the relationships between RSs and data mining are: Chapter 12, discussing the usage of active learning for selective information acquisition; Chapter 5, devoted to advanced optimization techniques for building rating prediction models; Chapter 7, presenting various rating prediction methods that exploit contextually tagged transactional data; Chapter 24, presenting data mining techniques that exploit the evaluations of items over several criteria to better predict the overall user evaluations; Chapter 25, focusing on data mining solutions to detect attacks to a recommender system and for building more robust algorithmic solutions; Chapter 4, illustrating various instance based-learning options currently used in collaborative filtering systems; Chapter 19 illustrating the use of data mining solutions operating on a multiway array or a hypergraph with hyperedges, i.e., (user, resource, tag) triples; Chapter 20 presenting various data mining solutions on trust networks.

Information retrieval (IR) aims to assist users in storing and searching various forms of content, such as texts, images and videos [63]. With IR tools, users can quickly find information that is both relevant and comprehensive for their needs. While IR did not begin with the Web, the WWW played an important role in establishing new ideas mainly due to the development of Web search engines.

Both IR and RSs are faced with similar filtering and ranking problems. IR generally focuses on developing global retrieval techniques, often neglecting the individual needs and preferences of users. Still [25] argues that recommender systems are not clearly separated from information retrieval. The “individualized” and “interesting and useful” criteria that RSs try to achieve are the core differences between RSs and information retrieval or search engines.

Recently, modern Web search engine have also relied on recommendation techniques to address Web search challenges and to implement advanced search features. For example, search engines recommend similar queries to the current user query. Various engines also attempt to apply some form of personalization by generating results to a user query that are not only relevant to the query terms but are also tailored to the users context (e.g., her location), and her search history.

Chapter 18 discusses the research goals of IR and personalized Web search from the RS perspective. The authors illustrate how techniques that originated in recent RS research may be applied to address search engine challenges. The chapter focuses on two promising ideas for search engines improvement: personalization and collaboration. The chapter describes a number of different approaches to personalizing Web searches by exploiting user preferences and context information to affect search results. In addition, the chapter discusses recent work in the area of collaborative information retrieval, which attempts to take advantage of the potential for cooperation between friends, colleagues or users with similar needs in implementing a variety of information-seeking tasks. This new line of research, termed social search, benefits from the social medium property of the Web in providing search results that are affected by the experience and preferences of similar users. The authors foresee a “convergence of recommender systems and search systems” and

believe that integrating these sources in search engine algorithms would result in highly satisfied users receiving the right information at the right time.

Other chapters that are related to IR research and illustrate techniques that are studied in this area include: Chapter 19, addressing problems related to the retrieval of tag-based information content and Chapter 3, presenting an overview of content-based approaches that are strongly rooted in current search engine technologies.

Finally, RSs are ultimately targeted to provide useful information to users and for that reason HCI plays a fundamental role in the ultimate acceptance of the computed recommendations. In fact, several field studies have clearly indicated that from a user's perspective, HCI aspects related to the usability of the system have a tremendous effect on the willingness of users to actually explore a systems recommendations and provide input to the system in return for more effective recommendations. These topics were discussed previously in Section 1.6.

1.8 Emerging Topics and Challenges

1.8.1 Emerging Topics Discussed in the Handbook

It is clear from the previous pages that RS research is evolving in many and diverse directions and new topics are emerging or becoming more important subjects of investigation. The reader is also referred to the proceedings of the last editions of the ACM RecSys conferences and several other excellent review papers for additional material [7, 3]. In this handbook we cover some of these topics. Indeed, several have been already presented, such as: context-aware recommender (Chapter 7); new visualization techniques (Chapter 17); community-based personalized search (Chapter 18); trust-based RS (Chapter 20). Other important topics are covered in the last two sections of this handbook and we want now to briefly introduce these chapters.

Chapter 19 presents social tagging systems (STS) a new RS-related topic that is emerging due to the growth of Web 2.0 applications. STS like Flickr, Bibsonomy, or Delicious, allow the ordinary user to publish and edit content as well as generate and share tags (i.e., free keywords). STS users are experiencing information overload problems since STS are used by millions of users who enter into the system uncontrolled content and tags that pose retrieving difficulties for traditional IR systems. Thus, RSs are required to assist users in finding relevant Information and some commercial STS are starting to offer recommendations (e.g., Delicious).

The chapter discusses the new challenges that RSs for STS face, such as new recommender tasks. These include not only traditional recommendations regarding content, but also recommendations for relevant tags and even other relevant users. Tag recommendation (i.e., recommending to the users relevant tags for an item), has different characteristics than traditional recommendations since the system can recommend recurrent tags, unlike traditional RSs that usually do not recommend the same item twice. In addition, RSs for STS deal with a three-dimensional prob-

lem (user, resource, tag), rather than the traditional two-dimensional problem (user, item), and this affects the complexity of the algorithms. The chapter includes a state-of-the-art survey about the new generation of RSs built to serve STS. It also details the challenges of deploying RS for real world STS, and offers new algorithms for dealing with the challenges of content in both STS and tag recommendation.

Chapter 21 deals with those situations when it would be good if the system could recommend information or items that are relevant to a group of users rather than to an individual. For instance, a RS may select television programs for a group to view or a sequence of songs to listen to, based on models of all group members. Recommending to groups is clearly more complicated than recommending to individuals. Assuming that we know precisely what is good for individual users, the issue is how to combine individual user models. In this chapter, the authors discuss how group recommendation works, what its problems are, and what advances have been made so far.

Chapter 22 discusses the ubiquitous issue of aggregating preferences, criteria or similarities. Normally such aggregation is done by using either the arithmetic mean or maximum/minimum functions. But many other aggregation functions which could deliver flexibility and adaptability, and ultimately more relevant recommendations, are often overlooked. In this chapter the authors review the basics of aggregation functions and their properties and present the most important families, including generalized means, Choquet and Sugeno integrals, ordered weighted averaging, triangular norms and conorms, as well as bipolar aggregation functions. Such functions can model various interactions between the inputs, including conjunctive, disjunctive and mixed behavior.

In Chapter 23, the authors focus on another fundamental problem of RSs, i.e., the need to actively look for new data during the operational life of the recommender. This issue is normally neglected on the assumption that there is not much space for controlling what data (e.g., ratings) the system can collect since these decisions are taken by the users when visiting the system. Actually, the RS provokes the users with its recommendations and many systems actually explicitly ask for user preferences during the recommendation process. Hence, by tuning the process, users can be pushed to provide a range of different information. Specifically they can be requested to rate particular items and the knowledge of the users opinions about these items could be estimated as particularly beneficial according to various criteria, e.g., to provide more diverse recommendations or simply to improve the prediction accuracy of the system for some users or for the whole population of users. At this point active learning comes in; it can augment RSs, helping users to become more self-aware of their own likes/dislikes, leading to more meaningful and useful questions. At the same time active learning can provide new information to the system that can be analyzed for subsequent recommendations. Hence, applying active learning to RSs enables personalization of the recommending process [61]. This is accomplished by allowing the system to actively influence the items the user is exposed to (e.g. the items displayed to the user during sign-up or during regular use), as well as by enabling the user to explore his/her interests freely.

Chapter 24 introduces another emerging topic, i.e., multi-criteria recommender systems. In the majority of RSs the utility associated with an item is usually considered a single criterion value, e.g., an overall evaluation or rating of an item by a user. But recently this assumption has been judged as limited because the suitability of the recommended item for a particular user may depend on several aspects that the user can take into consideration when making his or her choice. The incorporation of multiple criteria that can affect the users opinions may lead to more effective and accurate recommendations.

Chapter 24 provides an overview of multi-criteria RSs. First, it defines the recommendation problem as a multi-criteria decision-making problem and reviews methods and techniques that can support the implementation of multi-criteria recommenders. Then, it focuses on the category of multi-criteria rating recommender techniques that provide recommendations by modeling the users utility for an item as a vector of ratings along several criteria. A review of current algorithms that use multi-criteria ratings for calculating the rating prediction and generating recommendations is provided. The chapter concludes with a discussion on open issues and future challenges for these recommenders.

The last chapter of this handbook (Chapter 25) surveys articles dealing with security issues. This topic has become a major issue in the past few years. Recent works on the topic include [28, 45, 102, 112]. The chapter analyzes algorithms designed to generate more robust recommendations, i.e., recommendations that are harder for malicious users to influence. In fact, collaborative recommender systems are dependent on the goodwill of their users, i.e., there is an implicit assumption that users will interact with the system with the aim of getting good recommendations for themselves while providing useful data for their neighbors. However, users will have a range of purposes in interacting with RSs and in some cases, these purposes may be counter to those of the system owner or those of the majority of its user population. Namely these users may want to damage the Web site hosting the recommender or to influence the recommendations provided to visitors, e.g., to score some items better or worse rather than to arrive at a fair evaluation.

In this chapter the authors provide a model of efficient attacks, i.e., attacks that can, with relatively low cost, produce a large impact on system output. Since these attacks may very well be launched against a site, it makes sense to detect them so that countermeasures can be taken as soon as possible. At the same time, researchers have studied a number of algorithms that are intended to robustly withstand attacks and which have lower impact curves relative to efficient attacks. These approaches are also surveyed in this chapter. With the combination of these techniques, researchers have sought, not to eliminate attacks, but to control their impact to the point where they are no longer cost-effective.

1.8.2 Challenges

The list of newly emerging and challenging RS research topics is not limited to those described in the chapters that we have mentioned above. Moreover, covering all of them is not within the scope of this short introduction. The reader is referred to the final discussion sections in this handbook for other outstanding problems.

Below we briefly note additional challenging topics that we consider important for the development of the research on RSs and which are not covered in the handbook.

- Scalability of the algorithms with large and real-world datasets. As the research on core techniques progresses and matures, it becomes clear that a fundamental issue for RSs is to determine how to embed the core recommendation techniques in real operational systems and how to deal with massive and dynamic sets of data produced by the interactions of users with items (ratings, preferences, reviews, etc.). A solution that works fine when tested off-line on relatively small data sets may become inefficient or even totally inapplicable on very large datasets. New approaches and large-scale evaluation studies are needed [91, 92, 33, 38, 116, 75, 75].
- Proactive recommender systems, i.e., recommenders that decide to provide recommendations even if not explicitly requested [90, 24, 62, 80]. The largest majority of the recommender systems developed so far follow a “pull” model [94]; where the user originates the request for a recommendation. In the scenarios emerging today, where computers are ubiquitous and users are always connected, it seems natural to imagine that a RS can detect implicit requests. It therefore needs to predict not only what to recommend, but also when and how to “push” its recommendations. In this way the RS can become proactive without being perceived as disturbing.
- Privacy preserving recommender systems [81, 26, 79, 56, 17, 28, 102, 16, 5, 53, 70, 114]. RSs exploit user data to generate personalized recommendations. In the attempt to build increasingly better recommendations, they collect as much user data as possible. This will clearly have a negative impact on the privacy of the users and the users may start feeling that the system knows too much about their true preferences. Therefore, there is a need to design solutions that will parsimoniously and sensibly use user data. At the same time these solutions will ensure that knowledge about the users cannot be freely accessed by malicious users.
- Diversity of the items recommended to a target user [104, 66, 69, 55, 54, 46, 119]. In a recommendation list, it is more likely that the user will find a suitable item if there is a certain degree of diversity among the included items. There is often no value in having perfect recommendations for a restricted type of product, unless the user has expressed a narrow set of preferences. There are many situations, especially in the early stage of a recommendation process, in which the users want to explore new and diverse directions. In such cases, the user is using the recommender as a knowledge discovery tool. The research on this topic is still in an

early stage, and there is a need to characterize the nature of this “diversity”, i.e., whether we are looking for diversity among different recommendation sessions or within a session, and how to combine the diversity goal with the accuracy of the recommendation.

- Integration of long-term and short-term user preferences in the process of building a recommendation list [6, 40, 74]. Recommender systems may be divided in two classes: those that build a long-term profile, generated by aggregating all the user transaction data collected by the system (e.g., collaborative filtering) and those that are more focused on capturing the ephemeral preferences of the user, e.g., as in case-based approaches. Obviously both aspects are important and either the precise user task or the availability of items may come under consideration in resolving the preference integration problem. In fact, new research is required to build hybrid models that can correctly decide to drift or not toward the contingent user’s preferences when there is enough evidence to suggest that the user’s short-term preferences are departing from the long-term ones.
- Generic user models and cross domain recommender systems are able to mediate user data through different systems and application domains [41, 18, 52, 19, 20, 49, 15]. Using generic user model techniques, a single RS can produce recommendations about a variety of items. This is normally not possible for a general RS which can combine more techniques in a hybrid approach, but cannot easily benefit from user preferences collected in one domain to generate recommendations in a different one.
- Distributed recommender systems that operate in open networks [38, 116, 92, 113, 17, 102]. The computational model of the largest majority of RSs adheres to a typical client-server architecture, where the user-client requests recommendations to the server-recommender which replies with the suggestions. This is clearly a severe limitation and suffers from all the classical problems of centralized systems. The emerging scenario of grid or cloud computing can become an excellent opportunity to implement more robust and flexible computational models for RSs.
- Recommender that optimize a sequence of recommendations [120, 99, 10, 59, 61, 107, 106]. We mentioned already that conversational RSs have emerged in the attempt to improve the quality of recommendations provided by the systems based on a simpler approach: a one-time request/response. Conversational RSs can be further improved by implementing learning capabilities that can optimize not only the items that are recommended but also how the dialogue between the user and the system must unfold in all possible situations.
- Recommenders designed to operate in mobile devices and usage contexts [117, 98, 55, 51, 4, 115, 111, 57, 29, 9, 77, 76, 89, 73, 44, 95, 13]. Mobile computing is emerging as the most natural platform for personal computing. Many recommendation requests are likely to be made when the user is on the move, e.g., at shops or hotels in a visited city. This necessitates “mobilizing” the user interface and to design computational solutions that can efficiently use the still limited resources (computational power and screen size) of the mobile devices.

Finally, before ending this introduction, we want to present some additional challenges that were discussed in a tutorial held at the latest RecSys conference in New York, October 22-25, 2009 [<http://recsys.acm.org/tutorial3.pdf>]. John Riedl (University of Minnesota), Todd Beaupre (Yahoo!) and John Sanders (Netflix) mentioned eight important challenges for the research on recommender systems: transparency, exploration versus exploitation, guided navigation, time value, user action interpretation, evaluating recommenders, scalability, academic/industry partnerships.

Some of these issues have already been discussed in this introduction. For example, transparency was introduced when we discussed the role of the explanation of a recommendation, and we stressed the important role it plays in order to present a recommendation as more acceptable for the user. Also the evaluation of RSs, i.e., the range of possible and important dimensions that can be measured in an evaluation is a topic fully addressed in another chapter (Chapter 8).

The time value of recommendations is also partially discussed in our remarks about context-aware recommenders (Chapter 7). However, the challenge refers to the fact that a given set of recommendations may not be applicable forever but there could be a time interval when these items can be recommended. This is clear, for instance, when it comes to news items; people want to be informed about the most recent events and news cannot be meaningfully recommended even one day after the initial announcement.

Exploration vs. exploitation is touched upon in active learning (Chapter 23). This challenge refers to the fundamental dilemma that a designer must properly tackle, i.e., whether to keep recommending items that the system can now identify as good recommendations, given the data currently available for the system or to further explore user preferences (e.g., asking to rate additional and particular items) in order to build newer and possibly even better recommendations in the future.

indexGuided navigation Guided navigation refers to combining classical recommendation lists, i.e., suggestions with tools that let the user navigate more autonomously in the space of possible options. User action interpretation refers to the possibility that in addition to explicit ratings there could be many more actions performed by the user operating the recommender that can be detected, analyzed and used to build a better prediction model. The idea is that every single user action should be exploited in the recommendation process. But it is challenging to interpret the user's actions, i.e., the intent behind an action, and there are actions that should be discarded because they were not produced by genuine users, such as, actions performed by different users on the same browser, or false and malicious registrations or data or log data caused by robots or crawlers.

Scalability was also mentioned earlier. We stress again that this is clearly an issue about which discussion is missing in the current literature since it has been mostly investigated by practitioners.

Finally the discussion in that workshop became quite animated when the matter of cooperation between industry and academia was touched upon. Industry has specific problems but is not making them clearly visible. This is happening for many reasons, including the need to not disclose to competitors critical information. Con-

versely, academia is looking for problems that can be tackled in a framework of the resources and time available to them and will generally address a topic only if it is likely to have an impact in the scientific community. This has made and will make industry-academic cooperation difficult. But RSs is a research field that requires new concrete challenges and there is a real risk of stagnation if we fail to tackle the useful but risky challenges in favor of solved or mature problems.

We hope that this handbook, as a useful tool for practitioners and researchers, will contribute to further develop knowledge in this exciting and useful research area. In this way we believe that we can reduce the risk that these two groups will follow different roads. Currently the research on RSs has greatly benefited from the combined interest and efforts that industry and academia have invested in this field. We therefore wish the best to both groups as they read this handbook and we hope that it will attract even more researchers to work in this highly interesting and challenging field.

References

1. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.* **23**(1), 103–145 (2005)
2. Adomavicius, G., Tuzhilin, A.: Personalization technologies: a process-oriented perspective. *Commun. ACM* **48**(10), 83–90 (2005)
3. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749 (2005)
4. Ahn, H., Kim, K.J., Han, I.: Mobile advertisement recommender system using collaborative filtering: Mar-cf. In: *Proceedings of the 2006 Conference of the Korea Society of Management Information Systems*, pp. 709–715 (2006)
5. Aïmeur, E., Brassard, G., Fernandez, J.M., Onana, F.S.M.: Alambic : a privacy-preserving recommender system for electronic commerce. *Int. J. Inf. Sec.* **7**(5), 307–334 (2008)
6. Aïmeur, E., Vézeau, M.: Short-term profiling for a case-based reasoning recommendation system. In: R.L. de Mántaras, E. Plaza (eds.) *Machine Learning: 2000, 11th European Conference on Machine Learning*, pp. 23–30. Springer (2000)
7. Anand, S.S., Mobasher, B.: Intelligent techniques for web personalization. In: *Intelligent Techniques for Web Personalization*, pp. 1–36. Springer (2005)
8. Arazy, O., Kumar, N., Shapira, B.: Improving social recommender systems. *IT Professional* **11**(4), 38–44 (2009)
9. Averjanova, O., Ricci, F., Nguyen, Q.N.: Map-based interaction with a conversational mobile recommender system. In: *The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08*, pp. 212–218 (2008)
10. Baccigalupo, C., Plaza, E.: Case-based sequential ordering of songs for playlist recommendation. In: T. Roth-Berghofer, M.H. Göker, H.A. Güvenir (eds.) *ECCBR, Lecture Notes in Computer Science*, vol. 4106, pp. 286–300. Springer (2006)
11. Bailey, R.A.: *Design of comparative experiments*. Cambridge University Press Cambridge (2008)
12. Balabanovic, M., Shoham, Y.: Content-based, collaborative recommendation. *Communication of ACM* **40**(3), 66–72 (1997)
13. Bellotti, V., Begole, J.B., hsin Chi, E.H., Ducheneaut, N., Fang, J., Isaacs, E., King, T.H., Newman, M.W., Partridge, K., Price, B., Rasmussen, P., Roberts, M., Schiano, D.J., Walen-

- dowski, A.: Activity-based serendipitous recommendations with the magitti mobile leisure guide. In: M. Czerwinski, A.M. Lund, D.S. Tan (eds.) CHI, pp. 1157–1166. ACM (2008)
14. Ben-Shimon, D., Tsikinovsky, A., Rokach, L., Meisels, A., Shani, G., Naamani, L.: Recommender system from personal social networks. In: K. Wegrzyn-Wolska, P.S. Szczepaniak (eds.) AWIC, *Advances in Soft Computing*, vol. 43, pp. 47–55. Springer (2007)
 15. Berkovsky, S.: Mediation of User Models: for Enhanced Personalization in Recommender Systems. VDM Verlag (2009)
 16. Berkovsky, S., Borisov, N., Eytani, Y., Kuflik, T., Ricci, F.: Examining users' attitude towards privacy preserving collaborative filtering. In: International Workshop on Data Mining for User Modeling, at User Modeling 2007, 11th International Conference, UM 2007, Corfu, Greece, June 25, 2007, Proceedings (2007)
 17. Berkovsky, S., Eytani, Y., Kuflik, T., Ricci, F.: Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 9–16. ACM Press, New York, NY, USA (2007)
 18. Berkovsky, S., Kuflik, T., Ricci, F.: Cross-technique mediation of user models. In: Proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems [AH2006], pp. 21–30. Dublin (2006)
 19. Berkovsky, S., Kuflik, T., Ricci, F.: Mediation of user models for enhanced personalization in recommender systems. *User Modeling and User-Adapted Interaction* **18**(3), 245–286 (2008)
 20. Berkovsky, S., Kuflik, T., Ricci, F.: Cross-representation mediation of user models. *User Modeling and User-Adapted Interaction* **19**(1-2), 35–63 (2009)
 21. Billsus, D., Pazzani, M.: Learning probabilistic user models. In: UM97 Workshop on Machine Learning for User Modeling (1997). URL <http://www.dfki.de/~bauer/um-ws/>
 22. Bridge, D., Göker, M., McGinty, L., Smyth, B.: Case-based recommender systems. *The Knowledge Engineering review* **20**(3), 315–320 (2006)
 23. Brusilovsky, P.: Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction* **6**(2-3), 87–129 (1996)
 24. Bulander, R., Decker, M., Schiefer, G., Kolmel, B.: Comparison of different approaches for mobile advertising. *Mobile Commerce and Services*, 2005. WMCS '05. The Second IEEE International Workshop on pp. 174–182 (2005)
 25. Burke, R.: Hybrid web recommender systems. In: *The Adaptive Web*, pp. 377–408. Springer Berlin / Heidelberg (2007)
 26. Canny, J.F.: Collaborative filtering with privacy. In: *IEEE Symposium on Security and Privacy*, pp. 45–57 (2002)
 27. Carenini, G., Smith, J., Poole, D.: Towards more conversational and collaborative recommender systems. In: *Proceedings of the 2003 International Conference on Intelligent User Interfaces*, January 12-15, 2003, Miami, FL, USA, pp. 12–18 (2003)
 28. Cheng, Z., Hurley, N.: Effective diverse and obfuscated attacks on model-based recommender systems. In: *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pp. 141–148. ACM, New York, NY, USA (2009)
 29. Church, K., Smyth, B., Cotter, P., Bradley, K.: Mobile information access: A study of emerging search behavior on the mobile internet. *ACM Trans. Web* **1**(1), 4 (2007)
 30. Cosley, D., Lam, S.K., Albert, I., Konstant, J.A., Riedl, J.: Is seeing believing? how recommender system interfaces affect users' opinions. In: *Proceedings of the CHI 2003 Conference on Human factors in Computing Systems*. Fort Lauderdale, FL (2003)
 31. Felfernig, A., Friedrich, G., Schubert, M., Mandl, M., Mairitsch, M., Teppan, E.: Plausible repairs for inconsistent requirements. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 791–796. Pasadena, California, USA (2009)
 32. Fisher, G.: User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction* **11**, 65–86 (2001)
 33. George, T., Merugu, S.: A scalable collaborative filtering framework based on co-clustering. In: *Proceedings of the 5th IEEE Conference on Data Mining (ICDM)*, pp. 625–628. IEEE Computer Society, Los Alamitos, CA, USA (2005)

34. Golbeck, J.: Generating predictive movie recommendations from trust in social networks. In: Trust Management, 4th International Conference, iTrust 2006, Pisa, Italy, May 16-19, 2006, Proceedings, pp. 93–104 (2006)
35. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. *Commun. ACM* **35**(12), 61–70 (1992)
36. Groh, G., Ehlig, C.: Recommendations in taste related domains: collaborative filtering vs. social filtering. In: GROUP '07: Proceedings of the 2007 international ACM conference on Supporting group work, pp. 127–136. ACM, New York, NY, USA (2007)
37. Guy, I., Zwerdling, N., Carmel, D., Ronen, I., Uziel, E., Yogev, S., Ofek-Koifman, S.: Personalized recommendation of social software items based on social relations. In: RecSys '09: Proceedings of the third ACM conference on Recommender systems, pp. 53–60. ACM, New York, NY, USA (2009)
38. Han, P., Xie, B., Yang, F., Sheng, R.: A scalable p2p recommender system based on distributed collaborative filtering. *Expert systems with applications* (2004)
39. Hayes, C., Cunningham, P.: Smartradio-community based music radio. *Knowledge Based Systems* **14**(3-4), 197–201 (2001)
40. He, L., Zhang, J., Zhuo, L., Shen, L.: Construction of user preference profile in a personalized image retrieval. In: Neural Networks and Signal Processing, 2008 International Conference on, pp. 434–439 (2008)
41. Heckmann, D., Schwartz, T., Brandherm, B., Schmitz, M., von Wilamowitz-Moellendorff, M.: Gumo - the general user model ontology. In: User Modeling 2005, 10th International Conference, UM 2005, Edinburgh, Scotland, UK, July 24-29, 2005, Proceedings, pp. 428–432 (2005)
42. Herlocker, J., Konstan, J., Riedl, J.: Explaining collaborative filtering recommendations. In: In proceedings of ACM 2000 Conference on Computer Supported Cooperative Work, pp. 241–250 (2000)
43. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Transaction on Information Systems* **22**(1), 5–53 (2004)
44. Horozov, T., Narasimhan, N., Vasudevan, V.: Using location for personalized POI recommendations in mobile environments. In: Proc. Int'l Sym. Applications on Internet, pp. 124–129. IEEE Computer Society (2006)
45. Hurley, N., Cheng, Z., Zhang, M.: Statistical attack detection. In: RecSys '09: Proceedings of the third ACM conference on Recommender systems, pp. 149–156. ACM, New York, NY, USA (2009)
46. Hwang, C.S., Kuo, N., Yu, P.: Representative-based diversity retrieval. In: Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on, pp. 155–155 (2008)
47. Jannach, D.: Finding preferred query relaxations in content-based recommenders. In: 3rd International IEEE Conference on Intelligent Systems, pp. 355–360 (2006)
48. Jannach, D., Zanker, M., Felfernig, A., Friedrich, G.: *Recommender Systems An Introduction*. Cambridge University Press (2010)
49. Jessenitschnig, M., Zanker, M.: A generic user modeling component for hybrid recommendation strategies. *E-Commerce Technology, IEEE International Conference on* **0**, 337–344 (2009). DOI <http://doi.ieeecomputersociety.org/10.1109/CEC.2009.83>
50. Kay, J.: Scrutable adaptation: Because we can and must. In: Adaptive Hypermedia and Adaptive Web-Based Systems, 4th International Conference, AH 2006, Dublin, Ireland, June 21-23, 2006, Proceedings, pp. 11–19 (2006)
51. Kim, C.Y., Lee, J.K., Cho, Y.H., Kim, D.H.: Viscors: A visual-content recommender for the mobile web. *IEEE Intelligent Systems* **19**(6), 32–39 (2004)
52. Kobsa, A.: Generic user modeling systems. In: P. Brusilovsky, A. Kobsa, W. Nejdl (eds.) *The Adaptive Web, Lecture Notes in Computer Science*, vol. 4321, pp. 136–154. Springer (2007)
53. Kobsa, A.: Privacy-enhanced personalization. In: D. Wilson, H.C. Lane (eds.) *FLAIRS Conference*, p. 10. AAAI Press (2008)
54. Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. *IEEE Computer* **42**(8), 30–37 (2009)

55. Kramer, R., Modsching, M., ten Hagen, K.: Field study on methods for elicitation of preferences using a mobile digital assistant for a dynamic tour guide. In: SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, pp. 997–1001. ACM Press, New York, NY, USA (2006)
56. Lam, S.K., Frankowski, D., Riedl, J.: Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. In: G. Müller (ed.) *ETRICS, Lecture Notes in Computer Science*, vol. 3995, pp. 14–29. Springer (2006)
57. Lee, H., Park, S.J.: Moners: A news recommender for the mobile web. *Expert Systems with Applications* **32**(1), 143 – 150 (2007)
58. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing* **7**(1), 76–80 (2003)
59. Mahmood, T., Ricci, F.: Towards learning user-adaptive state models in a conversational recommender system. In: A. Hinneburg (ed.) *LWA 2007: Lernen - Wissen - Adaption*, Halle, September 2007, Workshop Proceedings, pp. 373–378. Martin-Luther-University Halle-Wittenberg (2007)
60. Mahmood, T., Ricci, F.: Improving recommender systems with adaptive conversational strategies. In: C. Cattuto, G. Ruffo, F. Menczer (eds.) *Hypertext*, pp. 73–82. ACM (2009)
61. Mahmood, T., Ricci, F., Venturini, A., Höpken, W.: Adaptive recommender systems for travel planning. In: W.H. Peter OConnor, U. Gretzel (eds.) *Information and Communication Technologies in Tourism 2008*, proceedings of ENTER 2008 International Conference, pp. 1–11. Springer, Innsbruck (2008)
62. Mahmoud, Q.: Provisioning context-aware advertisements to wireless mobile users. *Multi-media and Expo, 2006 IEEE International Conference on* pp. 669–672 (2006)
63. Manning, C.: *Introduction to Information Retrieval*. Cambridge University Press, Cambridge (2008)
64. Massa, P., Avesani, P.: Trust-aware collaborative filtering for recommender systems. In: *Proceedings of the International Conference on Cooperative Information Systems, CoopIS*, pp. 492–508 (2004)
65. McCarthy, K., Salamó, M., Coyle, L., McGinty, L., Smyth, B., Nixon, P.: Group recommender systems: a critiquing based approach. In: C. Paris, C.L. Sidner (eds.) *IUI*, pp. 267–269. ACM (2006)
66. McGinty, L., Smyth, B.: On the role of diversity in conversational recommender systems. In: A. Aamodt, D. Bridge, K. Ashley (eds.) *ICCBR 2003, the 5th International Conference on Case-Based Reasoning*, pp. 276–290. Trondheim, Norway (2003)
67. McGinty, L., Smyth, B.: Adaptive selection: An analysis of critiquing and preference-based feedback in conversational recommender systems. *International Journal of Electronic Commerce* **11**(2), 35–57 (2006)
68. McNee, S.M., Riedl, J., Konstan, J.A.: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pp. 1097–1101. ACM Press, New York, NY, USA (2006)
69. McSherry, D.: Diversity-conscious retrieval. In: S. Craw, A. Preece (eds.) *Advances in Case-Based Reasoning, Proceedings of the 6th European Conference on Case Based Reasoning, ECCBR 2002*, pp. 219–233. Springer Verlag, Aberdeen, Scotland (2002)
70. McSherry, F., Mironov, I.: Differentially private recommender systems: building privacy into the net. In: *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 627–636. ACM, New York, NY, USA (2009)
71. Mirzadeh, N., Ricci, F.: Cooperative query rewriting for decision making support and recommender systems. *Applied Artificial Intelligence* **21**, 1–38 (2007)
72. Montaner, M., López, B., de la Rosa, J.L.: A taxonomy of recommender agents on the internet. *Artificial Intelligence Review* **19**(4), 285–330 (2003)
73. Nguyen, Q.N., Ricci, F.: Replaying live-user interactions in the off-line evaluation of critique-based mobile recommendations. In: *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pp. 81–88. ACM Press, New York, NY, USA (2007)

74. Nguyen, Q.N., Ricci, F.: Conversational case-based recommendations exploiting a structured case model. In: *Advances in Case-Based Reasoning*, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings, pp. 400–414 (2008)
75. Papagelis, M., Rousidis, I., Plexousakis, D., Theoharopoulos, E.: Incremental collaborative filtering for highly-scalable recommendation algorithms. In: M.S. Hacid, N.V. Murray, Z.W. Ras, S. Tsumoto (eds.) *ISMIS, Lecture Notes in Computer Science*, vol. 3488, pp. 553–561. Springer (2005)
76. Park, M.H., Hong, J.H., Cho, S.B.: Location-based recommendation system using bayesian user's preference model in mobile devices. In: J. Indulska, J. Ma, L.T. Yang, T. Ungerer, J. Cao (eds.) *UIC, Lecture Notes in Computer Science*, vol. 4611, pp. 1130–1139. Springer (2007)
77. Park, S., Kang, S., Kim, Y.K.: A channel recommendation system in mobile environment. *Consumer Electronics, IEEE Transactions on* **52**(1), 33–39 (2006). DOI 10.1109/TCE.2006.1605022
78. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review* **13**, 393–408 (1999)
79. Polat, H., Du, W.: Privacy-preserving collaborative filtering using randomized perturbation techniques. In: *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, 19-22 December 2003, Melbourne, Florida, USA, pp. 625–628 (2003)
80. Puerta Melguizo, M.C., Boves, L., Deshpande, A., Ramos, O.M.: A proactive recommendation system for writing: helping without disrupting. In: *ECCE '07: Proceedings of the 14th European conference on Cognitive ergonomics*, pp. 89–95. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1362550.1362569>
81. Ramakrishnan, N., Keller, B.J., Mirza, B.J., Grama, A., Karypis, G.: When being weak is brave: Privacy in recommender systems. *IEEE Internet Computing* **cs.CG/0105028** (2001)
82. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Dynamic critiquing. In: *Advances in Case-Based Reasoning*, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings, pp. 763–777 (2004)
83. Reilly, J., Zhang, J., McGinty, L., Pu, P., Smyth, B.: Evaluating compound critiquing recommenders: a real-user study. In: *EC '07: Proceedings of the 8th ACM conference on Electronic commerce*, pp. 114–123. ACM, New York, NY, USA (2007)
84. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: An open architecture for collaborative filtering of netnews. In: *Proceedings ACM Conference on Computer-Supported Cooperative Work*, pp. 175–186 (1994)
85. Resnick, P., Varian, H.R.: Recommender systems. *Communications of the ACM* **40**(3), 56–58 (1997)
86. Ricci, F.: Travel recommender systems. *IEEE Intelligent Systems* **17**(6), 55–57 (2002)
87. Ricci, F., Cavada, D., Mirzadeh, N., Venturini, A.: Case-based travel recommendations. In: D.R. Fesenmaier, K. Woeber, H. Werthner (eds.) *Destination Recommendation Systems: Behavioural Foundations and Applications*, pp. 67–93. CABI (2006)
88. Ricci, F., Missier, F.D.: Supporting travel decision making through personalized recommendation. In: C.M. Karat, J.O. Blom, J. Karat (eds.) *Designing Personalized User Experiences in eCommerce*, pp. 231–251. Kluwer Academic Publisher (2004)
89. Ricci, F., Nguyen, Q.N.: Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intelligent Systems* **22**(3), 22–29 (2007). DOI <http://doi.ieeecomputersociety.org/10.1109/MIS.2007.43>
90. Sae-Ueng, S., Pinyapong, S., Ogino, A., Kato, T.: Personalized shopping assistance service at ubiquitous shop space. *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on* pp. 838–843 (2008). DOI 10.1109/WAINA.2008.287
91. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Incremental singular value decomposition algorithms for highly scalable recommender systems. In: *Proceedings of the 5th International Conference in Computers and Information Technology* (2002)

92. Sarwar, B.M., Konstan, J.A., Riedl, J.: Distributed recommender systems for internet commerce. In: M. Khosrow-Pour (ed.) *Encyclopedia of Information Science and Technology (II)*, pp. 907–911. Idea Group (2005)
93. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: *The Adaptive Web*, pp. 291–324. Springer Berlin / Heidelberg (2007)
94. Schafer, J.B., Konstan, J.A., Riedl, J.: E-commerce recommendation applications. *Data Mining and Knowledge Discovery* **5**(1/2), 115–153 (2001)
95. Schifanella, R., Panisson, A., Gena, C., Ruffo, G.: Mobhinter: epidemic collaborative filtering and self-organization in mobile ad-hoc networks. In: *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pp. 27–34. ACM, New York, NY, USA (2008)
96. Schwartz, B.: *The Paradox of Choice*. ECCO, New York (2004)
97. van Setten, M., McNee, S.M., Konstan, J.A.: Beyond personalization: the next stage of recommender systems research. In: R.S. Amant, J. Riedl, A. Jameson (eds.) *IUI*, p. 8. ACM (2005)
98. van Setten, M., Pokraev, S., Koolwaaij, J.: Context-aware recommendations in the mobile tourist application compass. In: W. Nejdl, P. De Bra (eds.) *Adaptive Hypermedia 2004*, pp. 235–244. Springer Verlag (2004)
99. Shani, G., Heckerman, D., Brafman, R.I.: An mdp-based recommender system. *Journal of Machine Learning Research* **6**, 1265–1295 (2005)
100. Sharda, N.: *Tourism Informatics: Visual Travel Recommender Systems, Social Communities, and User Interface Design*. Information Science Reference (2009)
101. Shardanand, U., Maes, P.: Social information filtering: algorithms for automating "word of mouth". In: *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*, pp. 210–217 (1995)
102. Shokri, R., Pedarsani, P., Theodorakopoulos, G., Hubaux, J.P.: Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In: *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pp. 157–164. ACM, New York, NY, USA (2009)
103. Sinha, R.R., Swearingen, K.: Comparing recommendations made by online systems and friends. In: *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries* (2001)
104. Smyth, B., McClave, P.: Similarity vs diversity. In: *Proceedings of the 4th International Conference on Case-Based Reasoning*. Springer-Verlag (2001)
105. Swearingen, K., Sinha, R.: Beyond algorithms: An HCI perspective on recommender systems. In: J.L. Herlocker (ed.) *Recommender Systems, papers from the 2001 ACM SIGIR Workshop*. New Orleans, LA - USA (2001)
106. Taghipour, N., Kardan, A.: A hybrid web recommender system based on q-learning. In: *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*, Fortaleza, Ceara, Brazil, March 16-20, 2008, pp. 1164–1168 (2008)
107. Taghipour, N., Kardan, A., Ghidary, S.S.: Usage-based web recommendations: a reinforcement learning approach. In: *Proceedings of the 2007 ACM Conference on Recommender Systems, RecSys 2007*, Minneapolis, MN, USA, October 19-20, 2007, pp. 113–120 (2007)
108. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.* **10**, 623–656 (2009)
109. Tan, P.N.: *Introduction to Data Mining*. Pearson Addison Wesley, San Francisco (2006)
110. Thompson, C.A., Goker, M.H., Langley, P.: A personalized system for conversational recommendations. *Artificial Intelligence Research* **21**, 393–428 (2004)
111. Tung, H.W., Soo, V.W.: A personalized restaurant recommender agent for mobile e-service. In: S.T. Yuan, J. Liu (eds.) *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE'04*, pp. 259–262. IEEE Computer Society Press, Taipei, Taiwan (2004)
112. Van Roy, B., Yan, X.: Manipulation-resistant collaborative filtering systems. In: *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pp. 165–172. ACM, New York, NY, USA (2009)

113. Wang, J., Pouwelse, J.A., Lagendijk, R.L., Reinders, M.J.T.: Distributed collaborative filtering for peer-to-peer file sharing systems. In: H. Haddad (ed.) SAC, pp. 1026–1030. ACM (2006)
114. Wang, Y., Kobsa, A.: Performance evaluation of a privacy-enhancing framework for personalized websites. In: G.J. Houben, G.I. McCalla, F. Pianesi, M. Zancanaro (eds.) UMAP, *Lecture Notes in Computer Science*, vol. 5535, pp. 78–89. Springer (2009)
115. Wietsma, R.T.A., Ricci, F.: Product reviews in mobile decision aid systems. In: Pervasive Mobile Interaction Devices (PERMID 2005) - Mobile Devices as Pervasive User Interfaces and Interaction Devices - Workshop in conjunction with: The 3rd International Conference on Pervasive Computing (PERVASIVE 2005), May 11 2005, Munich, Germany, pp. 15–18. LMU Munich (2005)
116. Xie, B., Han, P., Yang, F., Shen, R.: An efficient neighbor searching scheme of distributed collaborative filtering on p2p overlay network. *Database and Expert Systems Applications* pp. 141–150 (2004)
117. Yuan, S.T., Tsao, Y.W.: A recommendation mechanism for contextualized mobile advertising. *Expert Systems with Applications* **24**(4), 399–414 (2003)
118. Zhang, F.: Research on recommendation list diversity of recommender systems. *Management of e-Commerce and e-Government, International Conference on* pp. 72–76 (2008)
119. Zhang, M.: Enhancing diversity in top-n recommendation. In: RecSys '09: Proceedings of the third ACM conference on Recommender systems, pp. 397–400. ACM, New York, NY, USA (2009)
120. Zhou, B., Hui, S., Chang, K.: An intelligent recommender system using sequential web access patterns. In: *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, vol. 1, pp. 393–398 vol.1 (2004)
121. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: WWW '05: Proceedings of the 14th international conference on World Wide Web, pp. 22–32. ACM Press, New York, NY, USA (2005)

Part I
Basic Techniques

Chapter 2

Data Mining Methods for Recommender Systems

Xavier Amatriain, Alejandro Jaimes, Nuria Oliver, and Josep M. Pujol

Abstract In this chapter, we give an overview of the main Data Mining techniques used in the context of Recommender Systems. We first describe common preprocessing methods such as sampling or dimensionality reduction. Next, we review the most important classification techniques, including Bayesian Networks and Support Vector Machines. We describe the k -means clustering algorithm and discuss several alternatives. We also present association rules and related algorithms for an efficient training process. In addition to introducing these techniques, we survey their uses in Recommender Systems and present cases where they have been successfully applied.

2.1 Introduction

Recommender Systems (RS) typically apply techniques and methodologies from other neighboring areas – such as Human Computer Interaction (HCI) or Information Retrieval (IR). However, most of these systems bear in their core an algorithm that can be understood as a particular instance of a Data Mining (DM) technique.

The process of data mining typically consists of 3 steps, carried out in succession: *Data Preprocessing* [59], *Data Analysis*, and *Result Interpretation* (see Figure 2.1). We will analyze some of the most important methods for data preprocessing in Section 2.2. In particular, we will focus on sampling, dimensionality reduction, and the use of distance functions because of their significance and their role in RS. In Sections 2.3 through 2.5, we provide an overview introduction to the data mining methods that are most commonly used in RS: classification, clustering and associa-

Xavier Amatriain

Telefonica Research, Via Augusta, 122, Barcelona 08021, Spain e-mail: xar@tid.es

Alejandro Jaimes

Yahoo! Research, Av.Diagonal, 177, Barcelona 08018, Spain. Work on the chapter was performed while the author was at Telefonica Research. e-mail: ajaimes@yahoo-inc.com

Nuria Oliver

Telefonica Research, Via Augusta, 122, Barcelona 08021, Spain e-mail: nuriao@tid.es

Josep M. Pujol

Telefonica Research, Via Augusta, 122, Barcelona 08021, Spain e-mail: jmps@tid.es

tion rule discovery (see Figure 2.1 for a detailed view of the different topics covered in the chapter).

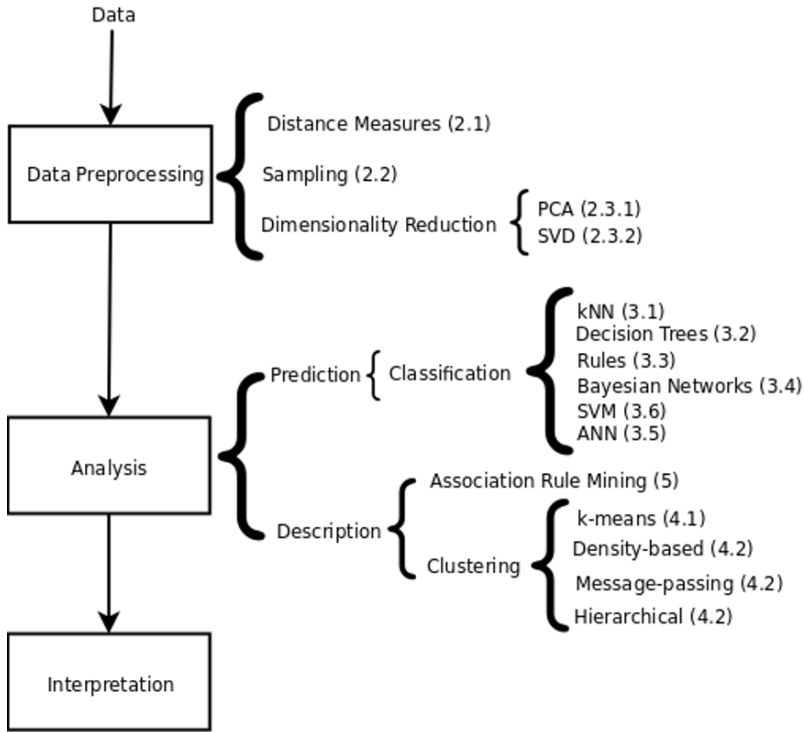


Fig. 2.1: Main steps and methods in a Data Mining problem, with their correspondence to chapter sections.

This chapter does not intend to give a thorough review of Data Mining methods, but rather to highlight the impact that DM algorithms have in the RS field, and to provide an overview of the key DM techniques that have been successfully used. We shall direct the interested reader to Data Mining textbooks (see [28, 73], for example) or the more focused references that are provided throughout the chapter.

2.2 Data Preprocessing

We define *data* as a collection of *objects* and their *attributes*, where an attribute is defined as a property or characteristic of an object. Other names for object include *record*, *item*, *point*, *sample*, *observation*, or *instance*. An attribute might be also be referred to as a *variable*, *field*, *characteristic*, or *feature*.

Real-life data typically needs to be *preprocessed* (e.g. cleansed, filtered, transformed) in order to be used by the machine learning techniques in the analysis step. In this section, we focus on three issues that are of particular importance when designing a RS. First, we review different similarity or distance measures. Next, we discuss the issue of sampling as a way to reduce the number of items in very large collections while preserving its main characteristics. Finally, we describe the most common techniques to reduce dimensionality.

2.2.1 Similarity Measures

One of the preferred approaches to collaborative filtering (CF) recommenders is to use the k NN classifier that will be described in Section 2.3.1. This classification method – as most classifiers and clustering techniques – is highly dependent on defining an appropriate similarity or distance measure.

The simplest and most common example of a distance measure is the Euclidean distance:

$$d(x, y) = \sqrt{\sum_{k=1}^n (x_k - y_k)^2} \quad (2.1)$$

where n is the number of dimensions (attributes) and x_k and y_k are the k^{th} attributes (components) of data objects x and y , respectively.

The Minkowski Distance is a generalization of Euclidean Distance:

$$d(x, y) = \left(\sum_{k=1}^n |x_k - y_k|^r \right)^{\frac{1}{r}} \quad (2.2)$$

where r is the degree of the distance. Depending on the value of r , the generic Minkowski distance is known with specific names: For $r = 1$, the *city block*, (*Manhattan*, *taxicab* or *L1 norm*) distance; For $r = 2$, the *Euclidean* distance; For $r \rightarrow \infty$, the *supremum* (*L_{max} norm* or *L_∞ norm*) distance, which corresponds to computing the maximum difference between any dimension of the data objects.

The Mahalanobis distance is defined as:

$$d(x, y) = \sqrt{(x - y)\sigma^{-1}(x - y)^T} \quad (2.3)$$

where σ is the covariance matrix of the data.

Another very common approach is to consider items as document vectors of an n -dimensional space and compute their similarity as the cosine of the angle that they form:

$$\cos(x, y) = \frac{(x \bullet y)}{\|x\| \|y\|} \quad (2.4)$$

where \bullet indicates vector dot product and $\|x\|$ is the norm of vector x . This similarity is known as the *cosine similarity* or the *L2 Norm*.

The similarity between items can also be given by their *correlation* which measures the linear relationship between objects. While there are several correlation coefficients that may be applied, the *Pearson correlation* is the most commonly used. Given the covariance of data points x and y Σ , and their standard deviation σ , we compute the Pearson correlation using:

$$Pearson(x, y) = \frac{\Sigma(x, y)}{\sigma_x \times \sigma_y} \quad (2.5)$$

RS have traditionally used either the cosine similarity (Eq. 2.4) or the Pearson correlation (Eq. 2.5) – or one of their many variations through, for instance, weighting schemes – both Chapters 5 and 4 detail the use of different distance functions for CF. However, most of the other distance measures previously reviewed are possible. Spertus *et al.* [69] did a large-scale study to evaluate six different similarity measures in the context of the Orkut social network. Although their results might be biased by the particular setting of their experiment, it is interesting to note that the best response to recommendations were to those generated using the cosine similarity. Lathia *et al.* [48] also carried out a study of several similarity measures where they concluded that, in the general case, the prediction accuracy of a RS was *not* affected by the choice of the similarity measure. As a matter of fact and in the context of their work, using a random similarity measure sometimes yielded better results than using any of the well-known approaches.

Finally, several similarity measures have been proposed in the case of items that only have binary attributes. First, the *M01*, *M10*, *M11*, and *M00* quantities are computed, where *M01* = the number of attributes where x was 0 and y was 1, *M10* = the number of attributes where x was 1 and y was 0, and so on. From those quantities we can compute: The *Simple Matching* coefficient $SMC = \frac{\text{number of matches}}{\text{number of attributes}} = \frac{M11+M00}{M01+M10+M00+M11}$; the *Jaccard* coefficient $JC = \frac{M11}{M01+M10+M11}$. The *Extended Jaccard* (Tanimoto) coefficient, a variation of JC for continuous or count attributes that is computed by $d = \frac{x \bullet y}{\|x\|^2 + \|y\|^2 - x \bullet y}$.

2.2.2 Sampling

Sampling is the main technique used in DM for selecting a subset of relevant data from a large data set. It is used both in the preprocessing and final data interpretation steps. Sampling may be used because processing the entire data set is computationally too expensive. It can also be used to create *training* and *testing* datasets. In this case, the training dataset is used to learn the parameters or configure the algorithms used in the analysis step, while the testing dataset is used to evaluate the model or configuration obtained in the training phase, making sure that it performs well (*i.e. generalizes*) with previously unseen data.

The key issue to sampling is finding a subset of the original data set that is *representative* – *i.e.* it has approximately the same property of interest – of the entire set. The simplest sampling technique is *random sampling*, where there is an equal probability of selecting any item. However, more sophisticated approaches are possible. For instance, in *stratified sampling* the data is split into several partitions based on a particular feature, followed by random sampling on each partition independently.

The most common approach to sampling consists of using sampling *without replacement*: When an item is selected, it is removed from the population. However, it is also possible to perform sampling *with replacement*, where items are not removed from the population once they have been selected, allowing for the same sample to be selected more than once.

It is common practice to use standard random sampling without replacement with an 80/20 proportion when separating the training and testing data sets. This means that we use random sampling without replacement to select 20% of the instances for the testing set and leave the remaining 80% for training. The 80/20 proportion should be taken as a rule of thumb as, in general, any value over 2/3 for the training set is appropriate.

Sampling can lead to an over-specialization to the particular division of the training and testing data sets. For this reason, the training process may be repeated several times. The training and test sets are created from the original data set, the model is trained using the training data and tested with the examples in the test set. Next, different training/test data sets are selected to start the training/testing process again that is repeated K times. Finally, the *average* performance of the K learned models is reported. This process is known as *cross-validation*. There are several cross-validation techniques. In *repeated random sampling*, a standard random sampling process is carried out K times. In *n-Fold cross validation*, the data set is divided into n folds. One of the folds is used for testing the model and the remaining $n - 1$ folds are used for training. The cross validation process is then repeated n times with each of the n subsamples used exactly once as validation data. Finally, the *leave-one-out (LOO)* approach can be seen as an extreme case of n -Fold cross validation where n is set to the number of items in the data set. Therefore, the algorithms are run as many times as data points using only one of them as a test each time. It should be noted, though, that as Isaksson *et al.* discuss in [44], cross-validation may be unreliable unless the data set is sufficiently large.

A common approach in RS is to sample the available feedback from the users – *e.g.* in the form of ratings – to separate it into training and testing. Cross-validation is also common. Although a standard random sampling is acceptable in the general case, in others we might need to bias our sampling for the test set in different ways. We might, for instance, decide to sample only from most recent ratings – since those are the ones we would be predicting in a real-world situation. We might also be interested in ensuring that the proportion of ratings per user is preserved in the test set and therefore impose that the random sampling is done on a per user basis. However, all these issues relate to the problem of evaluating RS, which is still a matter of research and discussion.

2.2.3 Reducing Dimensionality

It is common in RS to have not only a data set with features that define a high-dimensional space, but also very sparse information in that space – *i.e.* there are values for a limited number of features per object. The notions of density and distance between points, which are critical for clustering and outlier detection, become less meaningful in highly dimensional spaces. This is known as the *Curse of Dimensionality*. Dimensionality reduction techniques help overcome this problem by transforming the original high-dimensional space into a lower-dimensionality.

Sparsity and the *curse of dimensionality* are recurring problems in RS. Even in the simplest setting, we are likely to have a sparse matrix with thousands of rows and columns (*i.e.* users and items), most of which are zeros. Therefore, dimensionality reduction comes in naturally. Applying dimensionality reduction makes such a difference and its results are so directly applicable to the computation of the predicted value, that this is now considered to be an approach to RS design, rather than a preprocessing technique.

In the following, we summarize the two most relevant dimensionality reduction algorithms in the context of RS: *Principal Component Analysis (PCA)* and *Singular Value Decomposition (SVD)*. These techniques can be used in isolation or as a preprocessing step for any of the other techniques reviewed in this chapter.

2.2.3.1 Principal Component Analysis

Principal Component Analysis (PCA) [45] is a classical statistical method to find patterns in high dimensionality data sets. PCA allows to obtain an ordered list of components that account for the largest amount of the variance from the data in terms of least square errors: The amount of variance captured by the first component is larger than the amount of variance on the second component and so on. We can reduce the dimensionality of the data by neglecting those components with a small contribution to the variance.

Figure 2.2 shows the PCA analysis to a two-dimensional point cloud generated by a combination of Gaussians. After the data is centered, the principal components are obtained and denoted by u_1 and u_2 . Note that the length of the new coordinates is relative to the energy contained in their eigenvectors. Therefore, for the particular example depicted in Fig 2.2, the first component u_1 accounts for 83.5% of the energy, which means that removing the second component u_2 would imply losing only 16.5% of the information. The rule of thumb is to choose m' so that the cumulative energy is above a certain threshold, typically 90%. PCA allows us to retrieve the original data matrix by projecting the data onto the new coordinate system $X'_{n \times m'} = X_{n \times m} W'_{m \times m'}$. The new data matrix X' contains most of the information of the original X with a dimensionality reduction of $m - m'$.

PCA is a powerful technique, but it does have important limitations. PCA relies on the empirical data set to be a linear combination of a certain basis – although generalizations of PCA for non-linear data have been proposed. Another important

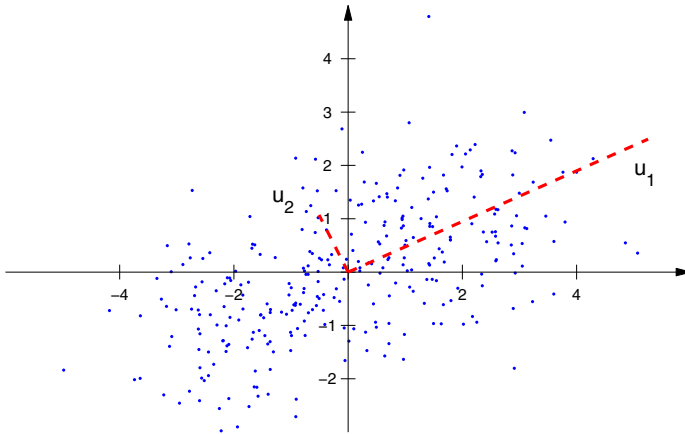


Fig. 2.2: PCA analysis of a two-dimensional point cloud from a combination of Gaussians. The principal components derived using PCS are u_1 and u_2 , whose length is relative to the energy contained in the components.

assumption of PCA is that the original data set has been drawn from a Gaussian distribution. When this assumption does not hold true, there is no warranty that the principal components are meaningful.

Although current trends seem to indicate that other matrix factorizations techniques such as SVD or Non-Negative Matrix Factorization are preferred, earlier works used PCA. Goldberg *et al.* proposed an approach to use PCA in the context of an online joke recommendation system [37]. Their system, known as *Eigentaste*¹, starts from a standard matrix of user ratings to items. They then select their *gauge* set by choosing the subset of items for which all users had a rating. This new matrix is then used to compute the global correlation matrix where a standard 2-dimensional PCA is applied.

2.2.3.2 Singular Value Decomposition

Singular Value Decomposition [38] is a powerful technique for dimensionality reduction. It is a particular realization of the *Matrix Factorization* approach and it is therefore also related to PCA. The key issue in an SVD decomposition is to find a lower dimensional feature space where the new features represent “concepts” and the strength of each concept in the context of the collection is computable. Because SVD allows to automatically derive semantic “concepts” in a low dimensional

¹ <http://eigentaste.berkeley.edu>

space, it can be used as the basis of *latent-semantic analysis*[24], a very popular technique for text classification in Information Retrieval .

The core of the SVD algorithm lies in the following theorem: It is always possible to decompose a given matrix A into $A = U\lambda V^T$. Given the $n \times m$ matrix data A (n items, m features), we can obtain an $n \times r$ matrix U (n items, r concepts), an $r \times r$ diagonal matrix λ (strength of each concept), and an $m \times r$ matrix V (m features, r concepts). Figure 2.3 illustrates this idea. The λ diagonal matrix contains the *singular values*, which will always be positive and sorted in decreasing order. The U matrix is interpreted as the “item-to-concept” similarity matrix, while the V matrix is the “term-to-concept” similarity matrix.

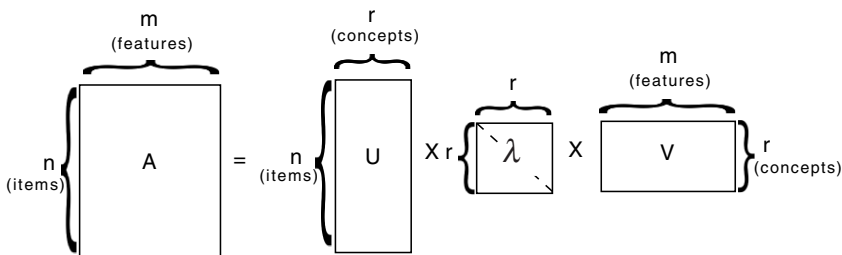


Fig. 2.3: Illustrating the basic Singular Value Decomposition Theorem: an item \times features matrix can be decomposed into three different ones: an item \times concepts, a concept strength, and a concept \times features.

In order to compute the SVD of a rectangular matrix A , we consider AA^T and $A^T A$. The columns of U are the eigenvectors of AA^T , and the columns of V are the eigenvectors of $A^T A$. The singular values on the diagonal of λ are the positive square roots of the nonzero eigenvalues of both AA^T and $A^T A$. Therefore, in order to compute the SVD of matrix A we first compute T as AA^T and D as $A^T A$ and then compute the eigenvectors and eigenvalues for T and D .

The r eigenvalues in λ are ordered in decreasing magnitude. Therefore, the original matrix A can be approximated by simply truncating the eigenvalues at a given k . The truncated SVD creates a rank- k approximation to A so that $A_k = U_k \lambda_k V_k^T$. A_k is the *closest* rank- k matrix to A . The term “closest” means that A_k minimizes the sum of the squares of the differences of the elements of A and A_k . The truncated SVD is a representation of the underlying latent structure in a reduced k -dimensional space, which generally means that the noise in the features is reduced.

The use of SVD as tool to improve collaborative filtering has been known for some time. Sarwar *et al.* [66] describe two different ways to use SVD in this context. First, SVD can be used to uncover latent relations between customers and products. In order to accomplish this goal, they first fill the zeros in the user-item matrix with the item average rating and then normalize by subtracting the user average. This matrix is then factored using SVD and the resulting decomposition can be used – after some trivial operations – directly to compute the predictions. The other

approach is to use the low-dimensional space resulting from the SVD to improve neighborhood formation for later use in a k NN approach.

As described by Sarwar *et al.* [65], one of the big advantages of SVD is that there are incremental algorithms to compute an approximated decomposition. This allows to accept new users or ratings without having to recompute the model that had been built from previously existing data. The same idea was later extended and formalized by Brand [14] into an online SVD model. The use of incremental SVD methods has recently become a commonly accepted approach after its success in the Netflix Prize ². The publication of Simon Funk's simplified incremental SVD method [35] marked an inflection point in the contest. Since its publication, several improvements to SVD have been proposed in this same context (see Paterek's ensembles of SVD methods [56] or Kurucz *et al.* evaluation of SVD parameters [47]).

Finally, it should be noted that different variants of Matrix Factorization (MF) methods such as the Non-negative Matrix Factorization (NNMF) have also been used [74]. These algorithms are, in essence, similar to SVD. The basic idea is to decompose the ratings matrix into two matrices, one of which contains features that describe the users and the other contains features describing the items. Matrix Factorization methods are better than SVD at handling the missing values by introducing a bias term to the model. However, this can also be handled in the SVD preprocessing step by replacing zeros with the item average. Note that both SVD and MF are prone to overfitting. However, there exist MF variants, such as the Regularized Kernel Matrix Factorization, that can avoid the issue efficiently. The main issue with MF – and SVD – methods is that it is unpractical to recompute the factorization every time the matrix is updated because of computational complexity. However, Rendle and Schmidt-Thieme [62] propose an online method that allows to update the factorized approximation without recomputing the entire model.

Chapter 5 details the use of SVD and MF in the context of the Netflix Prize and is therefore a good complement to this introduction.

2.2.4 Denoising

Data collected for data-mining purposes might be subject to different kinds of noise such as missing values or outliers. Denoising is a very important preprocessing step that aims at removing any unwanted effect in the data while maximizing its information.

In a general sense we define noise as any unwanted artifact introduced in the data collection phase that might affect the result of our data analysis and interpretation. In the context of RS, we distinguish between *natural* and *malicious* noise [55]. The former refers to noise that is unvoluntarily introduced by users when giving feedback on their preferences. The latter refers to noise that is deliberately introduced in a system in order to bias the results.

² <http://www.netflixprize.com>

It is clear that malicious noise can affect the output of a RS. But, also, we performed a study that concluded that the effects of natural noise on the performance of RS is far from being negligible [4]. In order to address this issue, we designed a denoising approach that is able to improve accuracy by asking some users to re-rate some items [5]. We concluded that accuracy improvements by investing in this pre-processing step could be larger than the ones obtained by complex algorithm optimizations.

2.3 Classification

A classifier is a mapping between a feature space and a label space, where the features represent characteristics of the elements to classify and the labels represent the classes. A restaurant RS, for example, can be implemented by a classifier that classifies restaurants into one of two categories (good, bad) based on a number of features that describe it.

There are many types of classifiers, but in general we will talk about either *supervised* or *unsupervised* classification. In supervised classification, a set of labels or categories is known in advance and we have a set of labeled examples which constitute a training set. In unsupervised classification, the labels or categories are unknown in advance and the task is to suitably (according to some criteria) organize the elements at hand. In this section we describe several algorithms to learn supervised classifiers and will be covering unsupervised classification (*i.e.* clustering) in Sec. 2.4.

2.3.1 Nearest Neighbors

Instance-based classifiers work by storing training records and using them to predict the class label of unseen cases. A trivial example is the so-called *rote-learner*. This classifier memorizes the entire training set and classifies only if the attributes of the new record match one of the training examples exactly. A more elaborate, and far more popular, instance-based classifier is the *Nearest neighbor classifier* (*k*NN) [22]. Given a point to be classified, the *k*NN classifier finds the *k* closest points (*nearest neighbors*) from the training records. It then assigns the class label according to the class labels of its *nearest-neighbors*. The underlying idea is that if a record falls in a particular neighborhood where a class label is predominant it is because the record is likely to belong to that very same class.

Given a query point q for which we want to know its class l , and a training set $X = \{\{x_1, l_1\} \dots \{x_n\}\}$, where x_j is the j -th element and l_j is its class label, the k -nearest neighbors will find a subset $Y = \{\{y_1, l_1\} \dots \{y_k\}\}$ such that $Y \in X$ and $\sum_1^k d(q, y_k)$ is minimal. Y contains the k points in X which are closest to the query point q . Then, the class label of q is $l = f(\{l_1 \dots l_k\})$.

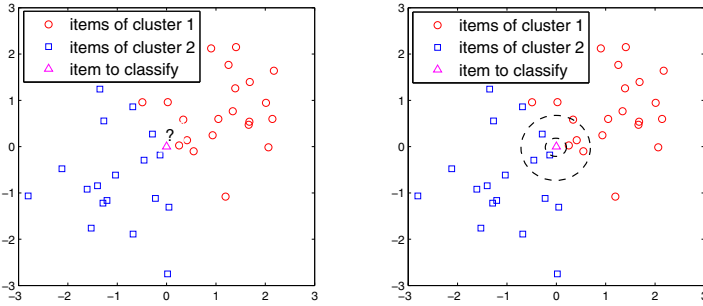


Fig. 2.4: Example of k -Nearest Neighbors. The left subfigure shows the training points with two class labels (circles and squares) and the query point (as a triangle). The right sub-figure illustrates closest neighborhood for $k = 1$ and $k = 7$. The query point would be classified as square for $k = 1$, and as a circle for $k = 5$ according to the simple majority vote rule. Note that the query points was just on the boundary between the two clusters.

Perhaps the most challenging issue in k NN is how to choose the value of k . If k is too small, the classifier will be sensitive to noise points. But if k is too large, the neighborhood might include too many points from other classes. The right plot in Fig. 2.4 shows how different k yields different class label for the query point, if $k = 1$ the class label would be *circle* whereas $k = 7$ classifies it as *square*. Note that the query point from the example is on the boundary of two clusters, and therefore, it is difficult to classify.

k NN classifiers are amongst the simplest of all machine learning algorithms. Since k NN does not build models explicitly it is considered a *lazy learner*. Unlike eager learners such as decision trees or rule-based systems (see 2.3.2 and 2.3.3, respectively), k NN classifiers leave many decisions to the classification step. Therefore, classifying unknown records is relatively expensive.

Nearest Neighbor is one of the most common approaches to CF – and therefore to designing a RS. As a matter of fact, any overview on RS – such as the one by Adomavicius and Tuzhilin [1] – will include an introduction to the use of nearest neighbors in this context. One of the advantages of this classifier is that it is conceptually very much related to the idea of CF: Finding like-minded users (or similar items) is essentially equivalent to finding neighbors for a given user or an item. The other advantage is that, being the k NN classifier a lazy learner, it does not require to learn and maintain a given model. Therefore, in principle, the system can adapt to rapid changes in the user ratings matrix. Unfortunately, this comes at the cost of recomputing the neighborhoods and therefore the similarity matrix. This is why we proposed a neighborhood model that uses a reduced set of experts as the source for selecting neighbors [3].

The k NN approach, although simple and intuitive, has shown good accuracy results and is very amenable to improvements. As a matter of fact, its supremacy as the *de facto* standard for CF recommendation has only been challenged recently by approaches based on dimensionality reduction such as the ones reviewed in Section 2.2.3. That said, the traditional k NN approach to CF has experienced improvements in several directions. For instance, in the context of the Netflix Prize, Bell and Koren propose a method to remove *global effects* such as the fact that some items may attract users that consistently rate lower. They also propose an optimization method for computing interpolating weights once the neighborhood is created.

See Chapters 5 and 4 for more details on enhanced CF techniques based on the use of neighborhoods.

2.3.2 Decision Trees

Decision trees [61, 63] are classifiers on a target attribute (or class) in the form of a tree structure. The observations (or items) to classify are composed of attributes and their target value. The nodes of the tree can be: a) *decision nodes*, in these nodes a single attribute-value is tested to determine to which branch of the subtree applies. Or b) *leaf nodes* which indicate the value of the target attribute.

There are many algorithms for decision tree induction: Hunts Algorithm, CART, ID3, C4.5, SLIQ, SPRINT to mention the most common. The recursive Hunt algorithm, which is one of the earliest and easiest to understand, relies on the *test condition* applied to a given attribute that discriminates the observations by their target values. Once the partition induced by the test condition has been found, the algorithm is recursively repeated until a partition is empty or all the observations have the same target value.

Splits can be decided by maximizing the information gain, defined as follows,

$$\Delta_i = I(\text{parent}) - \sum_{j=1}^{k_i} \frac{N(v_j)I(v_j)}{N} \quad (2.6)$$

where k_i are values of the attribute i , N is the number of observations, v_j is the j -th partition of the observations according to the values of attribute i . Finally, I is a function that measures node *impurity*. There are different measures of impurity: Gini Index, Entropy and misclassification error are the most common in the literature.

Decision tree induction stops once all observations belong to the same class (or the same range in the case of continuous attributes). This implies that the impurity of the leaf nodes is zero. For practical reasons, however, most decision trees implementations use pruning by which a node is no further split if its impurity measure or the number of observations in the node are below a certain threshold.

The main advantages of building a classifier using a decision tree is that it is inexpensive to construct and it is extremely fast at classifying unknown instances. Another appreciated aspect of decision tree is that they can be used to produce a set

of rules that are easy to interpret (see section 2.3.3) while maintaining an accuracy comparable to other basic classification techniques.

Decision trees may be used in a model-based approach for a RS. One possibility is to use content features to build a decision tree that models all the variables involved in the user preferences. Bouza *et al.* [12] use this idea to construct a Decision Tree using semantic information available for the items. The tree is built after the user has rated only two items. The features for each of the items are used to build a model that explains the user ratings. They use the information gain of every feature as the splitting criteria. It should be noted that although this approach is interesting from a theoretical perspective, the precision they report on their system is worse than that of recommending the average rating.

As it could be expected, it is very difficult and unpractical to build a decision tree that tries to explain all the variables involved in the decision making process. Decision trees, however, may also be used in order to model a particular part of the system. Cho *et al.* [18], for instance, present a RS for online purchases that combines the use of Association Rules (see Section 2.5) and Decision Trees. The Decision Tree is used as a filter to select which users should be targeted with recommendations. In order to build the model they create a candidate user set by selecting those users that have chosen products from a given category during a given time frame. In their case, the dependent variable for building the decision tree is chosen as whether the customer is likely to buy new products in that same category. Nikovski and Kulev [54] follow a similar approach combining Decision Trees and Association Rules. In their approach, frequent itemsets are detected in the purchase dataset and then they apply standard tree-learning algorithms for simplifying the recommendations rules.

Another option to use Decision Trees in a RS is to use them as a tool for item ranking. The use of Decision Trees for ranking has been studied in several settings and their use in a RS for this purpose is fairly straightforward [7, 17].

2.3.3 Ruled-based Classifiers

Rule-based classifiers classify data by using a collection of “**if ... then ...**” rules. The rule *antecedent* or condition is an expression made of attribute conjunctions. The rule *consequent* is a positive or negative classification.

We say that a rule r covers a given instance x if the attributes of the instance satisfy the rule condition. We define the *coverage* of a rule as the fraction of records that satisfy its antecedent. On the other hand, we define its *accuracy* as the fraction of records that satisfy both the antecedent and the consequent. We say that a classifier contains *mutually exclusive rules* if the rules are independent of each other – *i.e.* every record is covered by at most one rule. Finally we say that the classifier has *exhaustive rules* if they account for every possible combination of attribute values –*i.e.* each record is covered by at least one rule.

In order to build a rule-based classifier we can follow a direct method to extract rules directly from data. Examples of such methods are RIPPER, or CN2. On the other hand, it is common to follow an indirect method and extract rules from other classification models such as decision trees or neural networks.

The advantages of rule-based classifiers are that they are extremely expressive since they are symbolic and operate with the attributes of the data without any transformation. Rule-based classifiers, and by extension decision trees, are easy to interpret, easy to generate and they can classify new instances efficiently.

In a similar way to Decision Tress, however, it is very difficult to build a complete recommender model based on rules. As a matter of fact, this method is not very popular in the context of RS because deriving a rule-based system means that we either have some explicit prior knowledge of the decision making process or that we derive the rules from another model such a decision tree. However a rule-based system can be used to improve the performance of a RS by injecting partial domain knowledge or business rules. Anderson *et al.* [6], for instance, implemented a CF music RS that improves its performance by applying a rule-based system to the results of the CF process. If a user rates an album by a given artist high, for instance, predicted ratings for all other albums by this artist will be increased.

Gutta *et al.* [29] implemented a rule-based RS for TV content. In order to do, so they first derived a C4.5 Decision Tree that is then decomposed into rules for classifying the programs. Basu *et al.* [9] followed an inductive approach using the *Ripper* [20] system to learn rules from data. They report slightly better results when using hybrid content and collaborative data to learn rules than when following a pure CF approach.

2.3.4 Bayesian Classifiers

A Bayesian classifier [34] is a probabilistic framework for solving classification problems. It is based on the definition of conditional probability and the Bayes theorem. The Bayesian school of statistics uses probability to represent uncertainty about the relationships learned from the data. In addition, the concept of *priors* is very important as they represent our expectations or prior knowledge about what the true relationship might be. In particular, the probability of a model given the data (*posterior*) is proportional to the product of the *likelihood* times the *prior probability* (or prior). The likelihood component includes the effect of the data while the prior specifies the belief in the model before the data was observed.

Bayesian classifiers consider each attribute and class label as (continuous or discrete) random variables. Given a record with N attributes (A_1, A_2, \dots, A_N) , the goal is to predict class C_k by finding the value of C_k that maximizes the posterior probability of the class given the data $P(C_k|A_1, A_2, \dots, A_N)$. Applying Bayes' theorem, $P(C_k|A_1, A_2, \dots, A_N) \propto P(A_1, A_2, \dots, A_N|C_k)P(C_k)$

A particular but very common Bayesian classifier is the *Naive Bayes Classifier*. In order to estimate the conditional probability, $P(A_1, A_2, \dots, A_N|C_k)$, a Naive Bayes

Classifier assumes the probabilistic *independence* of the attributes – *i.e.* the presence or absence of a particular attribute is unrelated to the presence or absence of any other. This assumption leads to $P(A_1, A_2, \dots, A_N | C_k) = P(A_1 | C_k) P(A_2 | C_k) \dots P(A_N | C_k)$.

The main benefits of Naive Bayes classifiers are that they are robust to isolated noise points and irrelevant attributes, and they handle missing values by ignoring the instance during probability estimate calculations. However, the independence assumption may not hold for some attributes as they might be correlated. In this case, the usual approach is to use the so-called *Bayesian Belief Networks (BBN)* (or Bayesian Networks, for short). BBN's use an acyclic graph to encode the dependence between attributes and a probability table that associates each node to its immediate parents. BBN's provide a way to capture prior knowledge in a domain using a graphical model. In a similar way to Naive Bayes classifiers, BBN's handle incomplete data well and they are quite robust to model overfitting.

Bayesian classifiers are particularly popular for model-based RS. They are often used to derive a model for content-based RS. However, they have also been used in a CF setting. Ghani and Fano [36], for instance, use a Naive Bayes classifier to implement a content-based RS. The use of this model allows for recommending products from unrelated categories in the context of a department store.

Miyahara and Pazzani [52] implement a RS based on a Naive Bayes classifier. In order to do so, they define two classes: *like* and *don't like*. In this context they propose two ways of using the Naive Bayesian Classifier: The *Transformed Data Model* assumes that all features are completely independent, and feature selection is implemented as a preprocessing step. On the other hand, the *Sparse Data Model* assumes that only known features are informative for classification. Furthermore, it only makes use of data which both users rated in common when estimating probabilities. Experiments show both models to perform better than a correlation-based CF.

Pronk *et al.* [58] use a Bayesian Naive Classifier as the base for incorporating user control and improving performance, especially in cold-start situations. In order to do so they propose to maintain two profiles for each user: one learned from the rating history, and the other explicitly created by the user. The blending of both classifiers can be controlled in such a way that the user-defined profile is favored at early stages, when there is not too much rating history, and the learned classifier takes over at later stages.

In the previous section we mentioned that Gutta *et al.* [29] implemented a rule-based approach in a TV content RS. Another of the approaches they tested was a Bayesian classifier. They define a two-class classifier, where the classes are *watched/not watched*. The user profile is then a collection of attributes together with the number of times they occur in positive and negative examples. This is used to compute prior probabilities that a show belongs to a particular class and the conditional probability that a given feature will be present if a show is either positive or negative. It must be noted that features are, in this case, related to both content – *i.e.* genre – and contexts – *i.e.* time of the day. The posteriori probabilities for a new show are then computed from these.

Breese *et al.* [15] implement a Bayesian Network where each node corresponds to each item. The states correspond to each possible vote value. In the network, each item will have a set of parent items that are its best predictors. The conditional probability tables are represented by decision trees. The authors report better results for this model than for several nearest-neighbors implementations over several datasets.

Hierarchical Bayesian Networks have also been used in several settings as a way to add domain-knowledge for information filtering [78]. One of the issues with hierarchical Bayesian networks, however, is that it is very expensive to learn and update the model when there are many users in it. Zhang and Koren [79] propose a variation over the standard Expectation-Maximization (EM) model in order to speed up this process in the scenario of a content-based RS.

2.3.5 Artificial Neural Networks

An Artificial Neural Network (ANN) [81] is an assembly of inter-connected nodes and weighted links that is inspired in the architecture of the biological brain. Nodes in an ANN are called *neurons* as an analogy with biological neurons. These simple functional units are composed into networks that have the ability to learn a classification problem after they are trained with sufficient data.

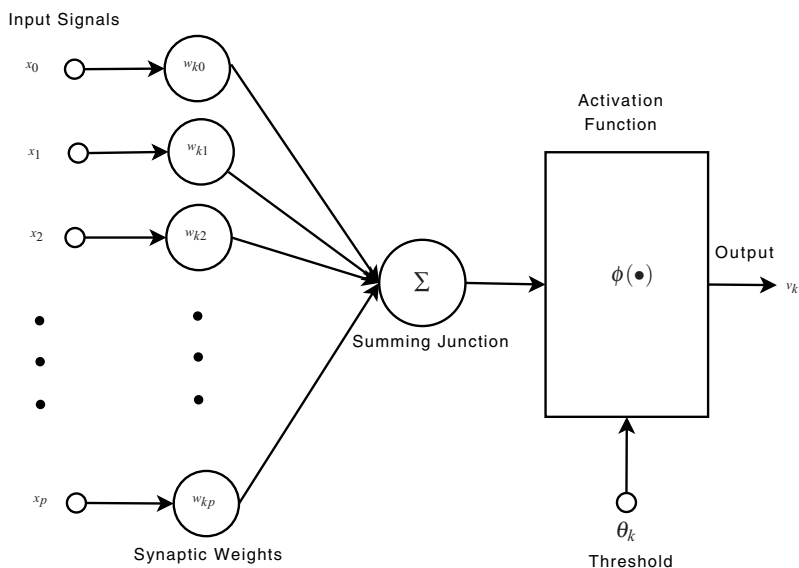


Fig. 2.5: Perceptron model

The simplest case of an ANN is the *perceptron* model, illustrated in figure 2.5. If we particularize the *activation function* ϕ to be the simple Threshold Function, the

output is obtained by summing up each of its input value according to the weights of its links and comparing its output against some threshold θ_k . The output function can be expressed using Eq. 2.7. The perceptron model is a linear classifier that has a simple and efficient learning algorithm. But, besides the simple Threshold Function used in the Perceptron model, there are several other common choices for the activation function such as sigmoid, tanh, or step functions.

$$y_k = \begin{cases} 1, & \text{if } \sum x_i w_{ki} \geq \theta_k \\ 0, & \text{if } \sum x_i w_{ki} < \theta_k \end{cases} \quad (2.7)$$

An ANN can have any number of layers. Layers in an ANN are classified into three types: input, hidden, and output. Units in the input layer respond to data that is fed into the network. Hidden units receive the weighted output from the input units. And the output units respond to the weighted output from the hidden units and generate the final output of the network. Using neurons as atomic functional units, there are many possible architectures to put them together in a network. But, the most common approach is to use the *feed-forward ANN*. In this case, signals are strictly propagated in one way: from input to output.

The main advantages of ANN are that – depending on the activation function – they can perform non-linear classification tasks, and that, due to their parallel nature, they can be efficient and even operate if part of the network fails. The main disadvantage is that it is hard to come up with the ideal network topology for a given problem and once the topology is decided this will act as a lower bound for the classification error. ANN's belong to the class of *sub-symbolic* classifiers, which means that they provide no semantics for inferring knowledge – *i.e.* they promote a kind of *black-box* approach.

ANN's can be used in a similar way as Bayesian Networks to construct model-based RS's. However, there is no conclusive study to whether ANN introduce any performance gain. As a matter of fact, Pazzani and Billsus [57] did a comprehensive experimental study on the use of several machine learning algorithms for web site recommendation. Their main goal was to compare the simple naive Bayesian Classifier with computationally more expensive alternatives such as Decision Trees and Neural Networks. Their experimental results show that Decision Trees perform significantly worse. On the other hand ANN and the Bayesian classifier performed similarly. They conclude that there does not seem to be a need for nonlinear classifiers such as the ANN. Berka *et al.* [31] used ANN to build an URL RS for web navigation. They implemented a content-independent system based exclusively on *trails* – *i.e.* associating pairs of domain names with the number of people who traversed them. In order to do so they used feed-forward Multilayer Perceptrons trained with the Backpropagation algorithm.

ANN can be used to combine (or hybridize) the input from several recommendation modules or data sources. Hsu *et al.* [30], for instance, build a TV recommender by importing data from four different sources: user profiles and stereotypes; viewing communities; program metadata; and viewing context. They use the back-propagation algorithm to train a three-layered neural network. Christakou and

Stafylopatis [19] also built a hybrid content-based CF RS. The content-based recommender is implemented using three neural networks per user, each of them corresponding to one of the following features: “kinds”, “stars”, and “synopsis”. They trained the ANN using the Resilient Backpropagation method.

2.3.6 Support Vector Machines

The goal of a Support Vector Machine (SVM) classifier [23] is to find a linear hyperplane (decision boundary) that separates the data in such a way that the margin is maximized. For instance, if we look at a two class separation problem in two dimensions like the one illustrated in figure 2.6, we can easily observe that there are many possible boundary lines to separate the two classes. Each boundary has an associated margin. The rationale behind SVM’s is that if we choose the one that maximizes the margin we are less likely to missclassify unknown items in the future.

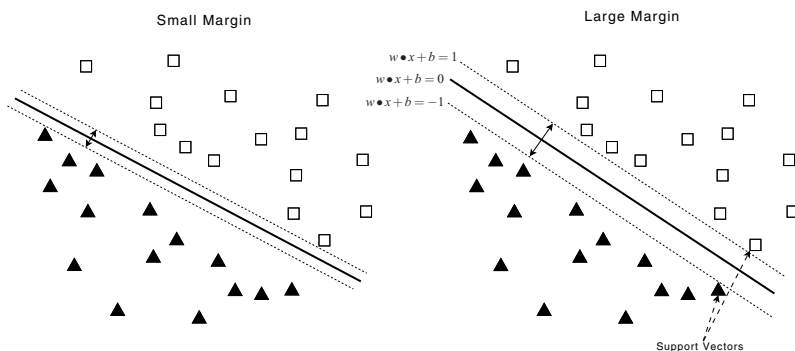


Fig. 2.6: Different boundary decisions are possible to separate two classes in two dimensions. Each boundary has an associated margin.

A linear separation between two classes is accomplished through the function $w \bullet x + b = 0$. We define a function that can classify items of being of class +1 or -1 as long as they are separated by some minimum distance from the class separation function. The function is given by Eq. 2.8

$$f(x) = \begin{cases} 1, & \text{if } w \bullet x + b \geq 1 \\ -1, & \text{if } w \bullet x + b \leq -1 \end{cases} \quad (2.8)$$

$$\text{Margin} = \frac{2}{\|w\|^2} \quad (2.9)$$

Following the main rationale for SVM's, we would like to maximize the margin between the two classes, given by equation 2.9. This is in fact equivalent to minimizing the inverse value $L(w) = \frac{\|w\|^2}{2}$ but subjected to the constraints given by $f(x)$. This is a constrained optimization problem and there are numerical approaches to solve it (e.g., quadratic programming).

If the items are not linearly separable we can decide to turn the svm into a *soft margin* classifier by introducing a *slack variable*. In this case the formula to minimize is given by equation 2.10 subject to the new definition of $f(x)$ in equation 2.11. On the other hand, if the decision boundary is not linear we need to transform data into a higher dimensional space. This is accomplished thanks to a mathematical transformation known as the *kernel trick*. The basic idea is to replace the dot products in equation 2.8 by a *kernel* function. There are many different possible choices for the kernel function such as Polynomial or Sigmoid. But the most common kernel functions are the family of Radial Basis Function (RBF).

$$L(w) = \frac{\|w\|^2}{2} + C \sum_{i=1}^N \varepsilon \quad (2.10)$$

$$f(x) = \begin{cases} 1, & \text{if } w \bullet x + b \geq 1 - \varepsilon \\ -1, & \text{if } w \bullet x + b \leq -1 + \varepsilon \end{cases} \quad (2.11)$$

Support Vector Machines have recently gained popularity for their performance and efficiency in many settings. SVM's have also shown promising recent results in RS. Kang and Yoo [46], for instance, report on an experimental study that aims at selecting the best preprocessing technique for predicting missing values for an SVM-based RS. In particular, they use SVD and Support Vector Regression. The Support Vector Machine RS is built by first binarizing the 80 levels of available user preference data. They experiment with several settings and report best results for a threshold of 32 – i.e. a value of 32 and less is classified as *prefer* and a higher value as *do not prefer*. The user id is used as the class label and the positive and negative values are expressed as preference values 1 and 2.

Xu and Araki [76] used SVM to build a TV program RS. They used information from the Electronic Program Guide (EPG) as features. But in order to reduce features they removed words with lowest frequencies. Furthermore, and in order to evaluate different approaches, they used both the Boolean and the *Term frequency - inverse document frequency* (TFIDF) weighting schemes for features. In the former, 0 and 1 are used to represent absence or presence of a term on the content. In the latter, this is turned into the TFIDF numerical value.

Xia *et al.*[75] present different approaches to using SVM's for RS in a CF setting. They explore the use of Smoothing Support Vector Machines (SSVM). They also introduce a SSVM-based heuristic (SSVMBH) to iteratively estimate missing elements in the user-item matrix. They compute predictions by creating a classifier for each user. Their experimental results report best results for the SSVMBH as compared to both SSVM's and traditional user-based and item-based CF. Finally,

Oku *et al.* [27] propose the use of Context-Aware Vector Machines (C-SVM) for context-aware RS. They compare the use of standard SVM, C-SVM and an extension that uses CF as well as C-SVM. Their results show the effectiveness of the context-aware methods for restaurant recommendations.

2.3.7 Ensembles of Classifiers

The basic idea behind the use of *ensembles* of classifiers is to construct a set of classifiers from the training data and predict class labels by aggregating their predictions. Ensembles of classifiers work whenever we can assume that the classifiers are independent. In this case we can ensure that the ensemble will produce results that are in the worst case as bad as the worst classifier in the ensemble. Therefore, combining independent classifiers of a similar classification error will only improve results.

In order to generate ensembles, several approaches are possible. The two most common techniques are *Bagging* and *Boosting*. In Bagging, we perform sampling with replacement, building the classifier on each bootstrap sample. Each sample has probability $(1 - \frac{1}{N})^N$ of being selected – note that if N is large enough, this converges to $1 - \frac{1}{e} \approx 0.623$. In Boosting we use an iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records. Initially, all records are assigned equal weights. But, unlike bagging, weights may change at the end of each boosting round: Records that are wrongly classified will have their weights increased while records that are classified correctly will have their weights decreased. An example of boosting is the AdaBoost algorithm.

The use of ensembles of classifiers is common practice in the RS field. As a matter of fact, any *hybridation* technique [16] can be considered an ensemble as it combines in one way or another several classifiers. An explicit example of this is Tiemann and Pauws' music recommender, in which they use ensemble learning methods to combine a social and a content-base RS [70].

Experimental results show that ensembles can produce better results than any classifier in isolation. Bell *et al.* [11], for instance, used a combination of 107 different methods in their progress prize winning solution to the Netflix challenge. They state that their findings show that it pays off more to find substantially different approaches rather than focusing on refining a particular technique. In order to blend the results from the ensembles they use a linear regression approach and to derive weights for each classifier, they partition the test dataset into 15 different bins and derive unique coefficients for each of the bins. Different uses of ensembles in the context of the Netflix prize can be tracked in other approaches such as in Schlar *et al.*'s [67] or Toescher *et al.*'s [71].

The boosting approach has also been used in RS. Freund *et al.*, for instance, present an algorithm called RankBoost to combine preferences [32]. They apply the algorithm to produce movie recommendations in a CF setting.

2.3.8 Evaluating Classifiers

The most commonly accepted evaluation measure for RS is the Mean Average Error or Root Mean Squared Error of the predicted interest (or rating) and the measured one. These measures compute accuracy without any assumption on the purpose of the RS. However, as McNee et al. point out [51], there is much more than accuracy to deciding whether an item should be recommended. Herlocker et al. [42] provide a comprehensive review of algorithmic evaluation approaches to RS. They suggest that some measures could potentially be more appropriate for some tasks. However, they are not able to validate the measures when evaluating the different approaches empirically on a class of recommendation algorithms and a single set of data.

A step forward is to consider that the purpose of a “real” RS is to produce a top-N list of recommendations and evaluate RS depending on how well they can classify items as being *recommendable*. If we look at our recommendation as a classification problem, we can make use of well-known measures for classifier evaluation such as precision and recall. In the following paragraphs, we will review some of these measures and their application to RS evaluation. Note however that learning algorithms and classifiers can be evaluated by multiple criteria. This includes how accurately they perform the classification, their computational complexity during training, complexity during classification, their sensitivity to noisy data, their scalability, and so on. But in this section we will focus only on classification performance.

In order to evaluate a model we usually take into account the following measures: **True Positives (TP)**: number of instances classified as belonging to class A that truly belong to class A; **True Negatives (TN)**: number of instances classified as not belonging to class A and that in fact do not belong to class A; **False Positives (FP)**: number of instances classified as class A but that do not belong to class A; **False Negatives (FN)**: instances not classified as belonging to class v but that in fact do belong to class A.

The most commonly used measure for model performance is its *Accuracy* defined as the ratio between the instances that have been correctly classified (as belonging or not to the given class) and the total number of instances: $Accuracy = (TP + TN) / (TP + TN + FP + FN)$. However, accuracy might be misleading in many cases. Imagine a 2-class problem in which there are 99,900 samples of class A and 100 of class B. If a classifier simply predicts everything to be of class A, the computed accuracy would be of 99.9% but the model performance is questionable because it will never detect any class B examples. One way to improve this evaluation is to define the cost matrix where we declare the cost of misclassifying class B examples as being of class A. In real world applications different types of errors may indeed have very different costs. For example, if the 100 samples above correspond to defective airplane parts in an assembly line, incorrectly rejecting a non-defective part (one of the 99,900 samples) has a negligible cost compared to the cost of mistakenly classifying a defective part as a good part.

Other common measures of model performance, particularly in Information Retrieval, are Precision and Recall. Precision, defined as $P = TP / (TP + FP)$, is a

measure of how many errors we make in classifying samples as being of class A. On the other hand, recall, $R = TP / (TP + FN)$, measures how good we are in not leaving out samples that should have been classified as belonging to the class. Note that these two measures are misleading when used in isolation in most cases. We could build a classifier of perfect precision by not classifying any sample as being of class A (therefore obtaining 0 TP but also 0 FP). Conversely, we could build a classifier of perfect recall by classifying all samples as belonging to class A. As a matter of fact, there is a measure, called the F_1 -measure that combines both Precision and Recall into a single measure as: $F_1 = \frac{2RP}{R+P} = \frac{2TP}{2TP+FN+FP}$

Sometimes we would like to compare several competing models rather than estimate their performance independently. In order to do so we use a technique developed in the 1950s for analysis of noisy signals: the Receiver Operating Characteristic (ROC) Curve. A ROC curve characterizes the relation between positive hits and false alarms. The performance of each classifier is represented as a point on the curve (see Fig. 2.7).

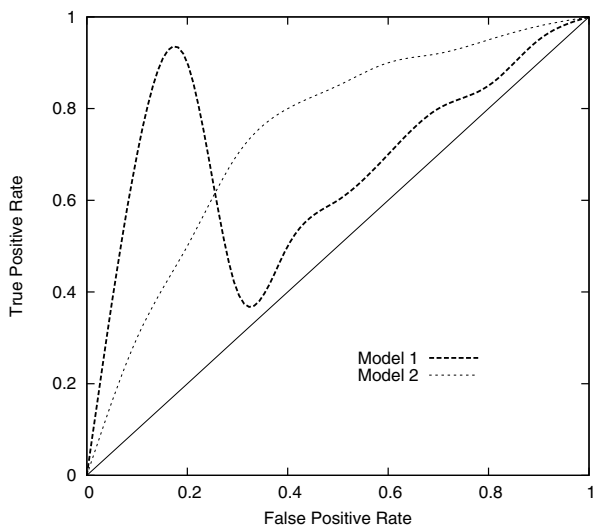


Fig. 2.7: Example of ROC Curve. Model 1 performs better for low False Positive Rates while Model 2 is fairly consistent throughout and outperforms Model 1 for False Positive Rates higher than 0.25

Ziegler *et al.* show [80] that evaluating recommender algorithms through top-N lists measures still does not map directly to the user's utility function. However, it does address some of the limitations of the more commonly accepted accuracy measures, such as MAE. Basu *et al.* [10], for instance, use this approach by analyzing which of the items predicted in the top quartile of the rating scale were actually evaluated in the top quartile by the user. McLaughlin and Herlocker [50] propose

a *modified precision* measure in which non-rated items are counted as *not recommendable*. This precision measure in fact represents a lower-bound of the “real” precision. Although the F-measure can be directly derived from the precision-recall values, it is not common to find it in RS evaluations. Huang *et al.* [43] and Bozzon *et al.* [13], and Miyahara and Pazzani [52] are some of the few examples of the use of this measure.

ROC curves have also been used in evaluating RS. Zhang *et al.* [64] use the value of the area under the ROC curve as their evaluation measure when comparing the performance of different algorithms under attack. Banerjee and Ramanathan [8] also use the ROC curves to compare the performance of different models.

It must be noted, though, that the choice of a good evaluation measure, even in the case of a top-N RS, is still a matter of discussion. Many authors have proposed measures that are only indirectly related to these traditional evaluation schemes. Deshpande and Karypis [25], for instance, propose the use of the *hit rate* and the *average reciprocal hit-rank*. On the other hand, Breese *et al.* [15] define a measure of the utility of the recommendation in a ranked list as a function of the neutral vote.

Note that Chapter 8 details on the use of some of these evaluation measures in the context of RS and is therefore a good place to continue if you are interested on this topic.

2.4 Cluster Analysis

The main problem for scaling a CF classifier is the amount of operations involved in computing distances – for finding the best k -nearest neighbors, for instance. A possible solution is, as we saw in section 2.2.3, to reduce dimensionality. But, even if we reduce dimensionality of features, we might still have many objects to compute the distance to. This is where clustering algorithms can come into play. The same is true for content-based RS, where distances among objects are needed to retrieve similar ones. Clustering is sure to improve efficiency because the number of operations is reduced. However, and unlike dimensionality reduction methods, it is unlikely that it can help improve accuracy. Therefore, clustering must be applied with care when designing a RS, measuring the compromise between improved efficiency and a possible decrease in accuracy.

Clustering [41], also referred to as unsupervised learning, consists of assigning items to groups so that the items in the same groups are more similar than items in different groups: the goal is to discover natural (or meaningful) groups that exist in the data. Similarity is determined using a distance measure, such as the ones reviewed in 2.2.1. The goal of a clustering algorithm is to minimize intra-cluster distances while maximizing inter-cluster distances.

There are two main categories of clustering algorithms: hierarchical and partitional. Partitional clustering algorithms divide data items into non-overlapping clusters such that each data item is in exactly one cluster. Hierarchical clustering algo-

gorithms successively cluster items within found clusters, producing a set of nested cluster organized as a hierarchical tree.

Many clustering algorithms try to minimize a function that measures the quality of the clustering. Such a quality function is often referred to as the objective function, so clustering can be viewed as an optimization problem: the ideal clustering algorithm would consider all possible partitions of the data and output the partitioning that minimizes the quality function. But the corresponding optimization problem is NP hard, so many algorithms resort to heuristics (e.g., in the k -means algorithm using only local optimization procedures potentially ending in local minima). The main point is that clustering is a difficult problem for which finding optimal solutions is often not possible. For that same reason, selection of the particular clustering algorithm and its parameters (e.g., similarity measure) depend on many factors, including the characteristics of the data. In the following paragraphs we describe the k -means clustering algorithm and some of its alternatives.

2.4.1 k -Means

k -Means clustering is a partitioning method. The function partitions the data set of N items into k disjoint subsets S_j that contain N_j items so that they are as close to each other as possible according a given distance measure. Each cluster in the partition is defined by its N_j members and by its centroid λ_j . The centroid for each cluster is the point to which the sum of distances from all items in that cluster is minimized. Thus, we can define the k -means algorithm as an iterative process to minimize $E = \sum_1^k \sum_{n \in S_j} d(x_n, \lambda_j)$, where x_n is a vector representing the n -th item, λ_j is the centroid of the item in S_j and d is the distance measure. The k -means algorithm moves items between clusters until E cannot be decreased further.

The algorithm works by randomly selecting k centroids. Then all items are assigned to the cluster whose centroid is the closest to them. The new cluster centroid needs to be updated to account for the items who have been added or removed from the cluster and the membership of the items to the cluster updated. This operation continues until there are no further items that change their cluster membership. Most of the convergence to the final partition takes place during the first iterations of the algorithm, and therefore, the stopping condition is often changed to “until relatively few points change clusters” in order to improve efficiency.

The basic k -means is an extremely simple and efficient algorithm. However, it does have several shortcomings: (1) it assumes prior knowledge of the data in order to choose the appropriate k ; (2) the final clusters are very sensitive to the selection of the initial centroids; and (3), it can produce empty cluster. k -means also has several limitations with regard to the data: it has problems when clusters are of differing sizes, densities, and non-globular shapes; and it also has problems when the data contains outliers.

Xue *et al.* [77] present a typical use of clustering in the context of a RS by employing the k -means algorithm as a pre-processing step to help in neighborhood for-

mation. They do not restrict the neighborhood to the cluster the user belongs to but rather use the distance from the user to different cluster centroids as a pre-selection step for the neighbors. They also implement a cluster-based smoothing technique in which missing values for users in a cluster are replaced by cluster representatives. Their method is reported to perform slightly better than standard k NN-based CF. In a similar way, Sarwar *et al.* [26] describe an approach to implement a scalable k NN classifier. They partition the user space by applying the *bisecting k-means* algorithm and then use those clusters as the base for neighborhood formation. They report a decrease in accuracy of around 5% as compared to standard k NN CF. However, their approach allows for a significant improvement in efficiency.

Connor and Herlocker [21] present a different approach in which, instead of users, they cluster items. Using the Pearson Correlation similarity measure they try out four different algorithms: average link hierarchical agglomerative [39], robust clustering algorithm for categorical attributes (ROCK) [40], kMetis, and hMetis³. Although clustering did improve efficiency, all of their clustering techniques yielded worse accuracy and coverage than the non-partitioned baseline. Finally, Li *et al.*[60] and Ungar and Foster [72] present a very similar approach for using *k-means* clustering for solving a probabilistic model interpretation of the recommender problem.

2.4.2 Alternatives to k-means

Density-based clustering algorithms such as DBSCAN work by building up on the definition of density as the number of points within a specified radius. DBSCAN, for instance, defines three kinds of points: *core points* are those that have more than a specified number of neighbors within a given distance; *border points* have fewer than the specified number but belong to a *core point* neighborhood; and *noise points* are those that are neither core or border. The algorithm iteratively removes *noise points* and performs clustering on the remaining points.

Message-passing clustering algorithms are a very recent family of graph-based clustering methods. Instead of considering an initial subset of the points as centers and then iteratively adapt those, message-passing algorithms initially consider all points as centers – usually known as *exemplars* in this context. During the algorithm execution points, which are now considered nodes in a network, exchange messages until clusters gradually emerge. *Affinity Propagation* is an important representative of this family of algorithms [33] that works by defining two kinds of messages between nodes: “responsibility”, which reflects how well-suited receiving point is to serve as exemplar of the point sending the message, taking into account other potential exemplars; and “availability”, which is sent from candidate exemplar to the point and reflects how appropriate it would be for the point to choose the candidate as its exemplar, taking into account support from other points that are choosing that same exemplar. Affinity propagation has been applied, with very good results, to

³ <http://www.cs.umn.edu/karypis/metis>

problems as different as DNA sequence clustering, face clustering in images, or text summarization.

Finally, **Hierarchical Clustering**, produces a set of nested clusters organized as a hierarchical tree (*dendrogram*). Hierarchical Clustering does not have to assume a particular number of clusters in advanced. Also, any desired number of clusters can be obtained by selecting the tree at the proper level. Hierarchical clusters can also sometimes correspond to meaningful taxonomies. Traditional hierarchical algorithms use a similarity or distance matrix and merge or split one cluster at a time. There are two main approaches to hierarchical clustering. In *agglomerative* hierarchical clustering we start with the points as individual clusters and at each step, merge the closest pair of clusters until only one cluster (or k clusters) are left. In *divisive* hierarchical clustering we start with one, all-inclusive cluster, and at each step, split a cluster until each cluster contains a point (or there are k clusters).

To the best of our knowledge, alternatives to k -means such as the previous have not been applied to RS. The simplicity and efficiency of the k -means algorithm shadows possible alternatives. It is not clear whether density-based or hierarchical clustering approaches have anything to offer in the RS arena. On the other hand, message-passing algorithms have been shown to be more efficient and their graph-based paradigm can be easily translated to the RS problem. It is possible that we see applications of these algorithms in the coming years.

2.5 Association Rule Mining

Association Rule Mining focuses on finding rules that will predict the occurrence of an item based on the occurrences of other items in a transaction. The fact that two items are found to be related means co-occurrence but not causality. Note that this technique should not be confused with rule-based classifiers presented in Sec. 2.3.3.

We define an *itemset* as a collection of one or more items (e.g. (Milk, Beer, Diaper)). A *k-itemset* is an itemset that contains k items. The frequency of a given itemset is known as *support count* (e.g. (Milk, Beer, Diaper) = 131). And the *support* of the itemset is the fraction of transactions that contain it (e.g. (Milk, Beer, Diaper) = 0.12). A *frequent itemset* is an itemset with a support that is greater or equal to a *minsup* threshold. An association rule is an expression of the form $X \Rightarrow Y$, where X and Y are itemsets. (e.g. $Milk, Diaper \Rightarrow Beer$). In this case the *support* of the association rule is the fraction of transactions that have both X and Y . On the other hand, the *confidence* of the rule is how often items in Y appear in transactions that contain X .

Given a set of transactions T , the goal of association rule mining is to find all rules having *support* \geq *minsupthreshold* and *confidence* \geq *minconfthreshold*. The brute-force approach would be to list all possible association rules, compute the support and confidence for each rule and then prune rules that do not satisfy both conditions. This is, however, computationally very expensive. For this reason, we take a two-step approach: (1) Generate all itemsets whose support \geq minsup

(Frequent Itemset Generation); (2) Generate high confidence rules from each frequent itemset (**Rule Generation**)

Several techniques exist to optimize the generation of frequent itemsets. On a broad sense they can be classified into those that try to minimize the number of candidates (M), those that reduce the number of transactions (N), and those that reduce the number of comparisons (NM). The most common approach though, is to reduce the number of candidates using the *Apriori principle*. This principle states that if an itemset is frequent, then all of its subsets must also be frequent. This is verified using the support measure because the support of an itemset never exceeds that of its subsets. The Apriori algorithm is a practical implementation of the principle.

Given a frequent itemset L , the goal when generating rules is to find all non-empty subsets that satisfy the minimum confidence requirement. If $|L| = k$, then there are $2^k - 1$ candidate association rules. So, as in the frequent itemset generation, we need to find ways to generate rules efficiently. For the Apriori Algorithm we can generate candidate rules by merging two rules that share the same prefix in the rule consequent.

The effectiveness of association rule mining for uncovering patterns and driving personalized marketing decisions has been known for a some time [2]. However, and although there is a clear relation between this method and the goal of a RS, they have not become mainstream. The main reason is that this approach is similar to item-based CF but is less flexible since it requires of an explicit notion of *transaction* – e.g. co-occurrence of events in a given session. In the next paragraphs we present some promising examples, some of which indicate that association rules still have not had their last word.

Mobasher *et al.* [53] present a system for web personalization based on association rules mining. Their system identifies association rules from pageviews co-occurrences based on users navigational patterns. Their approach outperforms a k NN-based recommendation system both in terms of precision and coverage. Smyth *et al.* [68] present two different case studies of using association rules for RS. In the first case they use the *a priori* algorithm to extract item association rules from user profiles in order to derive a better item-item similarity measure. In the second case, they apply association rule mining to a *conversational* recommender. The goal here is to find co-occurrent *critiques* – i.e. user indicating a preference over a particular feature of the recommended item. Lin *et al.* [49] present a new association mining algorithm that adjusts the minimum support of the rules during mining in order to obtain an appropriate number of significant rule therefore addressing some of the shortcomings of previous algorithms such as the *a priori*. They mine both association rules between users and items. The measured accuracy outperforms previously reported values for correlation-based recommendation and is similar to the more elaborate approaches such as the combination of SVD and ANN.

Finally, as already mentioned in section 2.3.2, Cho *et al.* [18] combine Decision Trees and Association Rule Mining in a web shop RS. In their system, association rules are derived in order to link related items. The recommendation is then computed by intersecting association rules with user preferences. They look for association rules in different transaction sets such as purchases, basket placement, and

click-through. They also use a heuristic for weighting rules coming from each of the transaction sets. Purchase association rules, for instance, are weighted higher than click-through association rules.

2.6 Conclusions

This chapter has introduced the main data mining methods and techniques that can be applied in the design of a RS. We have also surveyed their use in the literature and provided some rough guidelines on how and where they can be applied.

We started by reviewing techniques that can be applied in the pre-processing step. First, there is the choice of an appropriate distance measure, which is reviewed in Section 2.2.1. This is required by most of the methods in the following steps. The cosine similarity and Pearson correlation are commonly accepted as the best choice. Although there have been many efforts devoted to improving these distance measures, recent works seem to report that the choice of a distance function does not play such an important role. Then, in Section 2.2.2, we reviewed the basic sampling techniques that need to be applied in order to select a subset of an originally large data set, or to separating a training and a testing set. Finally, we discussed the use of dimensionality reduction techniques such as Principal Component Analysis and Singular Value Decomposition in Section 2.2.3 as a way to address the *curse of dimensionality* problem. We explained some success stories using dimensionality reduction techniques, especially in the context of the Netflix prize.

In Section 2.3, we reviewed the main classification methods: namely, nearest-neighbors, decision trees, rule-based classifiers, Bayesian networks, artificial neural networks, and support vector machines. We saw that, although k NN (see Section 2.3.1) CF is the preferred approach, all those classifiers can be applied in different settings. Decision trees (see Section 2.3.2) can be used to derive a model based on the content of the items or to model a particular part of the system. Decision rules (see Section 2.3.3) can be derived from a pre-existing decision trees, or can also be used to introduce business or domain knowledge. Bayesian networks (see Section 2.3.4) are a popular approach to content-based recommendation, but can also be used to derive a model-based CF system. In a similar way, Artificial Neural Networks can be used to derive a model-based recommender but also to combine/hybridize several algorithms. Finally, support vector machines (see Section 2.3.6) are gaining popularity also as a way to infer content-based classifications or derive a CF model.

Choosing the right classifier for a RS is not easy and is in many senses task and data-dependent. In the case of CF, some results seem to indicate that model-based approaches using classifiers such as the SVM or Bayesian Networks can slightly improve performance of the standard k NN classifier. However, those results are non-conclusive and hard to generalize. In the case of a content-based RS there is some evidence that in some cases Bayesian Networks will perform better than simpler

methods such as decision trees. However, it is not clear that more complex non-linear classifiers such as the ANN or SVMs can perform better.

Therefore, the choice of the right classifier for a specific recommending task still has nowadays much of exploratory. A practical rule-of-thumb is to start with the simplest approach and only introduce complexity if the performance gain obtained justifies it. The performance gain should of course balance different dimensions such as prediction accuracy or computational efficiency.

We reviewed clustering algorithms in Section 2.4. Clustering is usually used in RS to improve performance. A previous clustering step, either in the user or item space, reduces the number of distance computations we need to perform. However, this usually comes at the price of a lower accuracy so it should be handled with care. As a matter of fact, improving efficiency by using a dimensionality reduction technique such as SVD is probably a better choice in the general case. As opposed to what happens with classifiers, not so many clustering algorithms have been used in the context of RS. The simplicity and relative efficiency of the k -means algorithm (see Section 2.4.1) make it hard to find a practical alternative. We reviewed some of them such as Hierarchical Clustering or Message-passing algorithms in Section 2.4.2. Although these techniques have still not been applied for RS, they offer a promising avenue for future research.

Finally, in Section 2.5, we described association rules and surveyed their use in RS. Association rules offer an intuitive framework for recommending items whenever there is an explicit or implicit notion of *transaction*. Although there exist efficient algorithms for computing association rules, and they have proved more accurate than standard k NN CF, they are still not a favored approach.

The choice of the right DM technique in designing a RS is a complex task that is bound by many problem-specific constraints. However, we hope that the short review of techniques and experiences included in this chapter can help the reader make a much more informed decision. Besides, we have also uncovered areas that are open to many further improvements, and where there is still much exciting and relevant research to be done in the coming years.

Acknowledgments

This chapter has been written with partial support of an ICREA grant from the Generalitat de Catalunya.

References

1. Adomavicius, G., and Tuzhilin, A., Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

2. Agrawal, R., and Srikant, R., Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994.
3. Amatriain, X., Lathia, N., Pujol, J.M., Kwak, H., and Oliver, N., The wisdom of the few: A collaborative filtering approach based on expert opinions from the web. In *Proc. of SIGIR '09*, 2009.
4. Amatriain, X., Pujol, J.M., and Oliver, N., I like it... i like it not: Evaluating user ratings noise in recommender systems. In *UMAP '09*, 2009.
5. Amatriain, X., Pujol, J.M., Tintarev, N., and Oliver, N., Rate it again: Increasing recommendation accuracy by user re-rating. In *Recys '09*, 2009.
6. Anderson, M., Ball, M., Boley, H., Greene, S., Howse, N., Lemire, D., and S. McGrath. Racofi: A rule-applying collaborative filtering system. In *Proc. IEEE/WIC COLA '03*, 2003.
7. Baets, B.D., Growing decision trees in an ordinal setting. *International Journal of Intelligent Systems*, 2003.
8. Banerjee, S., and Ramanathan, K., Collaborative filtering on skewed datasets. In *Proc. of WWW '08*, 2008.
9. Basu, C., Hirsh, H., and Cohen, W., Recommendation as classification: Using social and content-based information in recommendation. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720. AAAI Press, 1998.
10. Basu, C., Hirsh, H., and Cohen, W., Recommendation as classification: Using social and content-based information in recommendation. In *AAAI Workshop on Recommender Systems*, 1998.
11. Bell, R.M., Koren, Y., and Volinsky, C., The bellkor solution to the netflix prize. Technical report, AT&T Labs Research, 2007.
12. Bouza, A., Reif, G., Bernstein, A., and Gall, H., Semtree: ontology-based decision tree algorithm for recommender systems. In *International Semantic Web Conference*, 2008.
13. Bozzon, A., Prandi, G., Valenzise, G., and Tagliasacchi, M., A music recommendation system based on semantic audio segments similarity. In *Proceeding of Internet and Multimedia Systems and Applications - 2008*, 2008.
14. Brand, M., Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining (SDM)*, 2003.
15. Breese, J., Heckerman, D., and Kadie, C., Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, page 4352, 1998.
16. Burke, R., Hybrid web recommender systems. pages 377–408. 2007.
17. Cheng, W., J. Hühn, and E. Hüllermeier. Decision tree and instance-based learning for label ranking. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 161–168, New York, NY, USA, 2009. ACM.
18. Cho, Y., Kim, J., and Kim, S., A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, 2002.
19. Christakou, C., and Stafylopatis, A., A hybrid movie recommender system based on neural networks. In *ISDA '05: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 500–505, 2005.
20. Cohen, W., Fast effective rule induction. In *Machine Learning: Proceedings of the 12th International Conference*, 1995.
21. Connor, M., and Herlocker, J., Clustering items for collaborative filtering. In *SIGIR Workshop on Recommender Systems*, 2001.
22. Cover, T., and Hart, P., Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.
23. Cristianini, N., and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000.
24. Deerwester, S., Dumais, S.T., Furnas, G.W., L. T. K., and Harshman, R., Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41, 1990.
25. Deshpande, M., and Karypis, G., Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.

26. B. S. et al. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, 2002.
27. K. O. et al. Context-aware svm for context-dependent information recommendation. In *International Conference On Mobile Data Management*, 2006.
28. P. T. et al. *Introduction to Data Mining*. Addison Wesley, 2005.
29. S. G. et al. Tv content recommender system. In *AAAI/IAAI 2000*, 2000.
30. S. H. et al. Amed- a personalized tv recommendation system. In *Interactive TV: a Shared Experience*, 2007.
31. T. B. et al. A trail based internet-domain recommender system using artificial neural networks. In *Proceedings of the Int. Conf. on Adaptive Hypermedia and Adaptive Web Based Systems*, 2002.
32. Freund, Y., Iyer, R., Schapire, R.E., and Singer, Y., An efficient boosting algorithm for combining preferences. *Mach. J., Learn. Res.*, 4:933–969, 2003.
33. Frey, B.J., and Dueck, D., Clustering by passing messages between data points. *Science*, 307, 2007.
34. Friedman, N., Geiger, D., and Goldszmidt, M., Bayesian network classifiers. *Mach. Learn.*, 29(2-3):131–163, 1997.
35. Funk, S., Netflix update: Try this at home, 2006.
36. Ghani, R., and Fano, A., Building recommender systems using a knowledge base of product semantics. In *In 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems*, 2002.
37. Goldberg, K., Roeder, T., Gupta, D., and Perkins, C., Eigentaste: A constant time collaborative filtering algorithm. *Journal Information Retrieval*, 4(2):133–151, July 2001.
38. Golub, G., and Reinsch, C., Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, April 1970.
39. Gose, E., Johnsonbaugh, R., and Jost, S., *Pattern Recognition and Image Analysis*. Prentice Hall, 1996.
40. Guha, S., Rastogi, R., and Shim, K., Rock: a robust clustering algorithm for categorical attributes. In *Proc. of the 15th Intl Conf. On Data Eng.*, 1999.
41. Hartigan, J.A., *Clustering Algorithms (Probability & Mathematical Statistics)*. John Wiley & Sons Inc, 1975.
42. Herlocker, J.L., Konstan, J.A., Terveen, L.G., and Riedl, J.T., Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.
43. Huang, Z., Zeng, D., and Chen, H., A link analysis approach to recommendation under sparse data. In *Proceedings of AMCIS 2004*, 2004.
44. Isaksson, A., Wallman, M., H. Göransson, and Gustafsson, M.G., Cross-validation and bootstrapping are unreliable in small sample classification. *Pattern Recognition Letters*, 29:1960–1965, 2008.
45. Jolliffe, I.T., *Principal Component Analysis*. Springer, 2002.
46. Kang, H., and Yoo, S., Svm and collaborative filtering-based prediction of user preference for digital fashion recommendation systems. *IEICE Transactions on Inf & Syst*, 2007.
47. Kurucz, M., Benczur, A.A., and Csalogany, K., Methods for large scale svd with missing values. In *Proceedings of KDD Cup and Workshop 2007*, 2007.
48. Lathia, N., Hailes, S., and Capra, L., The effect of correlation coefficients on communities of recommenders. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2000–2005, New York, NY, USA, 2008. ACM.
49. Lin, W., and Alvarez, S., Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery Journal*, 6(1), 2004.
50. M. R. McLaughlin and Herlocker, J.L., A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *Proc. of SIGIR '04*, 2004.
51. S. M. McNee, Riedl, J., and Konstan, J.A., Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1097–1101, New York, NY, USA, 2006. ACM Press.

52. Miyahara, K., and Pazzani, M.J., Collaborative filtering with the simple bayesian classifier. In *Pacific Rim International Conference on Artificial Intelligence*, 2000.
53. Mobasher, B., Dai, H., Luo, T., and Nakagawa, M., Effective personalization based on association rule discovery from web usage data. In *Workshop On Web Information And Data Management, WIDM '01*, 2001.
54. Nikovski, D., and Kulev, V., Induction of compact decision trees for personalized recommendation. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 575–581, New York, NY, USA, 2006. ACM.
55. M. P. Omahony. Detecting noise in recommender system databases. In *In Proceedings of the International Conference on Intelligent User Interfaces (IUI06), 29th 1st*, pages 109–115. ACM Press, 2006.
56. Paterek, A., Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop 2007*, 2007.
57. Pazzani, M.J., and Billsus, D., Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.
58. Pronk, V., Verhaegh, W., Proidl, A., and Tiemann, M., Incorporating user control into recommender systems based on naive bayesian classification. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 73–80, 2007.
59. Pyle, D., *Data Preparation for Data Mining*. Morgan Kaufmann, second edition edition, 1999.
60. Li, B., K.Q., Clustering approach for hybrid recommender system. In *Web Intelligence 03*, 2003.
61. Quinlan, J.R., Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.
62. Rendle, S., and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Recsys '08: Proceedings of the 2008 ACM conference on Recommender Systems*, 2008.
63. Rokach, L., Maimon, O., *Data Mining with Decision Trees: Theory and Applications*, World Scientific Publishing (2008).
64. Zhang, J., F.S., Ouyang, Y., and Makedon, F., Analysis of a low-dimensional linear model under recommendation attacks. In *Proc. of SIGIR '06*, 2006.
65. Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., Incremental svd-based algorithms for highly scalable recommender systems. In *5th International Conference on Computer and Information Technology (ICCIT)*, 2002.
66. Sarwar, B.M., Karypis, G., Konstan, J.A., and Riedl, J.T., Application of dimensionality reduction in recommender systems a case study. In *ACM WebKDD Workshop*, 2000.
67. Schlar, A., Tsikinovsky, A., Rokach, L., Meisels, A., and Antwarg, L., Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 261–264, New York, NY, USA, 2009. ACM.
68. Smyth, B., K. McCarthy, Reilly, J., D. O'Sullivan, L. McGinty, and Wilson, D., Case studies in association rule mining for recommender systems. In *Proc. of International Conference on Artificial Intelligence (ICAI '05)*, 2005.
69. Spertus, E., Sahami, M., and Buyukkokten, O., Evaluating similarity measures: A large-scale study in the orkut social network. In *Proceedings of the 2005 International Conference on Knowledge Discovery and Data Mining (KDD-05)*, 2005.
70. Tiemann, M., and Pauws, S., Towards ensemble learning for hybrid music recommendation. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 177–178, New York, NY, USA, 2007. ACM.
71. Toeschler, A., Jahrer, M., and Legenstein, R., Improved neighborhood-based algorithms for large-scale recommender systems. In *In KDD-Cup and Workshop 08*, 2008.
72. Ungar, L.H., and Foster, D.P., Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*, 2000.
73. Witten, I.H., and Frank, E., *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition edition, 2005.

74. Wu, M., Collaborative filtering via ensembles of matrix factorizations. In *Proceedings of KDD Cup and Workshop 2007*, 2007.
75. Xia, Z., Dong, Y., and Xing, G., Support vector machines for collaborative filtering. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 169–174, New York, NY, USA, 2006. ACM.
76. Xu, J., and Araki, K., A svm-based personal recommendation system for tv programs. In *Multi-Media Modelling Conference Proceedings*, 2006.
77. Xue, G., Lin, R., Yang, C., Xi, Q., Zeng, W., H., Yu, J., and Chen, Z., Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 2005 SIGIR*, 2005.
78. Yu, K., Tresp, V., and Yu, S., A nonparametric hierarchical bayesian framework for information filtering. In *SIGIR '04*, 2004.
79. Zhang, Y., and Koren, J., Efficient bayesian hierarchical user modeling for recommendation system. In *SIGIR 07*, 2007.
80. Ziegler, C., McNeen, S. M., Konstan, J.A., and Lausen, G., Improving recommendation lists through topic diversification. In *Proc. of WWW '05*, 2005.
81. Zurada, J., *Introduction to artificial neural systems*. West Publishing Co., St. Paul, MN, USA, 1992.

Chapter 3

Content-based Recommender Systems: State of the Art and Trends

Pasquale Lops, Marco de Gemmis and Giovanni Semeraro

Abstract Recommender systems have the effect of guiding users in a personalized way to interesting objects in a large space of possible options. *Content-based* recommendation systems try to recommend items similar to those a given user has liked in the past. Indeed, the basic process performed by a content-based recommender consists in matching up the attributes of a user profile in which preferences and interests are stored, with the attributes of a content object (item), in order to recommend to the user new interesting items. This chapter provides an overview of content-based recommender systems, with the aim of imposing a degree of order on the diversity of the different aspects involved in their design and implementation. The first part of the chapter presents the basic concepts and terminology of content-based recommender systems, a high level architecture, and their main advantages and drawbacks. The second part of the chapter provides a review of the state of the art of systems adopted in several application domains, by thoroughly describing both classical and advanced techniques for representing items and user profiles. The most widely adopted techniques for learning user profiles are also presented. The last part of the chapter discusses trends and future research which might lead towards the next generation of systems, by describing the role of User Generated Content as a way for taking into account evolving vocabularies, and the challenge of feeding users with serendipitous recommendations, that is to say surprisingly interesting items that they might not have otherwise discovered.

Pasquale Lops
Department of Computer Science, University of Bari “Aldo Moro”, Via E. Orabona, 4, Bari (Italy)
e-mail: lops@di.uniba.it

Marco de Gemmis
Department of Computer Science, University of Bari “Aldo Moro”, Via E. Orabona, 4, Bari (Italy)
e-mail: degemmis@di.uniba.it

Giovanni Semeraro
Department of Computer Science, University of Bari “Aldo Moro”, Via E. Orabona, 4, Bari (Italy)
e-mail: semeraro@di.uniba.it

3.1 Introduction

The abundance of information available on the Web and in Digital Libraries, in combination with their dynamic and heterogeneous nature, has determined a rapidly increasing difficulty in finding what we want when we need it and in a manner which best meets our requirements.

As a consequence, the role of user modeling and personalized information access is becoming crucial: users need a personalized support in sifting through large amounts of available information, according to their interests and tastes.

Many information sources embody recommender systems as a way of personalizing their content for users [73]. Recommender systems have the effect of guiding users in a personalized way to interesting or useful objects in a large space of possible options [17]. Recommendation algorithms use input about a customer's interests to generate a list of recommended items. At Amazon.com, recommendation algorithms are used to personalize the online store for each customer, for example showing programming titles to a software engineer and baby toys to a new mother [50].

The problem of recommending items has been studied extensively, and two main paradigms have emerged. *Content-based* recommendation systems try to recommend items similar to those a given user has liked in the past, whereas systems designed according to the *collaborative* recommendation paradigm identify users whose preferences are similar to those of the given user and recommend items they have liked [7].

In this chapter, a comprehensive and systematic study of content-based recommender systems is carried out. The intention is twofold:

- to provide an overview of state-of-the-art systems, by highlighting the techniques which revealed the most effective, and the application domains in which they have adopted;
- to present trends and directions for future research which might lead towards the next generation of content-based recommender systems.

The chapter is organized as follows. First, we present the basic concepts and terminology related to content-based recommenders. A classical framework for providing content-based recommendations is described, in order to understand the main components of the architecture, the process for producing recommendations and the advantages and drawbacks of using this kind of recommendation technique. Section 3.3 provides a thorough review of the state of the art of content-based systems, by providing details about the classical and advanced techniques for representing items to be recommended, and the methods for learning user profiles. Section 3.4 presents trends and future research in the field of content-based recommender systems, while conclusions are drawn in Section 3.5.

3.2 Basics of Content-based Recommender Systems

Systems implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user [63]. The profile is a structured representation of user interests, adopted to recommend new interesting items. The recommendation process basically consists in matching up the attributes of the user profile against the attributes of a content object. The result is a relevance judgment that represents the user's level of interest in that object. If a profile accurately reflects user preferences, it is of tremendous advantage for the effectiveness of an information access process. For instance, it could be used to filter search results by deciding whether a user is interested in a specific Web page or not and, in the negative case, preventing it from being displayed.

3.2.1 A High Level Architecture of Content-based Systems

Content-based Information Filtering (IF) systems need proper techniques for representing the items and producing the user profile, and some strategies for comparing the user profile with the item representation. The high level architecture of a content-based recommender system is depicted in Figure 3.1. The recommendation process is performed in three steps, each of which is handled by a separate component:

- **CONTENT ANALYZER** – When information has no structure (e.g. text), some kind of pre-processing step is needed to extract structured relevant information. The main responsibility of the component is to represent the content of items (e.g. documents, Web pages, news, product descriptions, etc.) coming from information sources in a form suitable for the next processing steps. Data items are analyzed by feature extraction techniques in order to shift item representation from the original information space to the target one (e.g. Web pages represented as keyword vectors). This representation is the input to the **PROFILE LEARNER** and **FILTERING COMPONENT**;
- **PROFILE LEARNER** – This module collects data representative of the user preferences and tries to generalize this data, in order to construct the user profile. Usually, the generalization strategy is realized through machine learning techniques [61], which are able to infer a model of user interests starting from items liked or disliked in the past. For instance, the **PROFILE LEARNER** of a Web page recommender can implement a relevance feedback method [75] in which the learning technique combines vectors of positive and negative examples into a prototype vector representing the user profile. Training examples are Web pages on which a positive or negative feedback has been provided by the user;
- **FILTERING COMPONENT** – This module exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. The result is a binary or continuous relevance judgment (com-

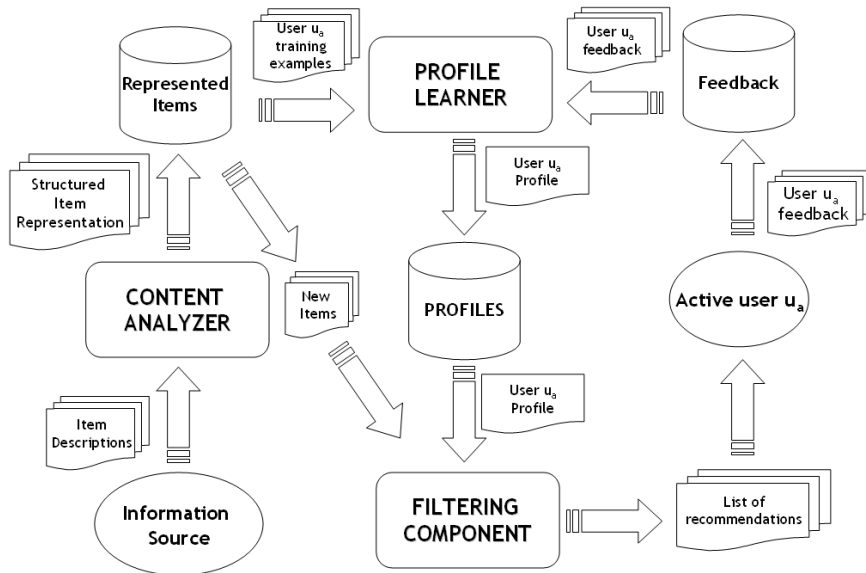


Fig. 3.1: High level architecture of a Content-based Recommender

puted using some similarity metrics [42]), the latter case resulting in a ranked list of potentially interesting items. In the above mentioned example, the matching is realized by computing the cosine similarity between the prototype vector and the item vectors.

The first step of the recommendation process is the one performed by the CONTENT ANALYZER, that usually borrows techniques from Information Retrieval systems [80, 6]. Item descriptions coming from *Information Source* are processed by the CONTENT ANALYZER, that extracts features (keywords, n-grams, concepts, ...) from unstructured text to produce a structured item representation, stored in the repository *Represented Items*.

In order to construct and update the *profile* of the *active user* u_a (user for which recommendations must be provided) her reactions to items are collected in some way and recorded in the repository *Feedback*. These reactions, called *annotations* [39] or *feedback*, together with the related item descriptions, are exploited during the process of learning a model useful to predict the actual relevance of newly presented items. Users can also explicitly define their areas of interest as an initial profile without providing any feedback.

Typically, it is possible to distinguish between two kinds of relevance feedback: positive information (inferring features liked by the user) and negative information (i.e., inferring features the user is not interested in [43]).

Two different techniques can be adopted for recording user's feedback. When a system requires the user to explicitly evaluate items, this technique is usually referred to as "explicit feedback"; the other technique, called "implicit feedback",

does not require any *active* user involvement, in the sense that feedback is derived from monitoring and analyzing user's activities.

Explicit evaluations indicate how relevant or interesting an item is to the user [74]. There are three main approaches to get explicit relevance feedback:

- *like/dislike* – items are classified as “relevant” or “not relevant” by adopting a simple binary rating scale, such as in [12];
- *ratings* – a discrete numeric scale is usually adopted to judge items, such as in [86]. Alternatively, symbolic ratings are mapped to a numeric scale, such as in Syskill & Webert [70], where users have the possibility of rating a Web page as *hot*, *lukewarm*, or *cold*;
- *text comments* – Comments about a single item are collected and presented to the users as a means of facilitating the decision-making process, such as in [72]. For instance, customer's feedback at Amazon.com or eBay.com might help users in deciding whether an item has been appreciated by the community. Textual comments are helpful, but they can overload the active user because she must read and interpret each comment to decide if it is positive or negative, and to what degree. The literature proposes advanced techniques from the affective computing research area [71] to make content-based recommenders able to automatically perform this kind of analysis.

Explicit feedback has the advantage of simplicity, albeit the adoption of numeric/symbolic scales increases the cognitive load on the user, and may not be adequate for catching user's feeling about items. Implicit feedback methods are based on assigning a relevance score to specific user actions on an item, such as saving, discarding, printing, bookmarking, etc. The main advantage is that they do not require a direct user involvement, even though biasing is likely to occur, e.g. interruption of phone calls while reading.

In order to build the profile of the active user u_a , the training set TR_a for u_a must be defined. TR_a is a set of pairs $\langle I_k, r_k \rangle$, where r_k is the rating provided by u_a on the item representation I_k . Given a set of item representation labeled with ratings, the PROFILE LEARNER applies supervised learning algorithms to generate a predictive model – the *user profile* – which is usually stored in a *profile repository* for later use by the FILTERING COMPONENT. Given a new item representation, the FILTERING COMPONENT predicts whether it is likely to be of interest for the active user, by comparing features in the item representation to those in the representation of user preferences (stored in the user profile). Usually, the FILTERING COMPONENT implements some strategies to rank potentially interesting items according to the relevance with respect to the user profile. Top-ranked items are included in a *list of recommendations* L_a , that is presented to u_a . User tastes usually change in time, therefore up-to-date information must be maintained and provided to the PROFILE LEARNER in order to automatically update the user profile. Further feedback is gathered on generated recommendations by letting users state their satisfaction or dissatisfaction with items in L_a . After gathering that feedback, the learning process is performed again on the new training set, and the resulting profile is adapted to the

updated user interests. The iteration of the feedback-learning cycle over time allows the system to take into account the dynamic nature of user preferences.

3.2.2 Advantages and Drawbacks of Content-based Filtering

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

- **USER INDEPENDENCE** - Content-based recommenders exploit solely ratings provided by the active user to build her own profile. Instead, collaborative filtering methods need ratings from other users in order to find the “nearest neighbors” of the active user, i.e., users that have similar tastes since they rated the same items similarly. Then, only the items that are most liked by the neighbors of the active user will be recommended;
- **TRANSPARENCY** - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes liked that item;
- **NEW ITEM** - Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects collaborative recommenders which rely solely on users’ preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

Nonetheless, content-based systems have several shortcomings:

- **LIMITED CONTENT ANALYSIS** - Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like. Some representations capture only certain aspects of the content, but there are many others that would influence a user’s experience. For instance, often there is not enough information in the word frequency to model the user interests in jokes or poems, while techniques for affective computing would be most appropriate. Again, for Web pages, feature extraction techniques from text completely ignore aesthetic qualities and additional multimedia information.

To sum up, both automatic and manually assignment of features to items could not be sufficient to define distinguishing aspects of items that turn out to be necessary for the elicitation of user interests.

- **OVER-SPECIALIZATION** - Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called *serendipity* problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty. To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be recommended just that kind of movies. A “perfect” content-based technique would rarely find anything *novel*, limiting the range of applications for which it would be useful.
- **NEW USER** - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

In the following, some strategies for tackling the above mentioned problems will be presented and discussed. More specifically, novel techniques for enhancing the content representation using common-sense and domain-specific knowledge will be described (Sections 3.3.1.3-3.3.1.4). This may help to overcome the limitations of traditional content analysis methods by providing new features, such as WordNet [60, 32] or Wikipedia concepts, which help to represent the items to be recommended in a more accurate and transparent way. Moreover, the integration of user-defined lexicons, such as folksonomies, in the process of generating recommendations will be presented in Section 3.4.1, as a way for taking into account evolving vocabularies.

Possible ways to feed users with *serendipitous* recommendations, that is to say, interesting items with a high degree of novelty, will be analyzed as a solution to the over-specialization problem (Section 3.4.2).

Finally, different strategies for overcoming the new user problem will be presented. Among them, social tags provided by users in a community can be exploited as a feedback on which recommendations are produced when few or no ratings for a specific user are available to the system (Section 3.4.1.1).

3.3 State of the Art of Content-based Recommender Systems

As the name implies, content-based filtering exploits the content of data items to predict its relevance based on the user’s profile. Research on content-based recommender systems takes place at the intersection of many computer science topics, especially Information Retrieval [6] and Artificial Intelligence.

From Information Retrieval (IR), research on recommendation technologies derives the vision that users searching for recommendations are engaged in an information seeking process. In IR systems the user expresses a one-off information need by giving a query (usually a list of keywords), while in IF systems the information need of the user is represented by her own profile. Items to be recommended can

be very different depending on the number and types of attributes used to describe them. Each item can be described through the same small number of attributes with known set of values, but this is not appropriate for items, such as Web pages, news, emails or documents, described through unstructured text. In that case there are no attributes with well-defined values, and the use of document modeling techniques with roots in IR research is desirable.

From an Artificial Intelligence perspective, the recommendation task can be cast as a learning problem that exploits past knowledge about users. At their simplest, user profiles are in the form of user-specified keywords or rules, and reflect the long-term interests of the user. Often, it is advisable for the recommender to learn the user profile rather than impose upon the user to provide one. This generally involves the application of Machine Learning (ML) techniques, whose goal is learning to categorize new information items based on previously seen information that have been explicitly or implicitly labelled as interesting or not by the user. Given these labelled information items, ML methods are able to generate a predictive model that, given a new information item, will help to decide whether it is likely to be of interest for the target user.

Section 3.3.1 describes alternative item representation techniques, ranging from traditional text representation, to more advanced techniques integrating ontologies and/or encyclopedic knowledge. Next, recommendation algorithms suitable for the described representations will be discussed in Section 3.3.2.

3.3.1 Item Representation

Items that can be recommended to the user are represented by a set of features, also called *attributes* or *properties*. For example, in a movie recommendation application, features adopted to describe a movie are: actors, directors, genres, subject matter, . . .). When each item is described by the same set of attributes, and there is a known set of values the attributes may take, the item is represented by means of structured data. In this case, many ML algorithms can be used to learn a user profile [69].

In most content-based filtering systems, item descriptions are textual features extracted from Web pages, emails, news articles or product descriptions. Unlike structured data, there are no attributes with well-defined values. Textual features create a number of complications when learning a user profile, due to the natural language ambiguity. The problem is that traditional keyword-based profiles are unable to capture the semantics of user interests because they are primarily driven by a string matching operation. If a string, or some morphological variant, is found in both the profile and the document, a match is made and the document is considered as relevant. String matching suffers from problems of:

- POLYSEMY, the presence of multiple meanings for one word;
- SYNONYMY, multiple words with the same meaning.

The result is that, due to synonymy, relevant information can be missed if the profile does not contain the exact keywords in the documents while, due to polysemy, wrong documents could be deemed relevant.

Semantic analysis and its integration in personalization models is one of the most innovative and interesting approaches proposed in literature to solve those problems. The key idea is the adoption of knowledge bases, such as lexicons or ontologies, for annotating items and representing profiles in order to obtain a “semantic” interpretation of the user information needs. In the next section, the basic keyword-based approach for document representation will be described, followed by a review of “traditional” systems relying on that model. Then, Sections 3.3.1.3 and 3.3.1.4 will provide an overview of techniques for semantic analysis based on ontological and world knowledge, respectively.

3.3.1.1 Keyword-based Vector Space Model

Most content-based recommender systems use relatively simple retrieval models, such as keyword matching or the Vector Space Model (VSM) with basic TF-IDF weighting. VSM is a spatial representation of text documents. In that model, each document is represented by a vector in a n -dimensional space, where each dimension corresponds to a term from the overall vocabulary of a given document collection. Formally, every document is represented as a vector of term weights, where each weight indicates the degree of association between the document and the term. Let $D = \{d_1, d_2, \dots, d_N\}$ denote a set of documents or corpus, and $T = \{t_1, t_2, \dots, t_n\}$ be the dictionary, that is to say the set of words in the corpus. T is obtained by applying some standard natural language processing operations, such as tokenization, stop-words removal, and stemming [6]. Each document d_j is represented as a vector in a n -dimensional vector space, so $d_j = \{w_{1j}, w_{2j}, \dots, w_{nj}\}$, where w_{kj} is the weight for term t_k in document d_j .

Document representation in the VSM raises two issues: weighting the terms and measuring the feature vector similarity. The most commonly used term weighting scheme, TF-IDF (Term Frequency-Inverse Document Frequency) *weighting*, is based on empirical observations regarding text [79]:

- rare terms are not less relevant than frequent terms (IDF assumption);
- multiple occurrences of a term in a document are not less relevant than single occurrences (TF assumption);
- long documents are not preferred to short documents (normalization assumption).

In other words, terms that occur frequently in one document (TF = term-frequency), but rarely in the rest of the corpus (IDF = inverse-document-frequency), are more likely to be relevant to the topic of the document. In addition, normalizing the resulting weight vectors prevent longer documents from having a better chance of retrieval. These assumptions are well exemplified by the TF-IDF function:

$$\text{TF-IDF}(t_k, d_j) = \underbrace{\text{TF}(t_k, d_j)}_{\text{TF}} \cdot \underbrace{\log \frac{N}{n_k}}_{\text{IDF}} \quad (3.1)$$

where N denotes the number of documents in the corpus, and n_k denotes the number of documents in the collection in which the term t_k occurs at least once.

$$\text{TF}(t_k, d_j) = \frac{f_{k,j}}{\max_z f_{z,j}} \quad (3.2)$$

where the maximum is computed over the frequencies $f_{z,j}$ of all terms t_z that occur in document d_j . In order for the weights to fall in the $[0, 1]$ interval and for the documents to be represented by vectors of equal length, weights obtained by Equation (3.1) are usually normalized by cosine normalization:

$$w_{k,j} = \frac{\text{TF-IDF}(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} \text{TF-IDF}(t_s, d_j)^2}} \quad (3.3)$$

which enforces the normalization assumption.

As stated earlier, a similarity measure is required to determine the closeness between two documents. Many similarity measures have been derived to describe the proximity of two vectors; among those measures, cosine similarity is the most widely used:

$$\text{sim}(d_i, d_j) = \frac{\sum_k w_{ki} \cdot w_{kj}}{\sqrt{\sum_k w_{ki}^2} \cdot \sqrt{\sum_k w_{kj}^2}} \quad (3.4)$$

In content-based recommender systems relying on VSM, both user profiles and items are represented as weighted term vectors. Predictions of a user's interest in a particular item can be derived by computing the cosine similarity.

3.3.1.2 Review of Keyword-based Systems

Several keyword-based recommender systems have been developed in a relatively short time, and it is possible to find them in various fields of applications, such as news, music, e-commerce, movies, etc. Each domain presents different problems, that require different solutions.

In the area of *Web recommenders*, famous systems in literature are *Letizia* [49], *Personal WebWatcher* [62, 63], *Syskill & Webert* [70, 68], *ifWeb* [4], *Amalthea* [66], and *WebMate* [23]. *Letizia* is implemented as a web-browser extension that tracks the user's browsing behavior and builds a personalized model consisting of keywords related to the user's interests. It relies on implicit feedback to infer the user's preferences. For example, bookmarking a page is interpreted as strong evidence for the user's interests in that page. In a similar way, *Personal WebWatcher* learns individual interests of users from the Web pages they visit, and from documents lying

one link away from the visited pages. It processes visited documents as positive examples of the user's interests, and non-visited documents as negative examples. *Amalthea* uses specific filtering agents to assist users in finding interesting information as well. User can specify filtering agents by providing pages (represented as weighted vectors) closely related to their interests.

The same approach is adopted by *Syskill & Webert*, that represents documents with the 128 most informative words (the "informativeness" of words in documents is determined in several different ways). Advanced representation techniques are adopted by *ifWeb*, that represents profiles in the form of a weighted semantic network. It supports explicit feedback and takes into account not only interests, but also explicit *disinterests*. Another interesting aspect is that it incorporates a mechanism for temporal decay, i.e. it ages the interests as expressed by the user. A different approach for representing user interests is adopted by *WebMate*, that keeps track of user interests in different domains by learning a user profile that consists of the keyword vectors that represents positive training examples. A profile of n keyword vectors can correctly represent up to n independent user interests.

In the field of *news filtering*, noteworthy recommender systems are *NewT* [87], *PSUN* [90], *INFOrmer* [91], *NewsDude* [12], *Daily Learner* [13], and *YourNews* [2]. *NewT* (News Tailor) allows users to provide positive and negative feedback on articles, part of articles, authors or sources. Several filtering agents are trained for different types of information, e.g. one for political news, one for sports, etc. In the same way *YourNews*, a more recent system for personalized news access, maintains a separate interest profile for 8 different topics (National, World, Business, etc.). The user interest profile for each topic is represented as a weighted prototype term vector extracted from the user's news view history. N articles from users' past views are collected, and the 100 top-weighted terms are extracted to generate the final prototype vectors. The system maintains short-term profiles by considering only the 20 most recently viewed news item, whereas long-term profiles consider all past views. The system can use profiles to suggest *recent* and *recommended* news.

Learning short-term and long-term profiles is quite typical of news filtering systems. *NewsDude* learns a short-term user model based on TF-IDF (cosine similarity), and a long-term model based on a naïve Bayesian classifier by relying on an initial training set of interesting news articles provided by the user. The news source is Yahoo! News. In the same way *Daily Learner*, a learning agent for wireless information access, adopts an approach for learning two separate user-models. The former, based on a Nearest Neighbor text classification algorithm, maintains the short-term interests of users, while the latter, based on a naïve Bayesian classifier, represents the long-term interests of users and relies on data collected over a longer period of time.

Among systems using a more complex representation for articles or profiles, *PSUN* and *INFOrmer* are worth to note. *PSUN* adopts an alternative representation for articles. Profiles are provided initially by presenting the system with some articles the user finds interesting. Recurring words in these articles are recorded by means of *n-grams* stored in a network of mutually attracting or repelling words, whose degree of attraction is determined by the number of co-occurrences. Each

user has multiple profiles that compete via a genetic algorithm, requiring explicit feedback. *INFormer* uses a semantic network for representing both user profiles and articles. A spreading activation technique [25] is used to compare articles and profiles, and a relevance feedback mechanism may be used to adapt the behavior of the system to user's changing interests. The pure spreading activation model consists of a network data structure consisting of nodes interconnected by links, that may be labeled and/or weighted. The processing starts by labeling a set of *source nodes* with activation weights and proceeds by iteratively propagating that activation to other nodes linked to the source nodes, until a termination condition ends the search process over the network.

A variety of content-based recommender systems exist in other application domains. *LIBRA* [65] implements a naïve Bayes text categorization method for book recommendation that exploits the product descriptions obtained from the Web pages of the Amazon on-line digital store. *Re:Agent* [16] is an intelligent email agent that can learn actions such as filtering, downloading to palmtops, forwarding email to voicemail, etc. using automatic feature extraction. *Re:Agent* users are required only to place example messages in folders corresponding to the desired actions. *Re:Agent* learns the concepts and decision policies from these folders. *Citeseeker* [15] assists the user in the process of performing a scientific literature search, by using word information and analyzing common citations in the papers. *INTIMATE* [53] recommends movies by using text categorization techniques to learn from movie synopses obtained from the Internet Movie Database¹. In order to get recommendations, the user is asked to rate a minimum number of movies into six categories: terrible, bad, below average, above average, good and excellent. In the same way, *Movies2GO* [67] learns user preferences from the synopsis of movies rated by the user. The innovative aspect of the system is to integrate voting schemes [93], designed to allow multiple individuals with conflicting preferences arrive at an acceptable compromise, and adapt them to manage conflicting preferences in a single user.

In the music domain, the commonly used technique for providing recommendations is collaborative filtering (see Last.fm² and MyStrands³ systems). The most noticeable system using (manual) content-based descriptions to recommend music is Pandora⁴. The main problem of the system is scalability, because the music annotation process is entirely done manually. Conversely, *FOAFing the music* [21, 22] is able to recommend, discover and explore music content, based on user profiling via *Friend of a Friend* (FOAF)⁵ descriptions, context-based information extracted from music related RSS feeds, and content-based descriptions automatically extracted from the audio itself.

In order to complete the survey of content-based recommender systems adopting the simple keyword-based vector space representation, we should also men-

¹ <http://www.imdb.com>

² <http://www.last.fm>

³ <http://www.mystrands.com>

⁴ <http://www.pandora.com>

⁵ <http://www.foaf-project.org>

tion some hybrid recommender systems that combine collaborative and content-based methods, such as *Fab* [7], *WebWatcher* [45], *P-Tango* [24], *ProfBuilder* [99], *PTV* [89], *Content-boosted Collaborative Filtering* [56], *CinemaScreen* [78] and the one proposed in [94].

The most important lesson learned from the analysis of the main systems developed in the last 15 years is that keyword-based representation for both items and profiles can give accurate performance, provided that a sufficient number of evidence of user interests is available. Most content-based systems are conceived as text classifiers built from training sets including documents which are either positive or negative examples of user interests. Therefore, accurate recommendations are achieved when training sets with a large number of examples are available, which guarantee reliable “syntactic” evidence of user interests. The problem with that approach is the “lack of intelligence”. When more advanced characteristics are required, keyword-based approaches show their limitations. If the user, for instance likes “French impressionism”, keyword-based approaches will only find documents in which the words “French” and “impressionism” occur. Documents regarding Claude Monet or Renoir exhibitions will not appear in the set of recommendations, even though they are likely to be very relevant for that user. More advanced representation strategies are needed in order to equip content-based recommender systems with “semantic intelligence”, which allows going beyond the syntactic evidence of user interests provided by keywords.

In the next sections, we will examine possible ways to infuse knowledge in the indexing phase by means of ontologies and encyclopedic knowledge sources.

3.3.1.3 Semantic Analysis by using Ontologies

Semantic analysis allows learning more accurate profiles that contain references to concepts defined in external knowledge bases. The main motivation for this approach is the challenge of providing a recommender system with the cultural and linguistic background knowledge which characterizes the ability of interpreting natural language documents and reasoning on their content.

In this section, a review of the main strategies adopted to introduce some semantics in the recommendation process is presented. The description of these strategies is carried out by taking into account several criteria:

- the type of knowledge source involved (e.g. lexicon, ontology, etc.);
- the techniques adopted for the annotation or representation of the items;
- the type of content included in the user profile;
- the item-profile matching strategy.

SiteIF [52] is a personal agent for a multilingual news Web site. To the best of our knowledge, it was the first system to adopt a sense-based document representation in order to build a model of the user interests. The external knowledge source involved in the representation process is MultiWordNet, a multilingual lexical database where English and Italian senses are aligned. Each news is automatically associated with

a list of MultiWordNet synsets by using Word Domain Disambiguation [51]. The user profile is built as a semantic network whose nodes represent synsets found in the documents read by the user. During the matching phase, the system receives as input the synset representation of a document and the current user model, and it produces as output an estimation of the document relevance by using the Semantic Network Value Technique [92].

ITR (ITem Recommender) is a system capable of providing recommendations for items in several domains (e.g., movies, music, books), provided that descriptions of items are available as text documents (e.g. plot summaries, reviews, short abstracts) [27, 83]. Similarly to SiteIF, ITR integrates linguistic knowledge in the process of learning user profiles, but Word Sense Disambiguation rather than Word Domain Disambiguation is adopted to obtain a sense-based document representation. The linguistic knowledge comes exclusively from the WordNet lexical ontology. Items are represented according to a synset-based vector space model, called bag-of-synsets (BOS), that is an extension of the classical bag-of-words (BOW) one [8, 84]. In the BOS model, a synset vector, rather than a word vector, corresponds to a document. The user profile is built as a Naïve Bayes binary text classifier able to categorize an item as interesting or not interesting. It includes those synsets that turn out to be most indicative of the user preferences, according to the value of the conditional probabilities estimated in the training phase. The item-profile matching consists in computing the probability for the item of being in the class “interesting”, by using the probabilities of synsets in the user profile.

SEWeP (Semantic Enhancement for Web Personalization) [31] is a Web personalization system that makes use of both the usage logs and the semantics of a Web site’s content in order to personalize it. A domain-specific taxonomy of categories has been used to semantically annotate Web pages, in order to have a uniform and consistent vocabulary. While the taxonomy is built manually, the annotation process is performed automatically. SEWeP, like SiteIF and ITR, makes use of the lexical knowledge stored in WordNet to “interpret” the content of an item and to support the annotation/representation process. Web pages are initially represented by keywords extracted from their content, then keywords are mapped to the concepts of the taxonomy. Given a keyword, a WordNet-based word similarity measure is applied to find the “closest” category word to that keyword. SEWeP does not build a personal profile of the user, rather it discovers navigational patterns. The categories which have been “semantically associated” to a pattern are used by the SEWeP recommendation engine to expand the recommendation set with pages characterized by the thematic categories that seem to be of interest for the user.

Quickstep [58] is a system for the recommendation of on-line academic research papers. The system adopts a research paper topic ontology based on the computer science classifications made by the DMOZ open directory project⁶ (27 classes used). Semantic annotation of papers consists in associating them with class names within the research paper topic ontology, by using a k-Nearest Neighbor classifier. Interest profiles are computed by correlating previously browsed research papers with

⁶ <http://www.dmoz.org/>

their classification. User profiles thus hold a set of topics and interest values in these topics. The item-profile matching is realized by computing a correlation between the top three interesting topics in the user profile and papers classified as belonging to those topics. *Foxtrot* [58] extends the Quickstep system by implementing a paper search interface, a profile visualization interface and an email notification, in addition to the Web page recommendation interface. Profile visualization is made possible because profiles are represented in ontological terms understandable to the users.

Informed Recommender [1] uses consumer product reviews to make recommendations. The system converts consumers' opinions into a structured form by using a translation ontology, which is exploited as a form of knowledge representation and sharing. The ontology provides a controlled vocabulary and relationships among words to describe: the consumer's skill level and experience with the product under review. To this purpose, the ontology contains two main parts: *opinion quality* and *product quality*, which formalize the two aforementioned aspects. A text-mining process automatically maps sentences in the reviews into the ontology information structure. The system does not build a profile of the user, rather it computes a set of recommendations on the basis of a user's request, e.g. the user asks about the quality of specific features of a product. *Informed Recommender* is able to answer to query and also recommends the best product according to the features the user is concerned with. Two aspects make this work noteworthy: one is that ontological knowledge can model different points of view according to which items can be annotated, the other is the use of review comments in the form of free text.

News@hand [18] is a system that adopts an ontology-based representation of item features and user preferences to recommend news. The annotation process associates the news with concepts belonging to the domain ontologies. A total of 17 ontologies have been used: they are adaptations of the IPTC ontology⁷, which contains concepts of multiple domains such as education, culture, politics, religion, science, sports, etc. It is not clear whether the annotation process is performed manually or by means of automated techniques such as text categorization. Item descriptions are vectors of TF-IDF scores in the space of concepts defined in the ontologies. User profiles are represented in the same space, except that a score measures the intensity of the user interest for a specific concept. Item-profile matching is performed as a cosine-based vector similarity.

A recommender system for *Interactive Digital Television* is proposed in [14], where the authors apply reasoning techniques borrowed from the Semantic Web in order to compare user preferences with items (TV programs) in a more flexible way, compared to the conventional syntactic metrics. The TV programs available during the recommendation process are annotated by metadata that describe accurately their main attributes. Both the knowledge about the TV domain and the user profiles are represented using an OWL ontology. Ontology-profiles provide a formal representation of the users' preferences, being able to *reason* about them and *discover* extra knowledge about their interests. The recommendation phase exploits

⁷ IPTC ontology, <http://nets.ii.uam.es/neptuno/iptc/>

the knowledge stored in the user profile to discover hidden semantic associations between the user's preferences and the available products. The inferred knowledge is processed and a spreading activation technique is adopted to suggest products to the user. The noteworthy aspect of this work is that ontology-profiles improve flat lists-based approaches which are not well structured to foster the discovery of new knowledge.

The JUMP System [10, 9] is capable of intelligent delivery of contextualized and personalized information to knowledge workers acting in their day-to-day working environment on non-routine tasks. The information needs of the JUMP user is represented in the form of a complex query, such as a *task support request*, rather than a user profile. An example of complex query is "I have to prepare a technical report for the VIKEF project". The semantic analysis of both documents and user information needs is based on a domain ontology in which concepts are manually annotated using WordNet synsets. The mapping between documents and domain/lexical concepts is performed automatically by means of Word Sense Disambiguation and Named Entity Recognition procedures, which exploit the lexical annotations in the domain ontology. The matching between concepts in the user request and documents is based on the relations in the domain ontology. For the processing of the example query, all instances of the concepts "technical report" and "project", and relations among these instances are extracted from the ontology.

The leading role of linguistic knowledge is highlighted by the wide use of WordNet, which is mostly adopted for the semantic interpretation of content by using word sense disambiguation. On the other hand, the studies described above showed that the great potential provided by WordNet is not sufficient alone for the full comprehension of the user interests and for their contextualization in the application domain. Domain specific knowledge is also needed. Ontologies play the fundamental role of formalizing the application domain, being exploited for the semantic descriptions of the items and for the representation of the concepts (i.e. classes and their instances) and relationships (i.e. hierarchical links and properties) identified in the domain. In conclusion, all studies which incorporated either linguistic or domain-specific knowledge or both in content-based filtering methods provided better and more accurate results compared to traditional content-based methods. This encourages researchers to design novel filtering methods which formalize and contextualize user interests by exploiting external knowledge sources such as thesauri or ontologies.

3.3.1.4 Semantic Analysis by using Encyclopedic Knowledge Sources

Common-sense and domain-specific knowledge may be useful to improve the effectiveness of natural language processing techniques by generating more informative features than the mere bag of words. The process of learning user profiles could benefit from the *infusion* of exogenous knowledge (externally supplied), with respect to the classical use of endogenous knowledge (extracted from the documents themselves). Many sources of world knowledge have become available in recent years.

Examples of general purpose knowledge bases include the Open Directory Project (ODP), Yahoo! Web Directory, and Wikipedia.

In the following we provide a brief overview of novel methods for generating new advanced features using world knowledge, even though those methods are not yet used in the context of learning user profiles.

Explicit Semantic Analysis (ESA) [34, 35] is a technique able to provide a fine-grained semantic representation of natural language texts in a high-dimensional space of natural (and also comprehensible) concepts derived from Wikipedia. Concepts are defined by Wikipedia articles, e.g., ITALY, COMPUTER SCIENCE, or RECOMMENDER SYSTEMS. The approach is inspired by the desire to augment text representation with massive amounts of world knowledge. In the case of Wikipedia as knowledge source, there are several advantages, such as its constant development by the community, the availability in several languages, and its high accuracy [37]. Empirical evaluations showed that ESA leads to substantial improvements in computing word and text relatedness, and in the text categorization task across a diverse collection of datasets. It has also been shown that ESA enhanced traditional BOW-based retrieval models [30].

Another interesting approach to add semantics to text is proposed by the Wikify! system [59, 26], which has the ability to identify important concepts in a text (key-word extraction), and then link these concepts to the corresponding Wikipedia pages (word sense disambiguation). The annotations produced by the Wikify! system can be used to automatically enrich documents with references to semantically related information. A Turing-like test to compare the quality of the system annotations to manual annotations produced by Wikipedia contributors has been designed. Human beings are asked to distinguish between manual and automatic annotations. Results suggest that the computer and human-generated Wikipedia annotations are hardly distinguishable, which indicates the high quality of the Wikify! system's annotations.

To the best of our knowledge, there are no (content-based) recommender systems able to exploit the above mentioned advanced semantic text representations for learning profiles containing references to world facts. The positive results obtained exploiting the advanced text representations in several tasks, such as semantic relatedness, text categorization and retrieval, suggest that similar positive results could be also obtained in the recommendation task. It seems a promising research area, not yet explored.

In [47], Wikipedia is used to estimate similarity between movies, in order to provide more accurate predictions for the Netflix Prize competition. More specifically, the content and the hyperlink structure of Wikipedia articles are exploited to identify similarities between movies. A similarity matrix containing the degree of similarity of each movie-movie pair is produced, and the prediction of user ratings from this matrix is computed by using a k-Nearest Neighbors and a Pseudo-SVD algorithm. Each of these methods combines the similarity estimates from Wikipedia with ratings from the training set to predict ratings in the test set. Unfortunately, these techniques did not show any significant improvement of the overall accuracy.

In [88], a quite complex, but not yet complete, approach for filtering RSS feeds and e-mails is presented. More specifically, the authors present an approach exploiting Wikipedia to automatically generate the user profile from the user's document collection. The approach mainly consists of four steps, namely the Wikipedia indexing, the profile generation, the problem-oriented index database creation, and the information filtering. The profile generation step exploits the collection of documents provided by the user, which implicitly represents a set of topics interesting for the user. A set of terms is extracted from each document, then similar Wikipedia articles are found by using the ESA algorithm. The system then extracts the list of Wikipedia categories from the articles and clusters these categories in order to get a subset of categories corresponding to one topic in the user profile. The user can also check her own profile and add or remove categories in order to refine topics. For each topic in the user profile, a problem-oriented Wikipedia corpus is created and indexed, and represents the base for filtering information.

In [85], a different approach to exploit Wikipedia in the content analysis step is presented. More specifically, the idea is to provide a *knowledge infusion* process into content-based recommender systems, in order to provide them with the *cultural* background knowledge that hopefully allows a more accurate content analysis than classical approaches based on words. The encyclopedic knowledge is useful to recognize specific domain-dependent concepts or named entities, especially in those contexts for which the adoption of domain ontologies is not feasible. Wikipedia entries have been modeled using Semantic Vectors, based on the *WordSpace* model [77], a vector space whose points are used to represent semantic concepts, such as words and documents. Relationships between words are then exploited by a spreading activation algorithm to produce new features that can be exploited in several ways during the recommendation process.

3.3.2 *Methods for Learning User Profiles*

Machine learning techniques, generally used in the task of inducing content-based profiles, are well-suited for text categorization [82]. In a machine learning approach to text categorization, an inductive process automatically builds a text classifier by learning from a set of *training documents* (documents labeled with the categories they belong to) the features of the categories.

The problem of learning user profiles can be cast as a binary text categorization task: each document has to be classified as interesting or not with respect to the user preferences. Therefore, the set of categories is $C = \{c_+, c_-\}$, where c_+ is the positive class (user-likes) and c_- the negative one (user-dislikes).

In the next sections we review the most used learning algorithms in content-based recommender systems. They are able to learn a function that models each user's interests. These methods typically require users to label documents by assigning a relevance score, and automatically infer profiles exploited in the filtering process to rank documents according to the user preferences.

3.3.2.1 Probabilistic Methods and Naïve Bayes

Naïve Bayes is a probabilistic approach to inductive learning, and belongs to the general class of Bayesian classifiers. These approaches generate a probabilistic model based on previously observed data. The model estimates the *a posteriori* probability, $P(c|d)$, of document d belonging to class c . This estimation is based on the *a priori* probability, $P(c)$, the probability of observing a document in class c , $P(d|c)$, the probability of observing the document d given c , and $P(d)$, the probability of observing the instance d . Using these probabilities, the Bayes theorem is applied to calculate $P(c|d)$:

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)} \quad (3.5)$$

To classify the document d , the class with the highest probability is chosen:

$$c = \operatorname{argmax}_{c_j} \frac{P(c_j)P(d|c_j)}{P(d)}$$

$P(d)$ is generally removed as it is equal for all c_j . As we do not know the value for $P(d|c)$ and $P(c)$, we estimate them by observing the training data. However, estimating $P(d|c)$ in this way is problematic, as it is very unlikely to see the same document more than once: the observed data is generally not enough to be able to generate good probabilities. The naïve Bayes classifier overcomes this problem by simplifying the model through the independence assumption: all the words or tokens in the observed document d are conditionally independent of each other given the class. Individual probabilities for the words in a document are estimated one by one rather than the complete document as a whole. The conditional independence assumption is clearly violated in real-world data, however, despite these violations, empirically the naïve Bayes classifier does a good job in classifying text documents [48, 11].

There are two commonly used working models of the naïve Bayes classifier, the *multivariate Bernoulli* event model and the *multinomial* event model [54]. Both models treat a document as a vector of values over the corpus vocabulary, V , where each entry in the vector represents whether a word occurred in the document, hence both models lose information about word order. The multivariate Bernoulli event model encodes each word as a binary attribute, i.e., whether a word appeared or not, while the multinomial event model counts how many times the word appeared in the document. Empirically, the multinomial naïve Bayes formulation was shown to outperform the multivariate Bernoulli model. This effect is particularly noticeable for large vocabularies [54]. The way the multinomial event model uses its document vector to calculate $P(c_j|d_i)$ is as follows:

$$P(c_j|d_i) = P(c_j) \prod_{w \in V_{d_i}} P(t_k|c_j)^{N_{(d_i,t_k)}} \quad (3.6)$$

where $N_{(d_i, t_k)}$ is defined as the number of times word or token t_k appeared in document d_i . Notice that, rather than getting the product of all the words in the corpus vocabulary V , only the subset of the vocabulary, V_{d_i} , containing the words that appear in the document d_i , is used.

A key step in implementing naïve Bayes is estimating the word probabilities $P(t_k|c_j)$. To make the probability estimates more robust with respect to infrequently encountered words, a smoothing method is used to modify the probabilities that would have been obtained by simple event counting. One important effect of smoothing is that it avoids assigning probability values equal to zero to words not occurring in the training data for a particular class. A rather simple smoothing method relies on the common Laplace estimates (i.e., adding one to all the word counts for a class). A more interesting method is Witten-Bell [100]. Although naïve Bayes performances are not as good as some other statistical learning methods such as nearest-neighbor classifiers or support vector machines, it has been shown that it can perform surprisingly well in the classification tasks where the computed probability is not important [29]. Another advantage of the naïve Bayes approach is that it is very efficient and easy to implement compared to other learning methods.

Although the classifiers based on the multinomial model significantly outperform those based on the multivariate one at large vocabulary sizes, their performance is unsatisfactory when: 1) documents in the training set have different lengths, thus resulting in a rough parameter estimation; 2) handling rare categories (few training documents available). These conditions frequently occur in the user profiling task, where no assumptions can be made on the length of training documents, and where obtaining an appropriate set of negative examples (i.e., examples of the class c_-) is problematic. Indeed, since users do not perceive having immediate benefits from giving negative feedback to the system [81], the training set for the class c_+ (user-likes) may be often larger than the one for the class c_- (user-dislikes). In [46], the authors propose a multivariate Poisson model for naïve Bayes text classification that allows more reasonable parameter estimation under the above mentioned conditions. We have adapted this approach to the case of user profiling task [36].

The naïve Bayes classifier has been used in several content-based recommendation systems, such as *Syskill & Webert* [70, 68], *NewsDude* [12], *Daily Learner* [13], *LIBRA* [65] and *ITR* [27, 83].

3.3.2.2 Relevance Feedback and Rocchio's Algorithm

Relevance feedback is a technique adopted in Information Retrieval that helps users to incrementally refine queries based on previous search results. It consists of the users feeding back into the system decisions on the relevance of retrieved documents with respect to their information needs.

Relevance feedback and its adaptation to text categorization, the well-known Rocchio's formula [75], are commonly adopted by content-based recommender systems. The general principle is to allow users to rate documents suggested by the recommender system with respect to their information need. This form of feedback can

subsequently be used to incrementally refine the user profile or to train the learning algorithm that infers the user profile as a classifier.

Some linear classifiers consist of an explicit profile (or prototypical document) of the category [82]. The Rocchio's method is used for inducing linear, profile-style classifiers. This algorithm represents documents as vectors, so that documents with similar content have similar vectors. Each component of such a vector corresponds to a term in the document, typically a word. The weight of each component is computed using the TF-IDF term weighting scheme. Learning is achieved by combining document vectors (of positive and negative examples) into a prototype vector for each class in the set of classes C . To classify a new document d , the similarity between the prototype vectors and the corresponding document vector representing d are calculated for each class (for example by using the cosine similarity measure), then d is assigned to the class whose document vector has the highest similarity value.

More formally, Rocchio's method computes a classifier $\vec{c}_i = \langle \omega_{1i}, \dots, \omega_{T|i} \rangle$ for the category c_i (T is the *vocabulary*, that is the set of distinct terms in the training set) by means of the formula:

$$\omega_{ki} = \beta \cdot \sum_{\{d_j \in POS_i\}} \frac{\omega_{kj}}{|POS_i|} - \gamma \cdot \sum_{\{d_j \in NEG_i\}} \frac{\omega_{kj}}{|NEG_i|} \quad (3.7)$$

where ω_{kj} is the TF-IDF weight of the term t_k in document d_j , POS_i and NEG_i are the set of positive and negative examples in the training set for the specific class c_j , β and γ are control parameters that allow to set the relative importance of *all* positive and negative examples. To assign a class \tilde{c} to a document d_j , the similarity between each prototype vector \vec{c}_i and the document vector \vec{d}_j is computed and \tilde{c} will be the c_i with the highest value of similarity. The Rocchio-based classification approach does not have any theoretic underpinning and there are guarantees on performance or convergence [69].

Relevance feedback has been used in several content-based recommendation systems, such as *YourNews* [2], *Fab* [7] and *NewT* [87].

3.3.2.3 Other Methods

Other learning algorithms have been used in content-based recommendation systems. A very brief description of the most important algorithms follows. A thorough review is presented in [64, 69, 82].

Decision trees are trees in which internal nodes are labeled by terms, branches departing from them are labeled by tests on the weight that the term has in the test document, and leaves are labeled by categories. Decision trees are learned by recursively partitioning training data, that is text documents, into subgroups, until those subgroups contain only instances of a single class. The test for partitioning data is run on the weights that the terms labeling the internal nodes have in the document. The choice of the term on which to operate the partition is generally

made according to an information gain or entropy criterion [101]. Decision trees are used in the *Syskill & Webert* [70, 68] recommender system.

Decision rule classifiers are similar to decision trees, because they operate in a similar way to the recursive data partitioning approach described above. An advantage of rule learners is that they tend to generate more compact classifiers than decision trees learners. Rule learning methods usually attempt to select from all the possible covering rules (i.e. rules that correctly classify all the training examples) the “best” one according to some minimality criterion.

Nearest neighbor algorithms, also called lazy learners, simply store training data in memory, and classify a new unseen item by comparing it to all stored items by using a similarity function. The “nearest neighbor” or the “ k -nearest neighbors” items are determined, and the class label for the unclassified item is derived from the class labels of the nearest neighbors. A similarity function is needed, for example the cosine similarity measure is adopted when items are represented using the VSM. Nearest neighbor algorithms are quite effective, albeit the most important drawback is their inefficiency at classification time, since they do not have a true training phase and thus defer all the computation to classification time. *Daily Learner* [13] and *Quickstep* [58] use the nearest neighbor algorithm to create a model of the user’s short term interest and for associating semantic annotations of papers with class names within the ontology, respectively.

3.4 Trends and Future Research

3.4.1 *The Role of User Generated Content in the Recommendation Process*

Web 2.0 is a term describing the trend in the use of World Wide Web technology that aims at promoting information sharing and collaboration among users. According to Tim O’Reilly⁸, the term “Web 2.0” means putting the user in the center, designing software that critically depends on its users since the content, as in Flickr, Wikipedia, Del.icio.us, or YouTube, is contributed by thousands or millions of users. That is why Web 2.0 is also called the “participative Web”. O’Reilly⁹ also defined Web 2.0 as “the design of systems that get better the more people use them”.

One of the forms of User Generated Content (UGC) that has drawn more attention from the research community is *folksonomy*, a taxonomy generated by users who collaboratively annotate and categorize resources of interests with freely chosen keywords called *tags*.

Despite the considerable amount of researches done in the context of recommender systems, the specific problem of integrating tags into standard recommender

⁸ <http://radar.oreilly.com/archives/2006/12/web-20-compact.html>, Accessed on March 18, 2009

⁹ <http://www.npr.org/templates/story/story.php?storyId=98499899>, Accessed on March 18, 2009

system algorithms, especially content-based ones, is less explored than the problem of recommending tags (i.e. assisting users for annotation purposes) [98, 95].

Folksonomies provide new opportunities and challenges in the field of recommender systems (see Chapter 19). It should be investigated whether they might be a valuable source of information about user interests and whether they could be included in user profiles. Indeed, several difficulties of tagging systems have been identified, such as polysemy and synonymy of tags, or the different expertise and purposes of tagging participants that may result in tags at various levels of abstraction to describe a resource, or the chaotic proliferation of tags [40].

3.4.1.1 Social Tagging Recommender Systems

Several methods have been proposed for taking into account user tagging activity within content-based recommender systems.

In [28], the user profile is represented in the form of a tag vector, with each element indicating the number of times a tag has been assigned to a document by that user. A more sophisticated approach is proposed in [57], which takes into account tag co-occurrence. The matching of profiles to information sources is achieved by using simple string matching. As the authors themselves foresee, the matching could be enhanced by adopting WORDNET.

In the work by Szomszor et al. [96], the authors describe a movie recommendation system built purely on the keywords assigned to movies via collaborative tagging. Recommendations for the active user are produced by algorithms based on the similarity between the keywords of a movie and those of the tag-clouds of movies she rated. As the authors themselves state, their recommendation algorithms can be improved by combining tag-based profiling techniques with more traditional content-based recommender strategies.

In [33], different strategies are proposed to build tag-based user profiles and to exploit them for producing music recommendations. Tag-based user profiles are defined as collections of tags, which have been chosen by a user to annotate tracks, together with corresponding scores representing the user interest in each of these tags, inferred from tag usage and frequencies of listened tracks.

While in the above described approaches only a single set of popular tags represents user interests, in [102] it is observed that this may not be the most suitable representation of a user profile, since it is not able to reflect the multiple interests of users. Therefore, the authors propose a network analysis technique (based on clustering), performed on the personal tags of a user to identify her different interests.

About tag interpretation, Cantador et al. [19] proposed a methodology to select “meaningful” tags from an initial set of raw tags by exploiting WORDNET, Wikipedia and Google. If a tag has an exact match in WORDNET, it is accepted, otherwise possible misspellings and compound nouns are discovered by using the Google “did you mean” mechanism (for example the tag *sanfrancisco* or *san farn-cisco* is corrected to *san francisco*). Finally, tags are correlated to their appropriate Wikipedia entries.

In the work by de Gemmis et al. [36], a more sophisticated approach, implementing a *hybrid* strategy for learning a user profile from both (static) content and tags associated with items rated by that user, is described. The authors include in the user profile not only her *personal* tags, but also the tags adopted by other users who rated the same items (*social* tags). This aspect is particularly important when users who contribute to the folksonomy have different expertise in the domain. The inclusion of social tags in the personal profile of a user allows also to extend the pure content-based recommendation paradigm toward a hybrid content-collaborative paradigm [17]. Furthermore, a solution to the challenging task of identifying user interests from tags is proposed. Since the main problem lies in the fact that tags are freely chosen by users and their actual meaning is usually not very clear, the authors suggested to semantically interpret tags by means of a Word Sense Disambiguation algorithm based on WORDNET. A similar hybrid approach, combining content-based profiles and interests revealed through tagging activities is also described in [38].

Some ideas on how to analyze tags by means of WORDNET in order to capture their intended meanings are also reported in [20], but suggested ideas are not supported by empirical evaluations. Another approach in which tags are semantically interpreted by means of WORDNET is the one proposed in [104]. The authors demonstrated the usefulness of tags in collaborative filtering, by designing an algorithm for neighbor selection that exploits a WORDNET-based semantic distance between tags assigned by different users.

We believe that it could be useful to investigate more on the challenging task of identifying the meaning of tags by relying on different knowledge sources such as WordNet or Wikipedia. Moreover, new strategies for integrating tags in the process of learning content-based profiles should be devised, by taking into account the different nature of personal, social and expert tags. Indeed, personal tags are mostly subjective and inconsistent, expert tags, on the other hand, are an attempt to be objective and consistent. Social tags leads to some form of coherence [5].

Another interesting research direction might be represented by the analysis of tags as a powerful kind of feedback to infer user profiles. Tags that express user opinions and emotions, such as boring, interesting, good, bad, etc., could represent a user's degree of satisfaction with an item. Techniques from the affective computing research area are needed.

3.4.2 Beyond Over-specialization: Serendipity

As introduced in Section 3.2.2, content-based systems suffer from over-specialization, since they recommend only items similar to those already rated by users. One possible solution to address this problem is the introduction of some randomness. For example, the use of genetic algorithms has been proposed in the context of information filtering [87]. In addition, the problem with over-specialization is not only that the content-based systems cannot recommend items that are different from anything the user has seen before. In certain cases, items should not be recommended if they are

too similar to something the user has already seen, such as a different news article describing the same event. Therefore, some content-based recommender systems, such as Daily-Learner [13], filter out items if they are too similar to something the user has seen before. The use of redundancy measures has been proposed by Zhang et al. [103] to evaluate whether a document that is deemed to be relevant contains some novel information as well. In summary, the *diversity* of recommendations is often a desirable feature in recommender systems.

Serendipity in a recommender can be seen as the experience of receiving an unexpected and fortuitous item recommendation, therefore it is a way to diversify recommendations. While people rely on exploration and luck to find new items that they did not know they wanted (e.g. a person may not know she likes watching talk shows until she accidentally turns to David Letterman), due to over-specialization, content-based systems have no inherent method for generating serendipitous recommendations, according to Gup's theory [41].

It is useful to make a clear distinction between *novelty* and *serendipity*. As explained by Herlocker [42], novelty occurs when the system suggests to the user an unknown item that she might have autonomously discovered. A serendipitous recommendation helps the user to find a surprisingly interesting item that she might not have otherwise discovered (or it would have been really hard to discover). To provide a clear example of the difference between novelty and serendipity, consider a recommendation system that simply recommends movies that were directed by the user's favorite director. If the system recommends a movie that the user was not aware of, the movie will be novel, but probably not serendipitous. On the other hand, a recommender that suggests a movie by a new director is more likely to provide serendipitous recommendations. Recommendations that are serendipitous are by definition also novel.

We look at the serendipity problem as the challenge of *programming* for serendipity, that is to find a manner to introduce serendipity in the recommendation process in an *operational* way. From this perspective, the problem has not been deeply studied, and there are really few theoretical and experimental studies.

Like Toms explains [97], there are three kinds of information searching:

1. seeking information about a well-defined object;
2. seeking information about an object that cannot be fully described, but that will be recognized at first sight;
3. acquiring information in an accidental, incidental, or serendipitous manner.

It is easy to realize that serendipitous happenings are quite useless for the first two ways of acquisition, but are extremely important for the third kind. As our discussion concerns the implementation of a serendipity-inducing strategy for a content-based recommender, the appropriate metaphor in a real-world situation could be one of a person going for shopping or visiting a museum who, while walking around seeking nothing in particular, would find something completely new that she has never expected to find, that is definitely interesting for her. Among different approaches which have been proposed for "operationally induced serendipity", Toms suggests four strategies, from simplistic to more complex ones [97]:

- Role of chance or *blind luck*, implemented via a random information node generator;
- Pasteur principle (“chance favors the prepared mind”), implemented via a user profile;
- Anomalies and exceptions, partially implemented via poor similarity measures;
- Reasoning by analogy, whose implementation is currently unknown.

In [44], it is described a proposal that implements the “Anomalies and exceptions” approach, in order to provide serendipitous recommendations alongside classical ones, thus providing the user with new entry points to the items in the system. The basic assumption is that the lower is the probability that user knows an item, the higher is the probability that a specific item could result in a serendipitous recommendation. The probability that a user knows something semantically near to what the system is confident she knows is higher than the probability of something semantically far. In other words, it is more likely to get a serendipitous recommendation by providing the user with something less similar to her profile. Following this principle, the basic idea underlying the system proposed in [44] is to ground the search for potentially *serendipitous items* on the similarity between the item descriptions and the user profile. The system is implemented as a naïve Bayes classifier, able to categorize an item as interesting (class c_+) or not (class c_-), depending on the a-posteriori probabilities computed by the classifier. In order to integrate Toms’ “poor similarity” within the recommender, the item-profile matching produces a list of items ranked according to the a-posteriori probability for the class c_+ . That list contains on the top the most similar items to the user profile, i.e. the items whose classification score for the class c_+ is high. On the other hand, the items for which the a-posteriori probability for the class c_- is higher, are ranked down in the list. The items on which the system is more uncertain are the ones for which the difference between the two classification scores for c_+ and c_- tends to zero. Therefore it is reasonable to assume that those items are not known by the user, since the system was not able to clearly classify them as relevant or not. The items for which the lowest difference between the two classification scores for c_+ and c_- is observed are the most uncertainly categorized, thus it might result to be the most serendipitous.

Regarding serendipity evaluation, there is a level of emotional response associated with serendipity that is difficult to capture, therefore an effective serendipity measurement should move beyond the conventional accuracy metrics and their associated experimental methodologies. New user-centric directions for evaluating new emerging aspects in recommender systems, such as serendipity of recommendations, are required [55]. Developing these measures constitutes an interesting and important research topic (see Chapter 8).

In conclusion, the adoption of strategies for realizing operational serendipity is an effective way to extend the capabilities of content-based recommender systems in order to mitigate the over-specialization problem, by providing the user with surprising suggestions.

3.5 Conclusions

In this chapter we surveyed the field of content-based recommender systems, by providing an overview of the most important aspects characterizing that kind of systems. Although there is a bunch of recommender systems in different domains, they share in common a means for representing items to be recommended and user profiles. The paper discusses the main issues related to the representation of items, starting from simple techniques for representing structured data, to more complex techniques coming from the Information Retrieval research area for unstructured data. We analyzed the main content recommender systems developed in the last 15 years, by highlighting the reasons for which a more complex “semantic analysis” of content is needed in order to go beyond the syntactic evidence of user interests provided by keywords. A review of the main strategies (and systems) adopted to introduce some semantics in the recommendation process is carried out, by providing evidence of the leading role of linguistic knowledge, even if a more specific knowledge is mandatory for a deeper understanding and contextualization of the user interests in different application domains. The latest issues in advanced text representation using sources of world knowledge, such as Wikipedia, have been highlighted, albeit they have not yet used in the context of learning user profiles. In order to complete the survey, a variety of learning algorithms have been described as well.

The last part of the chapter is devoted to the discussion of the main trends and research for the next generation of content-based recommender systems. More specifically, the chapter presents some aspects of the Web 2.0 (r)evolution, that changed the game for personalization, since the role of people evolved from passive consumers of information to that of active contributors. Possible strategies to integrate user-defined lexicons, such as folksonomies, as a way for taking into account evolving vocabularies are debated, by presenting some recent works and possible ideas for further investigations.

Finally, a very specific aspect of content recommender systems is presented. Due to the nature of this kind of systems, they can only recommend items that score highly against a user’s profile, thus the user is limited to being recommended items similar to those already rated. This shortcoming, called over-specialization, prevent these systems to be effectively used in real world scenarios. Possible ways to feed users with surprising and unexpected (serendipitous) recommendations are analyzed.

To conclude this survey, we want to underline the importance of research in language processing for advanced item representation in order to get more reliable recommendations. Just as an example, it is worth to cite the news published by the U.S. Patent and Trademark Office regarding series of intriguing patent applications from Google Inc. One of this patents, namely the Open Profile, for instance, would consider a user profile like “I really enjoy hiking, especially long hikes when you can camp out for a few days. Indoor activities don’t interest me at all, and I really don’t like boring outdoor activities like gardening”. Using smart language-processing algorithms to detect the user’s sentiments (“enjoy” or “don’t like” near “hiking” or

“gardening”) and other linguistic cues, the system would then potentially serve up active outdoor sports-related ads to this user but avoid ads about more hobbyist-oriented activities [3].

We hope that the issues presented in this chapter will contribute to stimulate the research community about the next generation of content-based recommendation technologies.

References

1. Aciar, S., Zhang, D., Simoff, S., Debenham, J.: Informed Recommender: Basing Recommendations on Consumer Product Reviews. *IEEE Intelligent Systems* **22**(3), 39–47 (2007)
2. Ahn, J., Brusilovsky, P., Grady, J., He, D., Syn, S.Y.: Open User Profiles for Adaptive News Systems: Help or Harm? In: C.L. Williamson, M.E. Zurko, P.F. Patel-Schneider, P.J. Shenoy (eds.) *Proceedings of the 16th International Conference on World Wide Web*, pp. 11–20. ACM (2007)
3. Anderson, M.: Google Searches for Ad Dollars in Social Networks. *IEEE Spectrum* **45**(12), 16 (2008)
4. Asnicar, F., Tasso, C.: ifWeb: a Prototype of User Model-based Intelligent Agent for Documentation Filtering and Navigation in the World Wide Web. In: C. Tasso, A. Jameson, C.L. Paris (eds.) *Proceedings of the First International Workshop on Adaptive Systems and User Modeling on the World Wide Web, Sixth International Conference on User Modeling*, pp. 3–12. Chia Laguna, Sardinia, Italy (1997)
5. Aurnhammer, M., Hanappe, P., Steels, L.: Integrating Collaborative Tagging and Emergent Semantics for Image Retrieval. In: *Proceedings of the WWW 2006 Collaborative Web Tagging Workshop* (2006)
6. Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*. Addison-Wesley (1999)
7. Balabanovic, M., Shoham, Y.: Fab: Content-based, Collaborative Recommendation. *Communications of the ACM* **40**(3), 66–72 (1997)
8. Basile, P., Degemmis, M., Gentile, A., Lops, P., Semeraro, G.: UNIBA: JIGSAW algorithm for Word Sense Disambiguation. In: *Proceedings of the 4th ACL 2007 International Workshop on Semantic Evaluations (SemEval-2007)*, Prague, Czech Republic, pp. 398–401. Association for Computational Linguistics (2007)
9. Basile, P., de Gemmis, M., Gentile, A., Iaquinta, L., Lops, P., Semeraro, G.: An Electronic Performance Support System Based on a Hybrid Content-Collaborative Recommender System. *Neural Network World: International Journal on Neural and Mass-Parallel Computing and Information Systems* **17**(6), 529–541 (2007)
10. Basile, P., de Gemmis, M., Gentile, A., Iaquinta, L., Lops, P.: The JUMP project: Domain Ontologies and Linguistic Knowledge @ Work. In: *Proceedings of the 4th Italian Semantic Web Applications and Perspectives - SWAP 2007, CEUR Workshop Proceedings*. CEUR-WS.org (2007)
11. Billsus, D., Pazzani, M.: Learning Probabilistic User Models. In: *Proceedings of the Workshop on Machine Learning for User Modeling*. Chia Laguna, IT (1997). URL citeseer.nj.nec.com/billsus96learning.html
12. Billsus, D., Pazzani, M.J.: A Hybrid User Model for News Story Classification. In: *Proceedings of the Seventh International Conference on User Modeling*. Banff, Canada (1999)
13. Billsus, D., Pazzani, M.J.: User Modeling for Adaptive News Access. *User Modeling and User-Adapted Interaction* **10**(2-3), 147–180 (2000)
14. Blanco-Fernandez, Y., Pazos-Arias J. J., G.S.A., Ramos-Cabrer, M., Lopez-Nores, M.: Providing Entertainment by Content-based Filtering and Semantic Reasoning in Intelligent Recommender Systems. *IEEE Transactions on Consumer Electronics* **54**(2), 727–735 (2008)

15. Bollacker, K.D., Giles, C.L.: CiteSeer: An Autonomous Web Agent for Automatic Retrieval and Identification of Interesting Publications. In: K. Sycara, M. Wooldridge (eds.) Proceedings of the Second International Conference on Autonomous Agents, pp. 116–123. ACM Press (1998)
16. Boone, G.: Concept Features in Re:Agent, an Intelligent Email Agent. In: K. Sycara, M. Wooldridge (eds.) Proceedings of the Second International Conference on Autonomous Agents, pp. 141–148. ACM Press (1998)
17. Burke, R.: Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction* **12**(4), 331–370 (2002)
18. Cantador, I., Bellogín, A., Castells, P.: News@hand: A Semantic Web Approach to Recommending News. In: W. Nejdl, J. Kay, P. Pu, E. Herder (eds.) Adaptive Hypermedia and Adaptive Web-Based Systems, *Lecture Notes in Computer Science*, vol. 5149, pp. 279–283. Springer (2008)
19. Cantador, I., Szomszor, M., Alani, H., Fernandez, M., Castells, P.: Ontological User Profiles with Tagging History for Multi-Domain Recommendations. In: Proceedings of the Collective Semantics: Collective Intelligence and the Semantic Web, CISWeb2008, Tenerife, Spain (2008)
20. Carmagnola, F., Cena, F., Cortassa, O., Gena, C., Torre, I.: Towards a Tag-Based User Model: How Can User Model Benefit from Tags? In: User Modeling 2007, *Lecture Notes in Computer Science*, vol. 4511, pp. 445–449. Springer (2007)
21. Celma, O., Ramírez, M., Herrera, P.: Foafing the Music: A Music Recommendation System based on RSS Feeds and User Preferences. In: 6th International Conference on Music Information Retrieval (ISMIR), pp. 464–467. London, UK (2005)
22. Celma, O., Serra, X.: FOAFing the Music: Bridging the Semantic Gap in Music Recommendation. *Web Semantics* **6**(4), 250–256 (2008)
23. Chen, L., Sycara, K.: WebMate: A Personal Agent for Browsing and Searching. In: K.P. Sycara, M. Wooldridge (eds.) Proceedings of the 2nd International Conference on Autonomous Agents, pp. 9–13. ACM Press, New York (1998)
24. Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., Sartin, M.: Combining Content-Based and Collaborative Filters in an Online Newspaper. In: Proceedings of ACM SIGIR Workshop on Recommender Systems (1999). URL citeseer.ist.psu.edu/claypool99combining.html
25. Collins, A.M., Loftus, E.F.: A Spreading Activation Theory of Semantic Processing. *Psychological Review* **82**(6), 407–428 (1975)
26. Csomai, A., Mihalcea, R.: Linking Documents to Encyclopedic Knowledge. *IEEE Intelligent Systems* **23**(5), 34–41 (2008)
27. Degenmis, M., Lops, P., Semeraro, G.: A Content-collaborative Recommender that Exploits WordNet-based User Profiles for Neighborhood Formation. *User Modeling and User-Adapted Interaction: The Journal of Personalization Research (UMUAI)* **17**(3), 217–255 (2007). Springer Science + Business Media B.V.
28. Diederich, J., Iofciu, T.: Finding Communities of Practice from User Profiles Based On Folksonomies. In: Innovative Approaches for Learning and Knowledge Sharing, EC-TEL Workshop Proc., pp. 288–297 (2006)
29. Domingos, P., Pazzani, M.J.: On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning* **29**(2-3), 103–130 (1997)
30. Egozi, O., Gabrilovich, E., Markovitch, S.: Concept-Based Feature Generation and Selection for Information Retrieval. In: D. Fox, C.P. Gomes (eds.) Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, pp. 1132–1137. AAAI Press (2008). ISBN 978-1-57735-368-3
31. Eirinaki, M., Vazirgiannis, M., Varlamis, I.: SEWeP: Using Site Semantics and a Taxonomy to enhance the Web Personalization Process. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 99–108. ACM (2003)
32. Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press (1998)

33. Firan, C.S., Nejdil, W., Paiu, R.: The Benefit of Using Tag-Based Profiles. In: Proceedings of the Latin American Web Conference, pp. 32–41. IEEE Computer Society, Washington, DC, USA (2007). DOI <http://dx.doi.org/10.1109/LA-WEB.2007.24>. ISBN 0-7695-3008-7
34. Gabrilovich, E., Markovitch, S.: Overcoming the Brittleness Bottleneck using Wikipedia: Enhancing Text Categorization with Encyclopedic Knowledge. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, pp. 1301–1306. AAAI Press (2006)
35. Gabrilovich, E., Markovitch, S.: Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis. In: M.M. Veloso (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1606–1611 (2007)
36. Gemmis, M.d., Lops, P., Semeraro, G., Basile, P.: Integrating Tags in a Semantic Content-based Recommender. In: Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys 2008, Lausanne, Switzerland, October 23-25, 2008, pp. 163–170 (2008)
37. Giles, J.: Internet Encyclopaedias Go Head to Head. *Nature* **438**, 900–901 (2005)
38. Godoy, D., Amandi, A.: Hybrid Content and Tag-based Profiles for Recommendation in Collaborative Tagging Systems. In: Proceedings of the 6th Latin American Web Congress (LA-WEB 2008), pp. 58–65. IEEE Computer Society (2008). ISBN 978-0-7695-3397-1
39. Goldberg, D., Nichols, D., Oki, B., Terry, D.: Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM* **35**(12), 61–70 (1992). URL <http://www.xerox.com/PARC/dlbox/tapestry-papers/TN44.ps>. Special Issue on Information Filtering
40. Golder, S., Huberman, B.A.: The Structure of Collaborative Tagging Systems. *Journal of Information Science* **32**(2), 198–208 (2006)
41. Gup, T.: Technology and the End of Serendipity. *The Chronicle of Higher Education* (44), 52 (1997)
42. Herlocker, L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems* **22**(1), 5–53 (2004)
43. Holte, R.C., Yan, J.N.Y.: Inferring What a User Is Not Interested in. In: G.I. McCalla (ed.) *Advances in Artificial Intelligence, Lecture Notes in Computer Science*, vol. 1081, pp. 159–171 (1996). ISBN 3-540-61291-2
44. Iaquinta, L., de Gemmis, M., Lops, P., Semeraro, G., Filannino, M., Molino, P.: Introducing Serendipity in a Content-based Recommender System. In: F. Xhafa, F. Herrera, A. Abraham, M. Köppen, J.M. Benitez (eds.) *Proceedings of the Eighth International Conference on Hybrid Intelligent Systems HIS-2008*, pp. 168–173. IEEE Computer Society Press, Los Alamitos, California (2008)
45. Joachims, T., Freitag, D., Mitchell, T.M.: Web Watcher: A Tour Guide for the World Wide Web. In: 15th International Joint Conference on Artificial Intelligence, pp. 770–777 (1997). URL citeseer.ist.psu.edu/article/joachims97webwatcher.html
46. Kim, S.B., Han, K.S., Rim, H.C., Myaeng, S.H.: Some Effective Techniques for Naïve Bayes Text Classification. *IEEE Trans. Knowl. Data Eng.* **18**(11), 1457–1466 (2006)
47. Lees-Miller, J., Anderson, F., Hoehn, B., Greiner, R.: Does Wikipedia Information Help Netflix Predictions? In: Seventh International Conference on Machine Learning and Applications (ICMLA), pp. 337–343. IEEE Computer Society (2008). ISBN 978-0-7695-3495-4
48. Lewis, D.D., Ringuette, M.: A Comparison of Two Learning Algorithms for Text Categorization. In: Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, pp. 81–93. Las Vegas, US (1994)
49. Lieberman, H.: Letizia: an Agent that Assists Web Browsing. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 924–929. Morgan Kaufmann (1995)
50. Linden, G., Smith, B., York, J.: Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* **7**(1), 76–80 (2003)
51. Magnini, B., Strapparava, C.: Experiments in Word Domain Disambiguation for Parallel Texts. In: Proc. of SIGLEX Workshop on Word Senses and Multi-linguality, Hong-Kong, October 2000. ACL (2000)

52. Magnini, B., Strapparava, C.: Improving User Modelling with Content-based Techniques. In: Proceedings of the 8th International Conference of User Modeling, pp. 74–83. Springer (2001)
53. Mak, H., Koprinska, I., Poon, J.: INTIMATE: A Web-Based Movie Recommender Using Text Categorization. In: Proceedings of the IEEE/WIC International Conference on Web Intelligence, pp. 602–605. IEEE Computer Society (2003). ISBN 0-7695-1932-6
54. McCallum, A., Nigam, K.: A Comparison of Event Models for Naïve Bayes Text Classification. In: Proceedings of the AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41–48. AAAI Press (1998)
55. McNee, S.M., Riedl, J., Konstan, J.A.: Accurate is not Always Good: How Accuracy Metrics have hurt Recommender Systems. In: Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (2006)
56. Melville, P., Mooney, R.J., Nagarajan, R.: Content-Boosted Collaborative Filtering for Improved Recommendations. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02), pp. 187–192. AAAI Press, Menlo Parc, CA, USA (2002)
57. Michlmayr, E., Cayzer, S.: Learning User Profiles from Tagging Data and Leveraging them for Personal(ized) Information Access. In: Proc. of the Workshop on Tagging and Metadata for Social Information Organization, Int. WWW Conf. (2007)
58. Middleton, S.E., Shadbolt, N.R., De Roure, D.C.: Ontological User Profiling in Recommender Systems. *ACM Transactions on Information Systems* **22**(1), 54–88 (2004)
59. Mihalcea, R., Csomai, A.: Wikify!: Linking Documents to Encyclopedic Knowledge. In: Proceedings of the sixteenth ACM conference on Conference on Information and Knowledge Management, pp. 233–242. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1321440.1321475>. ISBN 978-1-59593-803-9
60. Miller, G.: WordNet: An On-Line Lexical Database. *International Journal of Lexicography* **3**(4) (1990). (Special Issue)
61. Mitchell, T.: *Machine Learning*. McGraw-Hill, New York (1997)
62. Mladenic, D.: Machine learning used by Personal WebWatcher. In: Proceedings of ACAI-99 Workshop on Machine Learning and Intelligent Agents (1999)
63. Mladenic, D.: Text-learning and Related Intelligent Agents: A Survey. *IEEE Intelligent Systems* **14**(4), 44–54 (1999)
64. Montaner, M., Lopez, B., Rosa, J.L.D.L.: A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review* **19**(4), 285–330 (2003)
65. Mooney, R.J., Roy, L.: Content-Based Book Recommending Using Learning for Text Categorization. In: Proceedings of the 5th ACM Conference on Digital Libraries, pp. 195–204. ACM Press, New York, US, San Antonio, US (2000)
66. Moukas, A.: Amalthea Information Discovery and Filtering Using a Multiagent Evolving Ecosystem. *Applied Artificial Intelligence* **11**(5), 437–457 (1997)
67. Mukherjee, R., Jonsdottir, G., Sen, S., Sarathi, P.: MOVIES2GO: an Online Voting based Movie Recommender System. In: Proceedings of the Fifth International Conference on Autonomous Agents, pp. 114–115. ACM Press (2001)
68. Pazzani, M., Billsus, D.: Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning* **27**(3), 313–331 (1997)
69. Pazzani, M.J., Billsus, D.: Content-Based Recommendation Systems. In: P. Brusilovsky, A. Kobsa, W. Nejdl (eds.) *The Adaptive Web, Lecture Notes in Computer Science*, vol. 4321, pp. 325–341 (2007). ISBN 978-3-540-72078-2
70. Pazzani, M.J., Muramatsu, J., Billsus, D.: Syskill and Webert: Identifying Interesting Web Sites. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, pp. 54–61. AAAI Press / MIT Press, Menlo Park (1996)
71. Picard, R.W.: *Affective Computing*. MIT Press (2000)
72. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In: Proceedings of ACM 1994 Conference

- on Computer Supported Cooperative Work, pp. 175–186. ACM, Chapel Hill, North Carolina (1994). URL citeseer.ist.psu.edu/resnick94groupplens.html
73. Resnick, P., Varian, H.: Recommender Systems. *Communications of the ACM* **40**(3), 56–58 (1997)
 74. Rich, E.: User Modeling via Stereotypes. *Cognitive Science* **3**, 329–354 (1979)
 75. Rocchio, J.: Relevance Feedback Information Retrieval. In: G. Salton (ed.) *The SMART retrieval system - experiments in automated document processing*, pp. 313–323. Prentice-Hall, Englewood Cliffs, NJ (1971)
 76. Rokach, L., Maimon, O., *Data Mining with Decision Trees: Theory and Applications*, World Scientific Publishing (2008).
 77. Sahlgren, M.: *The Word-Space Model: Using Distributional Analysis to Represent Syntagmatic and Paradigmatic Relations between Words in High-dimensional Vector Spaces*. Ph.D. thesis, Stockholm: Stockholm University, Faculty of Humanities, Department of Linguistics (2006)
 78. Salter, J., Antonoupoulos, N.: CinemaScreen Recommender Agent: Combining collaborative and content-based filtering. *IEEE Intelligent Systems* **21**(1), 35–41 (2006)
 79. Salton, G.: *Automatic Text Processing*. Addison-Wesley (1989)
 80. Salton, G., McGill, M.: *Introduction to Modern Information Retrieval*. McGraw-Hill, New York (1983)
 81. Schwab, I., Kobsa, A., Koychev, I.: Learning User Interests through Positive Examples using Content Analysis and Collaborative Filtering (2001). URL citeseer.ist.psu.edu/schwab01learning.html
 82. Sebastiani, F.: Machine Learning in Automated Text Categorization. *ACM Computing Surveys* **34**(1) (2002)
 83. Semeraro, G., Basile, P., de Gemmis, M., Lops, P.: User Profiles for Personalizing Digital Libraries. In: Y.L. Theng, S. Foo, D.G.H. Lian, J.C. Na (eds.) *Handbook of Research on Digital Libraries: Design, Development and Impact*, pp. 149–158. IGI Global (2009). ISBN 978-159904879-6
 84. Semeraro, G., Degemmis, M., Lops, P., Basile, P.: Combining Learning and Word Sense Disambiguation for Intelligent User Profiling. In: M.M. Veloso (ed.) *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 2856–2861 (2007). ISBN 978-I-57735-298-3
 85. Semeraro, G., Lops, P., Basile, P., Gemmis, M.d.: Knowledge Infusion into Content-based Recommender Systems. In: *Proceedings of the 2009 ACM Conference on Recommender Systems, RecSys 2009, New York, USA, October 22–25, 2009* (2009). To appear
 86. Shardanand, U., Maes, P.: Social Information Filtering: Algorithms for Automating “Word of Mouth”. In: *Proceedings of ACM CHI’95 Conference on Human Factors in Computing Systems*, vol. 1, pp. 210–217 (1995). URL citeseer.ist.psu.edu/shardanand95social.html
 87. Sheth, B., Maes, P.: Evolving Agents for Personalized Information Filtering. In: *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, pp. 345–352. IEEE Computer Society Press (1993)
 88. Smirnov, A.V., Krizhanovsky, A.: Information Filtering based on Wiki Index Database. *CoRR abs/0804.2354* (2008)
 89. Smith, B., Cotter, P.: A Personalized TV Listings Service for the Digital TV Age. *Knowledge-Based Systems* **13**, 53–59 (2000)
 90. Sorensen, H., McElligott, M.: PSUN: A Profiling System for Usenet News. In: *Proceedings of CIKM ’95 Intelligent Information Agents Workshop* (1995)
 91. Sorensen, H., O’Riordan, A., O’Riordan, C.: Profiling with the INFormer Text Filtering Agent. *Journal of Universal Computer Science* **3**(8), 988–1006 (1997)
 92. Stefani, A., Strapparava, C.: Personalizing Access to Web Sites: The SiteIF Project. In: *Proc. of second Workshop on Adaptive Hypertext and Hypermedia*, Pittsburgh, June 1998 (1998)
 93. Straffin, P.D.J.: *Topics in the Theory of Voting*. The UMAP expository monograph series. Birkhauser (1980)

94. Symeonidis, P.: Content-based Dimensionality Reduction for Recommender Systems. In: C. Preisach, H. Burkhardt, L. Schmidt-Thieme, R. Decker (eds.) *Data Analysis, Machine Learning and Applications, Studies in Classification, Data Analysis, and Knowledge Organization*, pp. 619–626. Springer Berlin Heidelberg (2008). ISBN 978-3-540-78239-1
95. Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: Tag Recommendations based on Tensor Dimensionality Reduction. In: *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys 2008, Lausanne, Switzerland, October 23-25, 2008*, pp. 43–50 (2008)
96. Szomszor, M., Cattuto, C., Alani, H., O’Hara, K., Baldassarri, A., Loreto, V., Servedio, V.D.P.: Folksonomies, the Semantic Web, and Movie Recommendation. In: *Proceedings of the Workshop on Bridging the Gap between Semantic Web and Web 2.0 at the 4th ESWC (2007)*
97. Toms, E.: Serendipitous Information Retrieval. In: *Proceedings of DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries (2000)*
98. Tso-Sutter, K.H.L., Marinho, L.B., Schmidt-Thieme, L.: Tag-aware Recommender Systems by Fusion of Collaborative Filtering Algorithms. In: *SAC ’08: Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1995–1999. ACM (2008). ISBN 978-1-59593-753-7
99. Wasfi, A.M.: Collecting User Access Patterns for Building User Profiles and Collaborative Filtering. In: *Proceedings of the International Conference on Intelligent User Interfaces*, pp. 57–64 (1999)
100. Witten, I.H., Bell, T.: The Zero-frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression. *IEEE Transactions on Information Theory* **37**(4) (1991)
101. Yang, Y., Pedersen, J.O.: A Comparative Study on Feature Selection in Text Categorization. In: D.H. Fisher (ed.) *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pp. 412–420. Morgan Kaufmann Publishers, San Francisco, US, Nashville, US (1997). URL citeseer.ist.psu.edu/yang97comparative.html
102. Yeung, C.M.A., Gibbins, N., Shadbolt, N.: A Study of User Profile Generation from Folksonomies. In: *Proc. of the Workshop on Social Web and Knowledge Management, WWW Conf. (2008)*
103. Zhang, Y., Callan, J., Minka, T.: Novelty and Redundancy Detection in Adaptive Filtering. In: *Proceedings of the 25th International ACM SIGIR Conference*, pp. 81–88 (2002)
104. Zhao, S., Du, N., Nauertz, A., Zhang, X., Yuan, Q., Fu, R.: Improved Recommendation based on Collaborative Tagging Behaviors. In: *Proceedings of International Conference on Intelligent User Interfaces, IUI*, pp. 413–416. ACM (2008). ISBN 978-1-59593-987-6

Chapter 4

A Comprehensive Survey of Neighborhood-based Recommendation Methods

Christian Desrosiers and George Karypis

Abstract Among collaborative recommendation approaches, methods based on nearest-neighbors still enjoy a huge amount of popularity, due to their simplicity, their efficiency, and their ability to produce accurate and personalized recommendations. This chapter presents a comprehensive survey of neighborhood-based methods for the item recommendation problem. In particular, the main benefits of such methods, as well as their principal characteristics, are described. Furthermore, this document addresses the essential decisions that are required while implementing a neighborhood-based recommender system, and gives practical information on how to make such decisions. Finally, the problems of sparsity and limited coverage, often observed in large commercial recommender systems, are discussed, and a few solutions to overcome these problems are presented.

4.1 Introduction

The appearance and growth of online markets has had a considerable impact on the habits of consumers, providing them access to a greater variety of products and information on these goods. While this freedom of purchase has made online commerce into a multi-billion dollar industry, it also made it more difficult for consumers to select the products that best fit their needs. One of the main solutions proposed for this information overload problem are recommender systems, which provide automated and personalized suggestions of products to consumers. Recommender systems have been used in a wide variety of applications, such as the

Christian Desrosiers

Department of Software Engineering and IT, École de Technologie Supérieure, Montreal, Canada
e-mail: christian.desrosiers@etsmtl.ca

George Karypis

Department of Computer Science & Engineering, University of Minnesota, Minneapolis, USA
e-mail: karypis@cs.umn.edu

recommendation of books and CDs [47, 53], music [45, 70], movies [31, 51, 5], news [6, 41, 76], jokes [23], and web pages [7, 52].

The recommendation problem can be defined as estimating the response of a user for new items, based on historical information stored in the system, and suggesting to this user *novel* and *original* items for which the predicted response is *high*. The type of user-item responses varies from one application to the next, and falls in one of three categories: scalar, binary and unary. Scalar responses, also known as *ratings*, are numerical (e.g., 1-5 stars) or ordinal (e.g., strongly agree, agree, neutral, disagree, strongly disagree) values representing the possible levels of appreciation of users for items. Binary responses, on the other hand, only have two possible values encoding opposite levels of appreciation (e.g., like/dislike or interested/not interested). Finally, unary responses capture the interaction of a user with an item (e.g., purchase, online access, etc.) without giving explicit information on the appreciation of the user for this item. Since most users tend to interact with items that they find interesting, unary responses still provide useful information on the preferences of users.

The way in which user responses are obtained can also differ. For instance, in a movie recommendation application, users can enter ratings explicitly after watching a movie, giving their opinion on this movie. User responses can also be obtained implicitly from purchase history or access patterns [41, 76]. For example, the amount of time spent by a user browsing a specific type of item can be used as an indicator of the user's interest for this item type. For the purpose of simplicity, from this point on, we will call rating any type of user-item response.

4.1.1 Formal Definition of the Problem

In order to give a formal definition of the item recommendation task, we need to introduce some notation. Thus, the set of users in the system will be denoted by \mathcal{U} , and the set of items by \mathcal{I} . Moreover, we denote by \mathcal{R} the set of ratings recorded in the system, and write \mathcal{S} the set of possible values for a rating (e.g., $\mathcal{S} = [1, 5]$ or $\mathcal{S} = \{\text{like, dislike}\}$). Also, we suppose that no more than one rating can be made by any user $u \in \mathcal{U}$ for a particular item $i \in \mathcal{I}$ and write r_{ui} this rating. To identify the subset of users that have rated an item i , we use the notation \mathcal{U}_i . Likewise, \mathcal{I}_u represents the subset of items that have been rated by a user u . Finally, the items that have been rated by two users u and v , i.e. $\mathcal{I}_u \cap \mathcal{I}_v$, is an important concept in our presentation, and we use \mathcal{I}_{uv} to denote this concept. In a similar fashion, \mathcal{U}_{ij} is used to denote the set of users that have rated both items i and j .

Two of the most important problems associated with recommender systems are the *best item* and *top-N* recommendation problems. The first problem consists in finding, for a particular user u , the new item $i \in \mathcal{I} \setminus \mathcal{I}_u$ for which u is most likely to be interested in. When ratings are available, this task is most often defined as a regression or (multi-class) classification problem where the goal is to learn a function $f : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{S}$ that predicts the rating $f(u, i)$ of a user u for a new item i . This

function is then used to recommend to the active user u_a an item i^* for which the estimated rating has the highest value:

$$i^* = \arg \max_{j \in \mathcal{I} \setminus \mathcal{I}_u} f(u_a, j). \quad (4.1)$$

Accuracy is commonly used to evaluate the performance of the recommendation method. Typically, the ratings \mathcal{R} are divided into a *training* set $\mathcal{R}_{\text{train}}$ used to learn f , and a *test* set $\mathcal{R}_{\text{test}}$ used to evaluate the prediction accuracy. Two popular measures of accuracy are the *Mean Absolute Error* (MAE):

$$\text{MAE}(f) = \frac{1}{|\mathcal{R}_{\text{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\text{test}}} |f(u, i) - r_{ui}|, \quad (4.2)$$

and the *Root Mean Squared Error* (RMSE):

$$\text{RMSE}(f) = \sqrt{\frac{1}{|\mathcal{R}_{\text{test}}|} \sum_{r_{ui} \in \mathcal{R}_{\text{test}}} (f(u, i) - r_{ui})^2}. \quad (4.3)$$

When ratings are not available, for instance, if only the list of items purchased by each user is known, measuring the rating prediction accuracy is not possible. In such cases, the problem of finding the best item is usually transformed into the task of recommending to an active user u_a a list $L(u_a)$ containing N items likely to interest him or her [18, 45]. The quality of such method can be evaluated by splitting the items of \mathcal{I} into a set $\mathcal{I}_{\text{train}}$, used to learn L , and a test set $\mathcal{I}_{\text{test}}$. Let $T(u) \subset \mathcal{I}_u \cap \mathcal{I}_{\text{test}}$ be the subset of test items that a user u found relevant. If the user responses are binary, these can be the items that u has rated positively. Otherwise, if only a list of purchased or accessed items is given for each user u , then these items can be used as $T(u)$. The performance of the method is then computed using the measures of *precision* and *recall*:

$$\text{Precision}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| / |L(u)| \quad (4.4)$$

$$\text{Recall}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} |L(u) \cap T(u)| / |T(u)|. \quad (4.5)$$

A drawback of this task is that all items of a recommendation list $L(u)$ are considered equally interesting to user u . An alternative setting, described in [18], consists in learning a function L that maps each user u to a list $L(u)$ where items are *ordered* by their “interestingness” to u . If the test set is built by randomly selecting, for each user u , a single item i_u of \mathcal{I}_u , the performance of L can be evaluated with the *Average Reciprocal Hit-Rank* (ARHR):

$$\text{ARHR}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{\text{rank}(i_u, L(u))}, \quad (4.6)$$

where $\text{rank}(i_u, L(u))$ is the rank of item i_u in $L(u)$, equal to ∞ if $i_u \notin L(u)$. A more extensive description of evaluation measures for recommender systems can be found in Chapter 8 of this book.

4.1.2 Overview of Recommendation Approaches

While the recommendation problem truly emerged as an independent area of research in the mid 1990's, it has deeper roots in several other fields like cognitive science [42] and information retrieval [65]. Approaches for this problem are normally divided in two broad categories: *content-based* and *collaborative filtering* approaches.

4.1.2.1 Content-based approaches

The general principle of content-based (or cognitive) approaches [7, 8, 44, 58] is to identify the common characteristics of items that have received a favorable rating from a user u , and then recommend to u new items that share these characteristics. In content-based recommender systems, rich information describing the nature of each item i is assumed to be available in the form of a feature vector \mathbf{x}_i . For items in the form of text documents, such as news articles [8, 44] or Web documents [7, 58], this vector often contains the *Term Frequency-Inverse Document Frequency* (TF-IDF) [65] weights of the most informative keywords. Moreover, for each user u , a preference profile vector \mathbf{x}_u is usually obtained from the contents of items of \mathcal{I}_u . A technique to compute these profiles, used in several content-based recommender systems such as Newsweeder [44] and Fab [7], is the Rocchio algorithm [78, 12]. This technique updates the profile \mathbf{x}_u of user u whenever this user rates an item i by adding the weights of \mathbf{x}_i to \mathbf{x}_u , in proportion to the appreciation of u for i :

$$\mathbf{x}_u = \sum_{i \in \mathcal{I}_u} r_{ui} \mathbf{x}_i.$$

The user profiles can then be used to recommend new items to a user u , by suggesting the item whose feature vector \mathbf{x}_i is most similar to the profile vector \mathbf{x}_u , for example, using the cosine similarity [7, 8, 44] or the *Minimum Description Length* (MDL) [44, 62]. This approach can also be used to predict the rating of user u for a new item i [44], by building for each rating value $r \in \mathcal{S}$ a content profile vector $\mathbf{x}_u^{(r)}$ as the average of the feature vectors of items that have received this rating value from u . The predicted rating \hat{r}_{ui} for item i is the value r for which $\mathbf{x}_u^{(r)}$ is most similar to \mathbf{x}_i . Bayesian approaches using content information have also been proposed to predict ratings [8, 53, 58].

Recommender systems based purely on content generally suffer from the problems of *limited content analysis* and *over-specialization* [70]. Limited content anal-

ysis stems from the fact that the system may have only a limited amount of information on its users or the content of its items. The reasons for this lack of information can be numerous. For instance, privacy issues might refrain a user from providing personal information, or the precise content of items may be difficult or costly to obtain for some types of items, such as music or images. Finally, the content of an item is often insufficient to determine its quality. For example, it may be impossible to distinguish between a well written and a badly written article if both use the same terms. Over-specialization, on the other hand, is a side effect of the way in which content-based systems recommend new items, where the predicted rating of a user for an item is high if this item is similar to the ones liked by this user. For example, in a movie recommendation application, the system may recommend to a user a movie of the same genre or having the same actors as movies already seen by this user. Because of this, the system may fail to recommend items that are different but still interesting to the user. Solutions proposed for this problem include adding some randomness [71] or filtering out items that are too similar [8, 77]. More information on content-based recommendation approaches can be found in Chapter 3 of this book.

4.1.2.2 Collaborative filtering approaches

Unlike content-based approaches, which use the content of items previously rated by a user u , collaborative (or social) filtering approaches [18, 31, 41, 47, 60, 45, 70] rely on the ratings of u as well as those of other users in the system. The key idea is that the rating of u for a new item i is likely to be similar to that of another user v , if u and v have rated other items in a similar way. Likewise, u is likely to rate two items i and j in a similar fashion, if other users have given similar ratings to these two items.

Collaborative approaches overcome some of the limitations of content-based ones. For instance, items for which the content is not available or difficult to obtain can still be recommended to users through the feedback of other users. Furthermore, collaborative recommendations are based on the quality of items as evaluated by peers, instead of relying on content that may be a bad indicator of quality. Finally, unlike content-based systems, collaborative filtering ones can recommend items with very different content, as long as other users have already shown interest for these different items.

Following [1, 5, 10, 18], collaborative filtering methods can be grouped in the two general classes of *neighborhood* and *model*-based methods. In neighborhood-based (memory-based [10] or heuristic-based [1]) collaborative filtering [17, 18, 31, 41, 47, 54, 60, 45, 70], the user-item ratings stored in the system are directly used to predict ratings for new items. This can be done in two ways known as *user-based* or *item-based* recommendation. User-based systems, such as GroupLens [41], Bellcore video [31], and Ringo [70], evaluate the interest of a user u for an item i using the ratings for this item by other users, called *neighbors*, that have similar rating patterns. The neighbors of user u are typically the users v whose ratings on

the items rated by both u and v , i.e. \mathcal{I}_{uv} , are most correlated to those of u . Item-based approaches [18, 47, 45], on the other hand, predict the rating of a user u for an item i based on the ratings of u for items similar to i . In such approaches, two items are similar if several users of the system have rated these items in a similar fashion.

In contrast to neighborhood-based systems, which use the stored ratings directly in the prediction, model-based approaches use these ratings to learn a predictive model. The general idea is to model the user-item interactions with factors representing latent characteristics of the users and items in the system, like the preference class of users and the category class of items. This model is then trained using the available data, and later used to predict ratings of users for new items. Model-based approaches for the task of recommending items are numerous and include Bayesian Clustering [10], Latent Semantic Analysis [32], Latent Dirichlet Allocation [9], Maximum Entropy [78], Boltzmann Machines [64], Support Vector Machines [27], and Singular Value Decomposition [4, 42, 57, 74, 75]. A survey of state-of-the-art model-based methods can be found in Chapter 5 of this book.

4.1.3 Advantages of Neighborhood Approaches

While recent investigations show that state-of-the-art model-based approaches are superior to neighborhood ones in the task of predicting ratings [42, 73], there is also an emerging understanding that good prediction accuracy alone does not guarantee users an effective and satisfying experience [25]. Another factor that has been identified as playing an important role in the appreciation of users for the recommender system is *serendipity* [25, 45]. Serendipity extends the concept of novelty by helping a user find an interesting item he might not have otherwise discovered. For example, recommending to a user a movie directed by his favorite director constitutes a novel recommendation if the user was not aware of that movie, but is likely not serendipitous since the user would have discovered that movie on his own.

Model-based approaches excel at characterizing the preferences of a user with latent factors. For example, in a movie recommender system, such methods may determine that a given user is a fan of movies that are both funny and romantic, without having to actually define the notions “funny” and “romantic”. This system would be able to recommend to the user a romantic comedy that may not have been known to this user. However, it may be difficult for this system to recommend a movie that does not quite fit this high-level genre, for instance, a funny parody of horror movies. Neighborhood approaches, on the other hand, capture local associations in the data. Consequently, it is possible for a movie recommender system based on this type of approach to recommend a movie very different from the users usual taste or a movie that is not well known (e.g. repertoire film), if one of his closest neighbors has given it a strong rating. This recommendation may not be a guaranteed success, as would be a romantic comedy, but it may help the user discover a whole new genre or a new favorite actor/director.

The main advantages of neighborhood-based methods are:

- **Simplicity:** Neighborhood-based methods are intuitive and relatively simple to implement. In their simplest form, only one parameter (the number of neighbors used in the prediction) requires tuning.
- **Justifiability:** Such methods also provide a concise and intuitive justification for the computed predictions. For example, in item-based recommendation, the list of neighbor items, as well as the ratings given by the user to these items, can be presented to the user as a justification for the recommendation. This can help the user better understand the recommendation and its relevance, and could serve as basis for an interactive system where users can select the neighbors for which a greater importance should be given in the recommendation [4]. The necessity of explaining recommendations to users is addressed in Chapter 15 of this book.
- **Efficiency:** One of the strong points of neighborhood-based systems is their efficiency. Unlike most model-based systems, they require no costly training phases, which need to be carried out at frequent intervals in large commercial applications. While the recommendation phase is usually more expensive than for model-based methods, the nearest-neighbors can be pre-computed in an offline step, providing near instantaneous recommendations. Moreover, storing these nearest neighbors requires very little memory, making such approaches scalable to applications having millions of users and items.
- **Stability:** Another useful property of recommender systems based on this approach is that they are little affected by the constant addition of users, items and ratings, which are typically observed in large commercial applications. For instance, once item similarities have been computed, an item-based system can readily make recommendations to new users, without having to re-train the system. Moreover, once a few ratings have been entered for a new item, only the similarities between this item and the ones already in the system need to be computed.

4.1.4 Objectives and Outline

This chapter has two main objectives. It first serves as a general guide on neighborhood-based recommender systems, and presents practical information on how to implement such recommendation approaches. In particular, the main components of neighborhood-based methods will be described, as well as the benefits of the most common choices for each of these components. Secondly, it presents more specialized techniques addressing particular aspects of recommending items, such as data sparsity. Although such techniques are not required to implement a simple neighborhood-based system, having a broader view of the various difficulties and solutions for neighborhood methods may help with making appropriate decisions during the implementation process.

The rest of this document is structured as follows. In Section 4.2, the principal neighborhood approaches, predicting user ratings for new items based on regres-

sion or classification, are introduced, and the main advantages and flaws of these approaches are described. This section also presents two complementary ways of implementing such approaches, either based on user or item similarities, and analyses the impact of these two implementations on the accuracy, efficiency, stability, justifiability and serendipity of the recommender system. Section 4.3, on the other hand, focuses on the three main components of neighborhood-based recommendation methods: rating normalization, similarity weight computation, and neighborhood selection. For each of these components, the most common approaches are described, and their respective benefits compared. In Section 4.4, the problems of limited coverage and data sparsity are introduced, and several solutions are described to overcome these problems are described. In particular, several techniques based on dimensionality reduction and graphs are presented. Finally, the last section of this document summarizes the principal characteristics and methods of neighborhood-based recommendation, and gives a few more pointers on implementing such methods.

4.2 Neighborhood-based Recommendation

Recommender systems based on nearest-neighbors automate the common principle of *word-of-mouth*, where one relies on the opinion of like-minded people or other trusted sources to evaluate the value of an item (movie, book, articles, album, etc.) according to his own preferences. To illustrate this, consider the following example based on the ratings of Figure 4.1.

Example 4.1. User Eric has to decide whether or not to rent the movie “Titanic” that he has not yet seen. He knows that Lucy has very similar tastes when it comes to movies, as both of them hated “The Matrix” and loved “Forrest Gump”, so he asks her opinion on this movie. On the other hand, Eric finds out he and Diane have different tastes, Diane likes action movies while he does not, and he discards her opinion or considers the opposite in his decision.

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2	?	3	5	4
Diane	4	3	5	3	

Fig. 4.1: A “toy example” showing the ratings of four users for five movies.

4.2.1 User-based Rating Prediction

User-based neighborhood recommendation methods predict the rating r_{ui} of a user u for a new item i using the ratings given to i by users most similar to u , called nearest-neighbors. Suppose we have for each user $v \neq u$ a value w_{uv} representing the preference similarity between u and v (how this similarity can be computed will be discussed in Section 4.3.2). The k -nearest-neighbors (k -NN) of u , denoted by $\mathcal{N}(u)$, are the k users v with the highest similarity w_{uv} to u . However, only the users who have rated item i can be used in the prediction of r_{ui} , and we instead consider the k users most similar to u that *have rated* i . We write this set of neighbors as $\mathcal{N}_i(u)$. The rating r_{ui} can be estimated as the average rating given to i by these neighbors:

$$\hat{r}_{ui} = \frac{1}{|\mathcal{N}_i(u)|} \sum_{v \in \mathcal{N}_i(u)} r_{vi}. \quad (4.7)$$

A problem with (4.7) is that it does not take into account the fact that the neighbors can have different levels of similarity. Consider once more the example of Figure 4.1. If the two nearest-neighbors of Eric are Lucy and Diane, it would be foolish to consider equally their ratings of the movie “Titanic”, since Lucy’s tastes are much closer to Eric’s than Diane’s. A common solution to this problem is to weigh the contribution of each neighbor by its similarity to u . However, if these weights do not sum to 1, the predicted ratings can be well outside the range of allowed values. Consequently, it is customary to normalize these weights, such that the predicted rating becomes

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \quad (4.8)$$

In the denominator of (4.8), $|w_{uv}|$ is used instead of w_{uv} because negative weights can produce ratings outside the allowed range. Also, w_{uv} can be replaced by w_{uv}^α , where $\alpha > 0$ is an amplification factor [10]. When $\alpha > 1$, as is most often employed, an even greater importance is given to the neighbors that are the closest to u .

Example 4.2. Suppose we want to use (4.8) to predict Eric’s rating of the movie “Titanic” using the ratings of Lucy and Diane for this movie. Moreover, suppose the similarity weights between these neighbors and Eric are respectively 0.75 and 0.15. The predicted rating would be

$$\hat{r} = \frac{0.75 \times 5 + 0.15 \times 3}{0.75 + 0.15} \simeq 4.67,$$

which is closer to Lucy’s rating than to Diane’s.

Equation (4.8) also has an important flaw: it does not consider the fact that users may use different rating values to quantify the same level of appreciation for an item. For example, one user may give the highest rating value to only a few outstanding

items, while a less difficult one may give this value to most of the items he likes. This problem is usually addressed by converting the neighbors' ratings r_{vi} to normalized ones $h(r_{vi})$ [10, 60], giving the following prediction:

$$\hat{r}_{ui} = h^{-1} \left(\frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} h(r_{vi})}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|} \right). \quad (4.9)$$

Note that the predicted rating must be converted back to the original scale, hence the h^{-1} in the equation. The most common approaches to normalize ratings will be presented in Section 4.3.1.

4.2.2 User-based Classification

The prediction approach just described, where the predicted ratings are computed as a weighted average of the neighbors' ratings, essentially solves a *regression* problem. Neighborhood-based *classification*, on the other hand, finds the most likely rating given by a user u to an item i , by having the nearest-neighbors of u vote on this value. The vote v_{ir} given by the k -NN of u for the rating $r \in \mathcal{S}$ can be obtained as the sum of the similarity weights of neighbors that have given this rating to i :

$$v_{ir} = \sum_{v \in \mathcal{N}_i(u)} \delta(r_{vi} = r) w_{uv}, \quad (4.10)$$

where $\delta(r_{vi} = r)$ is 1 if $r_{vi} = r$, and 0 otherwise. Once this has been computed for every possible rating value, the predicted rating is simply the value r for which v_{ir} is the greatest.

Example 4.3. Suppose once again that the two nearest-neighbors of Eric are Lucy and Diane with respective similarity weights 0.75 and 0.15. In this case, ratings 5 and 3 each have one vote. However, since Lucy's vote has a greater weight than Diane's, the predicted rating will be $\hat{r} = 5$.

A classification method that considers normalized ratings can also be defined. Let \mathcal{S}' be the set of possible normalized values (that may require discretization), the predicted rating is obtained as:

$$\hat{r}_{ui} = h^{-1} \left(\arg \max_{r \in \mathcal{S}'} \sum_{v \in \mathcal{N}_i(u)} \delta(h(r_{vi}) = r) w_{uv} \right). \quad (4.11)$$

4.2.3 Regression VS Classification

The choice between implementing a neighborhood-based regression or classification method largely depends on the system's rating scale. Thus, if the rating scale is continuous, e.g. ratings in the *Jester* joke recommender system [23] can take any value between -10 and 10 , then a regression method is more appropriate. On the contrary, if the rating scale has only a few discrete values, e.g. "good" or "bad", or if the values cannot be ordered in an obvious fashion, then a classification method might be preferable. Furthermore, since normalization tends to map ratings to a continuous scale, it may be harder to handle in a classification approach.

Another way to compare these two approaches is by considering the situation where all neighbors have the same similarity weight. As the number of neighbors used in the prediction increases, the rating r_{ui} predicted by the regression approach will tend toward the mean rating of item i . Suppose item i has only ratings at either end of the rating range, i.e. it is either loved or hated, then the regression approach will make the safe decision that the item's worth is average. This is also justified from a statistical point of view since the expected rating (estimated in this case) is the one that minimizes the RMSE. On the other hand, the classification approach will predict the rating as the most frequent one given to i . This is more risky as the item will be labeled as either "good" or "bad". However, as mentioned before, taking risk may be desirable if it leads to serendipitous recommendations.

4.2.4 Item-based Recommendation

While user-based methods rely on the opinion of like-minded users to predict a rating, item-based approaches [18, 47, 45] look at ratings given to similar items. Let us illustrate this approach with our toy example.

Example 4.4. Instead of consulting with his peers, Eric instead determines whether the movie "Titanic" is right for him by considering the movies that he has already seen. He notices that people that have rated this movie have given similar ratings to the movies "Forrest Gump" and "Wall-E". Since Eric liked these two movies he concludes that he will also like the movie "Titanic".

This idea can be formalized as follows. Denote by $\mathcal{N}_u(i)$ the items rated by user u most similar to item i . The predicted rating of u for i is obtained as a weighted average of the ratings given by u to the items of $\mathcal{N}_u(i)$:

$$\hat{r}_{ui} = \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} r_{uj}}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \quad (4.12)$$

Example 4.5. Suppose our prediction is again made using two nearest-neighbors, and that the items most similar to "Titanic" are "Forrest Gump" and "Wall-E", with

respective similarity weights 0.85 and 0.75. Since ratings of 5 and 4 were given by Eric to these two movies, the predicted rating is computed as

$$\hat{r} = \frac{0.85 \times 5 + 0.75 \times 4}{0.85 + 0.75} \simeq 4.53.$$

Again, the differences in the users' individual rating scales can be considered by normalizing ratings with a h :

$$\hat{r}_{ui} = h^{-1} \left(\frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} h(r_{uj})}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|} \right). \quad (4.13)$$

Moreover, we can also define an item-based classification approach. In this case, the items j rated by user u vote for the rating to be given to a new item i , and these votes are weighted by the similarity between i and j . The normalized version of this approach can be expressed as follows:

$$\hat{r}_{ui} = h^{-1} \left(\arg \max_{r \in \mathcal{S}'} \sum_{j \in \mathcal{N}_u(i)} \delta(h(r_{uj}) = r) w_{ij} \right). \quad (4.14)$$

4.2.5 User-based VS Item-based Recommendation

When choosing between the implementation of a user-based and an item-based neighborhood recommender system, five criteria should be considered:

- **Accuracy:** The accuracy of neighborhood recommendation methods depends mostly on the ratio between the number of users and items in the system. As will be presented in the Section 4.3.2, the similarity between two users in user-based methods, which determines the neighbors of a user, is normally obtained by comparing the ratings made by these users on the same items. Consider a system that has 10,000 ratings made by 1,000 users on 100 items, and suppose, for the purpose of this analysis, that the ratings are distributed uniformly over the items¹. Following Table 4.1, the average number of users available as potential neighbors is roughly 650. However, the average number of common ratings used to compute the similarities is only 1. On the other hand, an item-based method usually computes the similarity between two items by comparing ratings made by the same user on these items. Assuming once more a uniform distribution of ratings, we find an average number of potential neighbors of 99 and an average number of ratings used to compute the similarities of 10.

¹ The distribution of ratings in real-life data is normally skewed, i.e. most ratings are given to a small proportion of items.

In general, a small number of high-confidence neighbors is by far preferable to a large number of neighbors for which the similarity weights are not trustworthy. In cases where the number of users is much greater than the number of items, such as large commercial systems like *Amazon.com*, item-based methods can therefore produce more accurate recommendations [19, 45]. Likewise, systems that have fewer users than items, e.g., a research paper recommender with thousands of users but hundreds of thousands of articles to recommend, may benefit more from user-based neighborhood methods [25].

Table 4.1: The average number of neighbors and average number of ratings used in the computation of similarities for user-based and item-based neighborhood methods. A uniform distribution of ratings is assumed with average number of ratings per user $p = |\mathcal{R}|/|\mathcal{U}|$, and average number of ratings per item $q = |\mathcal{R}|/|\mathcal{I}|$

	Avg. neighbors	Avg. ratings
User-based	$(\mathcal{U} - 1) \left(1 - \left(\frac{ \mathcal{I} - p}{ \mathcal{I} } \right)^p \right)$	$\frac{p^2}{ \mathcal{I} }$
Item-based	$(\mathcal{I} - 1) \left(1 - \left(\frac{ \mathcal{U} - q}{ \mathcal{U} } \right)^q \right)$	$\frac{q^2}{ \mathcal{U} }$

- **Efficiency:** As shown in Table 4.2, the memory and computational efficiency of recommender systems also depends on the ratio between the number of users and items. Thus, when the number of users exceeds the number of items, as is most often the case, item-based recommendation approaches require much less memory and time to compute the similarity weights (training phase) than user-based ones, making them more scalable. However, the time complexity of the online recommendation phase, which depends only on the number of available items and the maximum number of neighbors, is the same for user-based and item-based methods.

In practice, computing the similarity weights is much less expensive than the worst-case complexity reported in Table 4.2, due to the fact that users rate only a few of the available items. Accordingly, only the non-zero similarity weights need to be stored, which is often much less than the number of user pairs. This number can be further reduced by storing for each user only the top N weights, where N is a parameter [45]. In the same manner, the non-zero weights can be computed efficiently without having to test each pair of users or items, which makes neighborhood methods scalable to very large systems.

- **Stability:** The choice between a user-based and an item-based approach also depends on the frequency and amount of change in the users and items of the system. If the list of available items is fairly static in comparison to the users of the system, an item-based method may be preferable since the item similarity weights could then be computed at infrequent time intervals while still being able to recommend items to new users. On the contrary, in applications where the list

Table 4.2: The space and time complexity of user-based and item-based neighborhood methods, as a function of the maximum number of ratings per user $p = \max_u |\mathcal{I}_u|$, the maximum number of ratings per item $q = \max_i |\mathcal{U}_i|$, and the maximum number of neighbors used in the rating predictions k .

	Space	Time	
		Training	Online
User-based	$O(\mathcal{U} ^2)$	$O(\mathcal{U} ^2 p)$	$O(\mathcal{I} k)$
Item-based	$O(\mathcal{I} ^2)$	$O(\mathcal{I} ^2 q)$	$O(\mathcal{I} k)$

of available items is constantly changing, e.g., an online article recommender, user-based methods could prove to be more stable.

- **Justifiability:** An advantage of item-based methods is that they can easily be used to justify a recommendation. Hence, the list of neighbor items used in the prediction, as well as their similarity weights, can be presented to the user as an explanation of the recommendation. By modifying the list of neighbors and/or their weights, it then becomes possible for the user to participate interactively in the recommendation process. User-based methods, however, are less amenable to this process because the active user does not know the other users serving as neighbors in the recommendation.
- **Serendipity:** In item-based methods, the rating predicted for an item is based on the ratings given to similar items. Consequently, recommender systems using this approach will tend to recommend to a user items that are related to those usually appreciated by this user. For instance, in a movie recommendation application, movies having the same genre, actors or director as those highly rated by the user are likely to be recommended. While this may lead to safe recommendations, it does less to help the user discover different types of items that he might like as much.

Because they work with user similarity, on the other hand, user-based approaches are more likely to make serendipitous recommendations. This is particularly true if the recommendation is made with a small number of nearest-neighbors. For example, a user A that has watched only comedies may be very similar to a user B only by the ratings made on such movies. However, if B is fond of a movie in a different genre, this movie may be recommended to A through his similarity with B .

4.3 Components of Neighborhood Methods

In the previous section, we have seen that deciding between a regression and a classification rating prediction method, as well as choosing between a user-based or item-based recommendation approach, can have a significant impact on the accu-

racy, efficiency and overall quality of the recommender system. In addition to these crucial attributes, three very important considerations in the implementation of a neighborhood-based recommender system are 1) the normalization of ratings, 2) the computation of the similarity weights, and 3) the selection of neighbors. This section reviews some of the most common approaches for these three components, describes the main advantages and disadvantages of using each one of them, and gives indications on how to implement them.

4.3.1 Rating Normalization

When it comes to assigning a rating to an item, each user has its own personal scale. Even if an explicit definition of each of the possible ratings is supplied (e.g., 1=“strongly disagree”, 2=“disagree”, 3=“neutral”, etc.), some users might be reluctant to give high/low scores to items they liked/disliked. Two of the most popular rating normalization schemes that have been proposed to convert individual ratings to a more universal scale are *mean-centering* and *Z-score*.

4.3.1.1 Mean-centering

The idea of mean-centering [10, 60] is to determine whether a rating is positive or negative by comparing it to the mean rating. In user-based recommendation, a raw rating r_{ui} is transformed to a mean-centered one $h(r_{ui})$ by subtracting to r_{ui} the average \bar{r}_u of the ratings given by user u to the items in \mathcal{I}_u :

$$h(r_{ui}) = r_{ui} - \bar{r}_u.$$

Using this approach the user-based prediction of a rating r_{ui} is obtained as

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v)}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \quad (4.15)$$

In the same way, the *item*-mean-centered normalization of r_{ui} is given by

$$h(r_{ui}) = r_{ui} - \bar{r}_i,$$

where \bar{r}_i corresponds to the mean rating given to item i by user in \mathcal{U}_i . This normalization technique is most often used in item-based recommendation, where a rating r_{ui} is predicted as:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} (r_{uj} - \bar{r}_j)}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \quad (4.16)$$

An interesting property of mean-centering is that one can see right-away if the appreciation of a user for an item is positive or negative by looking at the sign of the normalized rating. Moreover, the module of this rating gives the level at which the user likes or dislikes the item.

Example 4.6. As shown in Figure 4.2, although Diane gave an average rating of 3 to the movies “Titanic” and “Forrest Gump”, the user-mean-centered ratings show that her appreciation of these movies is in fact negative. This is because her ratings are high on average, and so, an average rating correspond to a low degree of appreciation. Differences are also visible while comparing the two types of mean-centering. For instance, the item-mean-centered rating of the movie “Titanic” is neutral, instead of negative, due to the fact that much lower ratings were given to that movie. Likewise, Diane’s appreciation for “The Matrix” and John’s distaste for “Forrest Gump” are more pronounced in the item-mean-centered ratings.

User mean-centering:

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	2.50	-1.50		-0.50	-0.50
Lucy	-2.60	1.40	-1.60	1.40	1.40
Eric	-1.50		-0.50	1.50	0.50
Diane	0.25	-0.75	1.25	-0.75	

Item mean-centering:

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
John	2.00	-2.00		-1.75	-1.67
Lucy	-2.00	2.00	-1.33	1.25	1.33
Eric	-1.00		-0.33	1.25	0.33
Diane	1.00	0.00	1.67	-0.75	

Fig. 4.2: The *user* and *item* mean-centered ratings of Figure 4.1.

4.3.1.2 Z-score normalization

Consider, two users A and B that both have an average rating of 3. Moreover, suppose that the ratings of A alternate between 1 and 5, while those of B are always 3. A rating of 5 given to an item by B is more exceptional than the same rating given by A , and, thus, reflects a greater appreciation for this item. While mean-centering removes the offsets caused by the different perceptions of an average rating, Z-score normalization [29] also considers the spread in the individual rating scales.

Once again, this is usually done differently in user-based than in item-based recommendation. In user-based methods, the normalization of a rating r_{ui} divides the *user*-mean-centered rating by the standard deviation σ_u of the ratings given by user u :

$$h(r_{ui}) = \frac{r_{ui} - \bar{r}_u}{\sigma_u}.$$

A user-based prediction of rating r_{ui} using this normalization approach would therefore be obtained as

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v) / \sigma_v}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}. \quad (4.17)$$

Likewise, the z -score normalization of r_{ui} in item-based methods divides the *item*-mean-centered rating by the standard deviation of ratings given to item i :

$$h(r_{ui}) = \frac{r_{ui} - \bar{r}_i}{\sigma_i}.$$

The item-based prediction of rating r_{ui} would then be

$$\hat{r}_{ui} = \bar{r}_i + \sigma_i \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij} (r_{uj} - \bar{r}_j) / \sigma_j}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \quad (4.18)$$

4.3.1.3 Choosing a normalization scheme

In some cases, rating normalization can have undesirable effects. For instance, imagine the case of a user that gave only the highest ratings to the items he has purchased. Mean-centering would consider this user as “easy to please” and any rating below this highest rating (whether it is a positive or negative rating) would be considered as negative. However, it is possible that this user is in fact “hard to please” and carefully selects only items that he will like for sure. Furthermore, normalizing on a few ratings can produce unexpected results. For example, if a user has entered a single rating or a few identical ratings, his rating standard deviation will be 0, leading to undefined prediction values. Nevertheless, if the rating data is not overly sparse, normalizing ratings has been found to consistently improve the predictions [29, 33].

Comparing mean-centering with Z -score, as mentioned, the second one has the additional benefit of considering the variance in the ratings of individual users or items. This is particularly useful if the rating scale has a wide range of discrete values or if it is continuous. On the other hand, because the ratings are divided and multiplied by possibly very different standard deviation values, Z -score can be more sensitive than mean-centering and, more often, predict ratings that are outside the rating scale. Lastly, while an initial investigation found mean-centering and Z -score

to give comparable results [29], a more recent one showed Z-score to have more significant benefits [33].

Finally, if rating normalization is not possible or does not improve the results, another possible approach to remove the problems caused by the individual rating scale is *preference-based filtering*. The particularity of this approach is that it focuses on predicting the relative preferences of users instead of absolute rating values. Since the rating scale does not change the preference order for items, predicting relative preferences removes the need to normalize the ratings. More information on this approach can be found in [13, 21, 37, 36].

4.3.2 Similarity Weight Computation

The similarity weights play a double role in neighborhood-based recommendation methods: 1) they allow the selection of trusted neighbors whose ratings are used in the prediction, and 2) they provide the means to give more or less importance to these neighbors in the prediction. The computation of the similarity weights is one of the most critical aspects of building a neighborhood-based recommender system, as it can have a significant impact on both its accuracy and its performance.

4.3.2.1 Correlation-based similarity

A measure of the similarity between two objects a and b , often used in information retrieval, consists in representing these objects in the form of two vectors \mathbf{x}_a and \mathbf{x}_b and computing the *Cosine Vector* (CV) (or *Vector Space*) similarity [7, 8, 44] between these vectors:

$$\cos(\mathbf{x}_a, \mathbf{x}_b) = \frac{\mathbf{x}_a^\top \mathbf{x}_b}{\|\mathbf{x}_a\| \|\mathbf{x}_b\|}.$$

In the context of item recommendation, this measure can be employed to compute user similarities by considering a user u as a vector $\mathbf{x}_u \in \mathbb{R}^{|I|}$, where $\mathbf{x}_{ui} = r_{ui}$ if user u has rated item i , and 0 otherwise. The similarity between two users u and v would then be computed as

$$CV(u, v) = \cos(\mathbf{x}_u, \mathbf{x}_v) = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2 \sum_{j \in I_v} r_{vj}^2}}, \quad (4.19)$$

where I_{uv} once more denotes the items rated by both u and v . A problem with this measure is that it does not consider the differences in the mean and variance of the ratings made by users u and v .

A popular measure that compares ratings where the effects of mean and variance have been removed is the *Pearson Correlation* (PC) similarity:

$$PC(u, v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \bar{r}_v)^2}}. \quad (4.20)$$

Note that this is different from computing the CV similarity on the Z -score normalized ratings, since the standard deviation of the ratings is evaluated only on the common items I_{uv} , not on the entire set of items rated by u and v , i.e. \mathcal{I}_u and \mathcal{I}_v . The same idea can be used to obtain similarities between two items i and j [18, 45], this time by comparing the ratings made by users that have rated both of these items:

$$PC(i, j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_j)^2}}. \quad (4.21)$$

While the sign of a similarity weight indicates whether the correlation is direct or inverse, its magnitude (ranging from 0 to 1) represents the strength of the correlation.

Example 4.7. The similarities between the pairs of users and items of our toy example, as computed using PC similarity, are shown in Figure 4.3. We can see that Lucy’s taste in movies is very close to Eric’s (similarity of 0.922) but very different from John’s (similarity of -0.938). This means that Eric’s ratings can be trusted to predict Lucy’s, and that Lucy should discard John’s opinion on movies or consider the opposite. We also find that the people that like “The Matrix” also like “Die Hard” but hate “Wall-E”. Note that these relations were discovered without having any knowledge of the genre, director or actors of these movies.

The differences in the rating scales of individual users are often more pronounced than the differences in ratings given to individual items. Therefore, while computing the item similarities, it may be more appropriate to compare ratings that are centered on their *user* mean, instead of their *item* mean. The *Adjusted Cosine* (AC) similarity [45], is a modification of the PC item similarity which compares user-mean-centered ratings:

$$AC(i, j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_u)(r_{uj} - \bar{r}_u)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_u)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_u)^2}}.$$

In some cases, AC similarity has been found to outperform PC similarity on the prediction of ratings using an item-based method [45].

4.3.2.2 Other similarity measures

Several other measures have been proposed to compute similarities between users or items. One of them is the *Mean Squared Difference* (MSD) [70], which evaluates the

User-based Pearson correlation

	John	Lucy	Eric	Diane
John	1.000	-0.938	-0.839	0.659
Lucy	-0.938	1.000	0.922	-0.787
Eric	-0.839	0.922	1.000	-0.659
Diane	0.659	-0.787	-0.659	1.000

Item-based Pearson correlation

	The Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
Matrix	1.000	-0.943	0.882	-0.974	-0.977
Titanic	-0.943	1.000	-0.625	0.931	0.994
Die Hard	0.882	-0.625	1.000	-0.804	-1.000
Forrest Gump	-0.974	0.931	-0.804	1.000	0.930
Wall-E	-0.977	0.994	-1.000	0.930	1.000

Fig. 4.3: The *user* and *item* PC similarity for the ratings of Figure 4.1.

similarity between two users u and v as the inverse of the average squared difference between the ratings given by u and v on the same items:

$$\text{MSD}(u, v) = \frac{|\mathcal{I}_{uv}|}{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - r_{vi})^2}. \quad (4.22)$$

While it could be modified to compute the differences on normalized ratings, the MSD similarity is limited compared to PC similarity because it does not capture negative correlations between user preferences or the appreciation of different items. Having such negative correlations may improve the rating prediction accuracy [28].

Another well-known similarity measure is the *Spearman Rank Correlation* (SRC) [39]. While PC uses the rating values directly, SRC instead considers the ranking of these ratings. Denote by k_{ui} the rating rank of item i in user u 's list of rated items (tied ratings get the average rank of their spot). The SRC similarity between two users u and v is evaluated as:

$$\text{SRC}(u, v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (k_{ui} - \bar{k}_u)(k_{vi} - \bar{k}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (k_{ui} - \bar{k}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (k_{vi} - \bar{k}_v)^2}}, \quad (4.23)$$

where \bar{k}_u is the average rank of items rated by u (which can differ from $|\mathcal{I}_u| + 1$ if there are tied ratings).

The principal advantage of SRC is that it avoids the problem of rating normalization, described in the last section, by using rankings. On the other hand, this measure may not be the best one when the rating range has only a few possible values, since

that would create a large number of tied ratings. Moreover, this measure is typically more expensive than PC as ratings need to be sorted in order to compute their rank.

Table 4.3 shows the user-based prediction accuracy (MAE) obtained with MSD, SRC and PC similarity measures, on the *MovieLens*² dataset [28]. Results are given for different values of k , which represents the maximum number of neighbors used in the predictions. For this data, we notice that MSD leads to the least accurate predictions, possibly due to the fact that it does not take into account negative correlations. Also, these results show PC to be slightly more accurate than SRC. Finally, although PC has been generally recognized as the best similarity measure, see e.g. [28], a more recent investigation has shown that the performance of such measures depended greatly on the data [33].

Table 4.3: The rating prediction accuracy (MAE) obtained using the Mean Squared Difference (MSD), the Spearman Rank Correlation and the Pearson Correlation (PC) similarity. Results are shown for predictions using an increasing number of neighbors k .

k	MSD	SRC	PC
5	0.7898	0.7855	0.7829
10	0.7718	0.7636	0.7618
20	0.7634	0.7558	0.7545
60	0.7602	0.7529	0.7518
80	0.7605	0.7531	0.7523
100	0.7610	0.7533	0.7528

4.3.2.3 Accounting for significance

Because the rating data is frequently sparse in comparison to the number of users and items of a system, similarity weights are often computed using only a few ratings given to common items or made by the same users. For example, if the system has 10,000 ratings made by 1,000 users on 100 items (assuming a uniform distribution of ratings), Table 4.1 shows us that the similarity between two users is computed, on average, by comparing the ratings given by these users to a *single* item. If these few ratings are equal, then the users will be considered as “fully similar” and will likely play an important role in each other’s recommendations. However, if the users’ preferences are in fact different, this may lead to poor recommendations.

Several strategies have been proposed to take into account the *significance* of a similarity weight. The principle of these strategies is essentially the same: reduce the magnitude of a similarity weight when this weight is computed using only a few ratings. For instance, in *Significance Weighting* [29, 49], a user similarity weight

² <http://www.grouplens.org/>

w_{uv} is penalized by a factor proportional to the number of commonly rated items, if this number is less than a given parameter $\gamma > 0$:

$$w'_{uv} = \frac{\min\{|\mathcal{I}_{uv}|, \gamma\}}{\gamma} \times w_{uv}. \quad (4.24)$$

Likewise, an item similarity w_{ij} , obtained from a few ratings, can be adjusted as

$$w'_{ij} = \frac{\min\{|\mathcal{U}_{ij}|, \gamma\}}{\gamma} \times w_{ij}. \quad (4.25)$$

In [29, 28], it was found that using $\gamma \geq 25$ could significantly improve the accuracy of the predicted ratings, and that a value of 50 for γ gave the best results. However, the optimal value for this parameter is data dependent and should be determined using a cross-validation approach.

A characteristic of significance weighting is its use of a threshold γ determining when a weight should be adjusted. A more continuous approach, described in [4], is based on the concept of *shrinkage* where a weak or biased estimator can be improved if it is “shrunk” toward a null-value. This approach can be justified using a Bayesian perspective, where the best estimator of a parameter is the posterior mean, corresponding to a linear combination of the prior mean of the parameter (null-value) and an empirical estimator based fully on the data. In this case, the parameters to estimate are the similarity weights and the null value is zero. Thus, a user similarity w_{uv} estimated on a few ratings is shrunk as

$$w'_{uv} = \frac{|\mathcal{I}_{uv}|}{|\mathcal{I}_{uv}| + \beta} \times w_{uv}, \quad (4.26)$$

where $\beta > 0$ is a parameter whose value should also be selected using cross-validation. In this approach, w_{uv} is shrunk proportionally to $\beta/|\mathcal{I}_{uv}|$, such that almost no adjustment is made when $|\mathcal{I}_{uv}| \gg \beta$. Item similarities can be shrunk in the same way:

$$w'_{ij} = \frac{|\mathcal{U}_{ij}|}{|\mathcal{U}_{ij}| + \beta} \times w_{ij}, \quad (4.27)$$

As reported in [4], a typical value for β is 100.

4.3.2.4 Accounting for variance

Ratings made by two users on universally liked/disliked items may not be as informative as those made for items with a greater rating variance. For instance, most people like classic movies such as “The Godfather”, so basing the weight computation on such movies would produce artificially high values. Likewise, a user that always rates items in the same way may provide less predictive information than one whose preferences vary from one item to another.

A recommendation approach that addresses this problem is the *Inverse User Frequency* [10]. Based on the information retrieval notion of *Inverse Document Frequency* (IDF), a weight λ_i is given to each item i , in proportion to the log-ratio of users that have rated i :

$$\lambda_i = \log \frac{|\mathcal{U}|}{|\mathcal{U}_i|}.$$

While computing the *Frequency-Weighted Pearson Correlation* (FWPC) between users u and v , the correlation between the ratings given to an item i is weighted by λ_i :

$$\text{FWPC}(u, v) = \frac{\sum_{i \in \mathcal{I}_{uv}} \lambda_i (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} \lambda_i (r_{ui} - \bar{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} \lambda_i (r_{vi} - \bar{r}_v)^2}}. \quad (4.28)$$

This approach, which was found to improve the prediction accuracy of a user-based recommendation method [10], could also be adapted to the computation of item similarities.

More advanced strategies have also been proposed to consider rating variance. One of these strategies, described in [35], computes the factors λ_i by maximizing the average similarity between users. In this approach, the similarity between two users u and v , given an item weight vector $\lambda = (\lambda_1, \dots, \lambda_{|\mathcal{I}|})$, is evaluated as the likelihood of u to have the same rating behavior as user v :

$$\Pr(u|v, \lambda) = \frac{1}{Z_v} \exp \left(\sum_{i \in \mathcal{I}_{uv}} \lambda_i r_{ui} r_{vi} \right),$$

where Z_v is a normalization constant. The optimal item weight vector is the one maximizing the average similarity between users.

4.3.3 Neighborhood Selection

The number of nearest-neighbors to select and the criteria used for this selection can also have a serious impact on the quality of the recommender system. The selection of the neighbors used in the recommendation of items is normally done in two steps: 1) a global filtering step where only the most likely candidates are kept, and 2) a per prediction step which chooses the best candidates for this prediction.

4.3.3.1 Pre-filtering of neighbors

In large recommender systems that can have millions of users and items, it is usually not possible to store the (non-zero) similarities between each pair of users or items, due to memory limitations. Moreover, doing so would be extremely wasteful as only the most significant of these values are used in the predictions. The pre-filtering of

neighbors is an essential step that makes neighborhood-based approaches practicable by reducing the amount of similarity weights to store, and limiting the number of candidate neighbors to consider in the predictions. There are several ways in which this can be accomplished:

- **Top- N filtering:** For each user or item, only a list of the N nearest-neighbors and their respective similarity weight is kept. To avoid problems with efficiency or accuracy, N should be chosen carefully. Thus, if N is too large, an excessive amount of memory will be required to store the neighborhood lists and predicting ratings will be slow. On the other hand, selecting a too small value for N may reduce the coverage of the recommendation method, which causes some items to be never recommended.
- **Threshold filtering:** Instead of keeping a fixed number of nearest-neighbors, this approach keeps all the neighbors whose similarity weight has a magnitude greater than a given threshold w_{\min} . While this is more flexible than the previous filtering technique, as only the most significant neighbors are kept, the right value of w_{\min} may be difficult to determine.
- **Negative filtering:** In general, negative rating correlations are less reliable than positive ones. Intuitively, this is because strong positive correlation between two users is a good indicator of their belonging to a common group (e.g., teenagers, science-fiction fans, etc.). However, although negative correlation may indicate membership to different groups, it does not tell how different these groups are, or whether these groups are compatible for other categories of items. While experimental investigations [29, 25] have found negative correlations to provide no significant improvement in the prediction accuracy, whether such correlations can be discarded depends on the data.

Note that these three filtering approaches are not mutually exclusive and can be combined to fit the needs of the recommender system. For instance, one could discard all negative similarities *as well as* those with a magnitude lower than a given threshold.

4.3.3.2 Neighbors in the predictions

Once a list of candidate neighbors has been computed for each user or item, the prediction of new ratings is normally made with the k -nearest-neighbors, that is, the k neighbors whose similarity weight has the greatest magnitude. The important question is which value to use for k .

As shown in Table 4.3, the prediction accuracy observed for increasing values of k typically follows a *concave* function. Thus, when the number of neighbors is restricted by using a small k (e.g., $k < 20$), the prediction accuracy is normally low. As k increases, more neighbors contribute to the prediction and the variance introduced by individual neighbors is averaged out. As a result, the prediction accuracy improves. Finally, the accuracy usually drops when too many neighbors are used in

the prediction (e.g., $k > 50$), due to the fact that the few strong local relations are “diluted” by the many weak ones. Although a number of neighbors between 20 to 50 is most often described in the literature, see e.g. [28, 25], the optimal value of k should be determined by cross-validation.

On a final note, more serendipitous recommendations may be obtained at the cost of a decrease in accuracy, by basing these recommendations on a few very similar users. For example, the system could find the user most similar to the active one and recommend the new item that has received the highest rated from this user.

4.4 Advanced Techniques

The neighborhood approaches based on rating correlation, such as the ones presented in the previous sections, have two important flaws:

- **Limited coverage:** Because rating correlation measures the similarity between two users by comparing their ratings for the same items, users can be neighbors *only if* they have rated common items. This assumption is very limiting, as users having rated a few or no common items may still have similar preferences. Moreover, since only items rated by neighbors can be recommended, the coverage of such methods can also be limited.
- **Sensitivity to sparse data:** Another consequence of rating correlation, addressed briefly in Section 4.2.5, is the fact that the accuracy of neighborhood-based recommendation methods suffers from the lack of available ratings. Sparsity is a problem common to most recommender systems due to the fact that users typically rate only a small proportion of the available items [7, 25, 68, 67]. This is aggravated by the fact that users or items newly added to the system may have no ratings at all, a problem known as *cold-start* [69]. When the rating data is sparse, two users or items are unlikely to have common ratings, and consequently, neighborhood-based approaches will predict ratings using a very limited number of neighbors. Moreover, similarity weights may be computed using only a small number of ratings, resulting in biased recommendations (see Section 4.3.2.3 for this problem).

A common solution for these problems is to fill the missing ratings with default values [10, 18], such as the middle value of the rating range, and the average user or item rating. A more reliable approach is to use content information to fill out the missing ratings [16, 25, 41, 50]. For instance, the missing ratings can be provided by autonomous agents called *filterbots* [25, 41], that act as ordinary users of the system and rate items based on some specific characteristics of their content. The missing ratings can instead be predicted by a content-based approach [50], such as those described in Section 4.1.2.1. Finally, content similarity can also be used “instead of” or “in addition to” rating correlation similarity to find the nearest-neighbors employed in the predictions [7, 46, 59, 72].

These solutions, however, also have their own drawbacks. For instance, giving a default value to missing ratings may induce bias in the recommendations. Also, as discussed in Section 4.1.2.1, item content may not be available to compute ratings or similarities. This section presents two approaches proposed for the problems of limited coverage and sparsity: *dimensionality reduction* and *graph-based* methods.

4.4.1 Dimensionality Reduction Methods

Dimensionality reduction methods [4, 7, 23, 42, 67, 74, 75] address the problems of limited coverage and sparsity by projecting users and items into a reduced latent space that captures their most salient features. Because users and items are compared in this dense subspace of high-level features, instead of the “rating space”, more meaningful relations can be discovered. In particular, a relation between two users can be found, even though these users have rated different items. As a result, such methods are generally less sensitive to sparse data [4, 7, 67].

There are essentially two ways in which dimensionality reduction can be used to improve recommender systems: 1) decomposition of a user-item *rating* matrix, and 2) decomposition of a sparse *similarity* matrix.

4.4.1.1 Decomposing the rating matrix

A popular dimensionality reduction approach to item recommendation is *Latent Semantic Indexing* (LSI) [15]. In this approach, the $|\mathcal{U}| \times |\mathcal{I}|$ user-item rating matrix R of rank n is approximated by a matrix $\hat{R} = PQ^\top$ of rank $k < n$, where P is a $|\mathcal{U}| \times k$ matrix of *users* factors and Q a $|\mathcal{I}| \times k$ matrix of *item* factors. Intuitively, the u -th row of P , $\mathbf{p}_u \in \mathbb{R}^k$, represents the coordinates of user u projected in the k -dimensional latent space. Likewise, the i -th row of Q , $\mathbf{q}_i \in \mathbb{R}^k$, can be seen as the coordinates of item i in this latent space. Matrices P and Q are normally found by minimizing the reconstruction error defined with the squared Frobenius norm:

$$\begin{aligned} \text{err}(P, Q) &= \|R - PQ^\top\|_F^2 \\ &= \sum_{u,i} (r_{ui} - \mathbf{p}_u \mathbf{q}_i^\top)^2. \end{aligned}$$

Minimizing this error is equivalent to finding the *Singular Value Decomposition* (SVD) of R [24]:

$$R = U\Sigma V^\top,$$

where U is the $|\mathcal{U}| \times n$ matrix of left singular vectors, V is the $|\mathcal{I}| \times n$ matrix of right singular vectors, and Σ is the $n \times n$ diagonal matrix of singular values. Denote by Σ_k , U_k and V_k the matrices obtained by selecting the subset containing the k highest

singular values and their corresponding singular vectors, the user and item factor matrices correspond to $P = U_k \Sigma_k^{1/2}$ and $Q = V_k \Sigma_k^{1/2}$.

Once P and Q have been obtained, the typical *model-based* prediction of a rating r_{ui} is:

$$r_{ui} = \mathbf{p}_u \mathbf{q}_i^\top.$$

There is, however, a major problem with applying SVD to the rating matrix R : most values r_{ui} of R are undefined, since there may not be a rating given to i by u . Although it is possible to assign a default value to r_{ui} , as mentioned above, this would introduce a bias in the data. More importantly, this would make the large matrix R dense and, consequently, render impractical the SVD decomposition of R . The common solution to this problem is to learn P and Q using only the known ratings [4, 42, 73, 75]:

$$\text{err}(P, Q) = \sum_{r_{ui} \in \mathcal{R}} (r_{ui} - \mathbf{p}_u \mathbf{q}_i^\top)^2 + \lambda (\|\mathbf{p}_u\|^2 + \|\mathbf{q}_i\|^2), \quad (4.29)$$

where λ is a parameter that controls the level of regularization. A more comprehensive description of this recommendation approach can be found in Chapter 5 of this book.

In neighborhood-based recommendation, the same principle can be used to compute the similarity between users or items in the latent-space [7]. This can be done by solving the following problem:

$$\text{err}(P, Q) = \sum_{r_{ui} \in \mathcal{R}} (z_{ui} - \mathbf{p}_u \mathbf{q}_i^\top)^2 \quad (4.30)$$

subject to:

$$\|\mathbf{p}_u\| = 1, \forall u \in \mathcal{U}, \quad \|\mathbf{q}_i\| = 1, \forall i \in \mathcal{I},$$

where z_{ui} is the mean-centered rating r_{ui} normalized to the $[-1, 1]$ range. For example, if r_{\min} and r_{\max} are the lowest and highest values in the original rating range,

$$z_{ui} = \frac{r_{ui} - \bar{r}_u}{r_{\max} - r_{\min}}.$$

This problem corresponds to finding, for each user u and item i , coordinates on the surface of the k -dimensional unit sphere such that u will give a high rating to i if their coordinates are close together on the surface. If two users u and v are nearby on the surface, then they will give similar ratings to the same items, and, thus, the similarity between these users can be computed as

$$w_{uv} = \mathbf{p}_u \mathbf{p}_v^\top.$$

Likewise, the similarity between two items i and j can be obtained as

$$w_{ij} = \mathbf{q}_i \mathbf{q}_j^\top.$$

4.4.1.2 Decomposing the similarity matrix

The principle of this second dimensionality reduction approach is the same as the previous one: decompose a matrix into its principal factors representing projection of users or items in the latent space. However, instead of decomposing the rating matrix, a sparse similarity matrix is decomposed. Let W be a symmetric matrix of rank n representing either user or item similarities. To simplify the presentation, we will suppose the former case. Once again, we want to approximate W with a matrix $\hat{W} = PP^\top$ of lower rank $k < n$ by minimizing the following objective:

$$\begin{aligned} \text{err}(P) &= \|R - PP^\top\|_F^2 \\ &= \sum_{u,v} \left(w_{uv} - \mathbf{p}_u \mathbf{p}_v^\top \right)^2. \end{aligned}$$

Matrix \hat{W} can be seen as a “compressed” version of W which is less sparse than W . As before, finding the factor matrix P is equivalent to computing the eigenvalue decomposition of W :

$$W = V\Lambda V^\top,$$

where Λ is a diagonal matrix containing the $|\mathcal{U}|$ eigenvalues of W , and V is a $|\mathcal{U}| \times |\mathcal{U}|$ orthogonal matrix containing the corresponding eigenvectors. Let V_k be a matrix formed by the k principal (normalized) eigenvectors of W , which correspond to the axes of the k -dimensional latent subspace. The coordinates $\mathbf{p}_u \in \mathbb{R}^k$ of a user u in this subspace is given by the u -th row of matrix $P = V_k \Lambda_k^{1/2}$. Furthermore, the user similarities computed in this latent subspace are given by matrix

$$\begin{aligned} W' &= PP^\top \\ &= V_k \Lambda_k V_k^\top. \end{aligned} \tag{4.31}$$

This approach was used to recommend jokes in the Eigentaste system [23]. In Eigentaste, a matrix W containing the PC similarities between pairs of items is decomposed to obtain the latent subspace defined by the two principal eigenvectors of W . Denote V_2 the matrix containing these eigenvectors. A user u , represented by the u -th row \mathbf{r}_u of the rating matrix R , is projected in the plane defined by V_2 :

$$\mathbf{r}'_u = \mathbf{r}_u V_2.$$

In an offline step, the users of the system are clustered in the plane using a recursive subdivision technique. Then, the rating of user u for an item i is evaluated as the mean rating for i made by users in the same cluster as u .

4.4.2 Graph-based Methods

In graph-based approaches, the data is represented in the form of a graph where nodes are users, items or both, and edges encode the interactions or similarities between the users and items. For example, in Figure 4.4, the data is modeled as a bipartite graph where the two sets of nodes represent users and items, and an edge connects user u to item i if there is a rating given to i by u in the system. A weight can also be given to this edge, such as the value of its corresponding rating. In another model, the nodes can represent either users or items, and an edge connects two nodes if the ratings corresponding two these nodes are sufficiently correlated. The weight of this edge can be the corresponding correlation value.

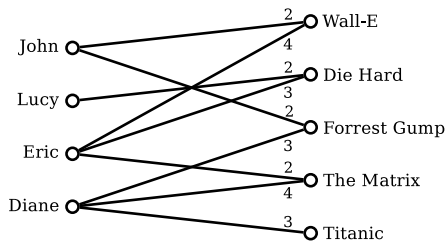


Fig. 4.4: A bipartite graph representation of the ratings of Figure 4.1 (only ratings with value in $\{2, 3, 4\}$ are shown).

In these models, standard approaches based on correlation predict the rating of a user u for an item i using only the nodes directly connected to u or i . Graph-based approaches, on the other hand, allow nodes that are not directly connected to influence each other by propagating information along the edges of the graph. The greater the weight of an edge, the more information is allowed to pass through it. Also, the influence of a node on another should be smaller if the two nodes are further away in the graph. These two properties, known as *propagation* and *attenuation* [26, 34], are often observed in graph-based similarity measures.

The transitive associations captured by graph-based methods can be used to recommend items in two different ways. In the first approach, the proximity of a user u to an item i in the graph is used directly to evaluate the rating of u for i [19, 26, 34]. Following this idea, the items recommended to u by the system are those that are the “closest” to u in the graph. On the other hand, the second approach considers the proximity of two users or item nodes in the graph as a measure of similarity, and uses this similarity as the weights w_{uv} or w_{ij} of a neighborhood-based recommendation method [19, 48].

4.4.2.1 Path-based similarity

In path-based similarity, the distance between two nodes of the graph is evaluated as a function of the number of paths connecting the two nodes, as well as the length of these paths.

Shortest path

A recommendation approach that computes the similarity between two users based on their shortest distance in a graph is the one described in [2]. In this method, the data is modeled as a directed graph whose nodes are users, and in which edges are determined based on the notions of *horting* and *predictability*. Horting is an asymmetric relation between two users that is satisfied if these users have rated similar items. Formally, a user u horts another user v provided either $|\mathcal{I}_{uv}| \geq \alpha$ or $|\mathcal{I}_{uv}|/|\mathcal{I}_u| \geq \beta$ is satisfied, where α, β are predetermined thresholds. Predictability, on the other hand, is a stronger property additionally requiring the ratings of u to be similar to those of v , under a mapping representing the difference in the rating scales of u and v . Thus, v predicts u , provided u horts v and there exists a linear transformation $l : \mathcal{S} \rightarrow \mathcal{S}$ such that

$$\frac{1}{|\mathcal{I}_{uv}|} \sum_{i \in \mathcal{I}_{uv}} |r_{ui} - l(r_{vi})| \leq \gamma,$$

where γ is another given threshold.

The relations of predictability are represented as directed edges in the graph, such that there is a directed edge from u to v if v predicts u . Accordingly, a directed path connecting two users u and v represents the transitive predictability of v for the ratings of u , under a sequence of transformations. Following this idea, the rating of user u for a new item i is predicted using the shortest directed paths from u to other users that have rated i . Let $P = \{u, v_1, v_2, \dots, v_m\}$ be such a path, where $v_m \in \mathcal{U}_i$. The rating of user v_m for item i is transformed in the rating scale of u using the composition of the linear mappings along the path:

$$\hat{r}_{ui}^{(P)} = (l_m \circ \dots \circ l_2 \circ l_1)(r_{vi}).$$

The final prediction of rating r_{ui} is computed as the average of the predictions $\hat{r}_{ui}^{(P)}$ obtained for all shortest paths P .

Number of paths

The number of paths between a user and an item in a bipartite graph can also be used to evaluate their compatibility [34]. Let R be once again the $|U| \times |I|$ rating matrix where r_{ui} equals 1 if user u has rated item i , and 0 otherwise. The adjacency matrix

A of the bipartite graph can be defined from R as

$$A = \begin{pmatrix} 0 & R^T \\ R & 0 \end{pmatrix}.$$

In this approach, the association between a user u and an item i is defined as the sum of the weights of all distinctive paths connecting u to v (allowing nodes to appear more than once in the path), whose length is no more than a given maximum length K . Note that, since the graph is bipartite, K should be an odd number. In order to attenuate the contribution of longer paths, the weight given to a path of length k is defined as α^k , where $\alpha \in [0, 1]$. Using the fact that the number of k length paths between pairs of nodes is given by A^k , the user-item association matrix S_K is

$$\begin{aligned} S_K &= \sum_{k=1}^K \alpha^k A^k \\ &= (I - \alpha A)^{-1} (\alpha A - \alpha^K A^K). \end{aligned} \quad (4.32)$$

This method of computing distances between nodes in a graph is known as the *Katz* measure [38]. Note that this measure is closely related to the *Von Neumann Diffusion* kernel [20, 40, 43]

$$\begin{aligned} K_{\text{VND}} &= \sum_{k=0}^{\infty} \alpha^k A^k \\ &= (I - \alpha A)^{-1} \end{aligned} \quad (4.33)$$

and the *Exponential Diffusion* kernel

$$\begin{aligned} K_{\text{ED}} &= \sum_{k=0}^{\infty} \frac{1}{k!} \alpha^k A^k \\ &= \exp(\alpha A), \end{aligned} \quad (4.34)$$

where $A^0 = I$.

In recommender systems that have a large number of users and items, computing these association values may require extensive computational resources. To overcome these limitations, spreading activation techniques [14] have been used in [34]. Essentially, such techniques work by first activating a selected subset of nodes as starting nodes, and then iteratively activating the nodes that can be reached directly from the nodes that are already active, until a convergence criterion is met.

4.4.2.2 Random walk similarity

Transitive associations in graph-based methods can also be defined within a probabilistic framework. In this framework, the similarity or affinity between users or items is evaluated as a probability of reaching these nodes in a random walk. For-

mally, this can be described with a first-order Markov process defined by a set of n states and a $n \times n$ transition probability matrix P such that the probability of jumping from state i to j at any time-step t is

$$p_{ij} = \Pr(s(t+1) = j | s(t) = i).$$

Denote $\pi(t)$ the vector containing the state probability distribution of step t , such that $\pi_i(t) = \Pr(s(t) = i)$, the evolution of the Markov chain is characterized by

$$\pi(t+1) = P^\top \pi(t).$$

Moreover, under the condition that P is row-stochastic, i.e. $\sum_j p_{ij} = 1$ for all i , the process converges to a stable distribution vector $\pi(\infty)$ corresponding to the positive eigenvector of P^\top with an eigenvalue of 1. This process is often described in the form of a weighted graph having a node for each state, and where the probability of jumping from a node to an adjacent node is given by the weight of the edge connecting these nodes.

Itemrank

A recommendation approach, based on the PageRank algorithm for ranking Web pages [11], is ItemRank [26]. This approach ranks the preferences of a user u for new items i as the probability of u to visit i in a random walk of a graph in which nodes correspond to the items of the system, and edges connect items that have been rated by common users. The edge weights are given by the $|\mathcal{I}| \times |\mathcal{I}|$ transition probability matrix P for which $p_{ij} = |\mathcal{U}_{ij}|/|\mathcal{U}_i|$ is the estimated conditional probability of a user to rate an item j if it has rated an item i .

As in PageRank, the random walk can, at any step t , either jump using P to an adjacent node with fixed probability α , or “teleport” to any node with probability $(1 - \alpha)$. Let \mathbf{r}_u be the u -th row of the rating matrix R , the probability distribution of user u to teleport to other nodes is given by vector $\mathbf{d}_u = \mathbf{r}_u / \|\mathbf{r}_u\|$. Following these definitions, the state probability distribution vector of user u at step $t+1$ can be expressed recursively as

$$\pi_u(t+1) = \alpha P^\top \pi_u(t) + (1 - \alpha) \mathbf{d}_u. \quad (4.35)$$

For practical reasons, $\pi_u(\infty)$ is usually obtained with a procedure that first initializes the distribution as uniform, i.e. $\pi_u(0) = \frac{1}{n} \mathbf{1}_n$, and then iteratively updates π_u , using (4.35), until convergence. Once $\pi_u(\infty)$ has been computed, the system recommends to u the item i for which π_{ui} is the highest.

Average first-passage/commute time

Other distance measures based on random walks have been proposed for the recommendation problem. Among these are the *average first-passage time* and the *average commute time* [19, 20]. The average first-passage time $m(j|i)$ [56] is the average number of steps needed by a random walker to reach a node j for the first time, when starting from a node $i \neq j$. Let P be the $n \times n$ transition probability matrix, $m(j|i)$ can be expressed recursively as

$$m(j|i) = \begin{cases} 0 & , \text{ if } i = j \\ 1 + \sum_{k=1}^n p_{ik} m(j|k) & , \text{ otherwise} \end{cases}$$

A problem with the average first-passage time is that it is not symmetric. A related measure that does not have this problem is the average commute time $n(i, j) = m(j|i) + m(i|j)$ [22], corresponding to the average number of steps required by a random walker starting at node $i \neq j$ to reach node j for the first time and go back to i . This measure has several interesting properties. Namely, it is a true distance measure in some Euclidean space [22], and is closely related to the well-known property of resistance in electrical networks and to the pseudo-inverse of the graph Laplacian matrix [19].

In [19], the average commute time is used to compute the distance between the nodes of a bipartite graph representing the interactions of users and items in a recommender system. For each user u there is a directed edge from u to every item $i \in \mathcal{I}_u$, and the weight of this edge is simply $1/|\mathcal{I}_u|$. Likewise, there is a directed edge from each item i to every user $u \in \mathcal{U}_i$, with weight $1/|\mathcal{U}_i|$. Average commute times can be used in two different ways: 1) recommending to u the item i for which $n(u, i)$ is the smallest, or 2) finding the users nearest to u , according to the commute time distance, and then suggest to u the item most liked by these users.

4.5 Conclusion

One of the earliest approaches proposed for the task item recommendation, neighborhood-based recommendation still ranks among the most popular methods for this problem. Although quite simple to describe and implement, this recommendation approach has several important advantages, including its ability to explain a recommendation with the list of the neighbors used, its computational and space efficiency which allows it to scale to large recommender systems, and its marked stability in an online setting where new users and items are constantly added. Another of its strengths is its potential to make serendipitous recommendations that can lead users to the discovery of unexpected, yet very interesting items.

In the implementation of a neighborhood-based approach, one has to make several important decisions. Perhaps the one having the greatest impact on the accuracy

and efficiency of the recommender system is choosing between a user-based and an item-based neighborhood method. In typical commercial recommender systems, where the number of users far exceeds the number of available items, item-based approaches are typically preferred since they provide more accurate recommendations, while being more computationally efficient and requiring less frequent updates. On the other hand, user-based methods usually provide more original recommendations, which may lead users to a more satisfying experience. Moreover, the different components of a neighborhood-based method, which include the normalization of ratings, the computation of the similarity weights and the selection of the nearest-neighbors, can also have a significant influence on the quality of the recommender system. For each of these components, several different alternatives are available. Although the merit of each of these has been described in this document and in the literature, it is important to remember that the “best” approach may differ from one recommendation setting to the next. Thus, it is important to evaluate them on data collected from the actual system, and in light of the particular needs of the application.

Finally, when the performance of a neighborhood-based approach suffers from the problems of limited coverage and sparsity, one may explore techniques based on dimensionality reduction or graphs. Dimensionality reduction provides a compact representation of users and items that captures their most significant features. An advantage of such an approach is that it can obtain meaningful relations between pairs of users or items, even though these users have rated different items, or these items were rated by different users. On the other hand, graph-based techniques exploit the transitive relations in the data. These techniques also avoid the problems of sparsity and limited coverage by evaluating the relationship between users or items that are not “directly connected”. However, unlike dimensionality reduction, graph-based methods also preserve some of the “local” relations in the data, which are useful in making serendipitous recommendations.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749 (2005)
2. Aggarwal, C.C., Wolf, J.L., Wu, K.L., Yu, P.S.: Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In: *KDD '99: Proc. of the 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 201–212. ACM, New York, NY, USA (1999)
3. Balabanović, M., Shoham, Y.: Fab: Content-based, collaborative recommendation. *Communications of the ACM* **40**(3), 66–72 (1997)
4. Bell, R., Koren, Y., Volinsky, C.: Modeling relationships at multiple scales to improve accuracy of large recommender systems. In: *KDD '07: Proc. of the 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 95–104. ACM, New York, NY, USA (2007)
5. Bell, R.M., Koren, Y.: Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In: *ICDM '07: Proc. of the 2007 Seventh IEEE Int. Conf. on Data Mining*, pp. 43–52. IEEE Computer Society, Washington, DC, USA (2007)

6. Billsus, D., Brunk, C.A., Evans, C., Gladish, B., Pazzani, M.: Adaptive interfaces for ubiquitous web access. *Communications of the ACM* **45**(5), 34–38 (2002)
7. Billsus, D., Pazzani, M.J.: Learning collaborative information filters. In: *ICML '98: Proc. of the 15th Int. Conf. on Machine Learning*, pp. 46–54. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
8. Billsus, D., Pazzani, M.J.: User modeling for adaptive news access. *User Modeling and User-Adapted Interaction* **10**(2-3), 147–180 (2000)
9. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. *Journal of Machine Learning Research* **3**, 993–1022 (2003)
10. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: *Proc. of the 14th Annual Conf. on Uncertainty in Artificial Intelligence*, pp. 43–52. Morgan Kaufmann (1998)
11. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* **30**(1-7), 107–117 (1998)
12. Buckley, C., Salton, G.: Optimization of relevance feedback weights. In: *SIGIR '95: Proc. of the 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pp. 351–357. ACM, New York, NY, USA (1995)
13. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to order things. In: *NIPS '97: Proc. of the 1997 Conf. on Advances in Neural Information Processing Systems*, pp. 451–457. MIT Press, Cambridge, MA, USA (1998)
14. Crestani, F., Lee, P.L.: Searching the Web by constrained spreading activation. *Information Processing and Management* **36**(4), 585–605 (2000)
15. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. *Journal of the American Society for Information Science* **41**, 391–407 (1990)
16. Degenmis, M., Lops, P., Semeraro, G.: A content-collaborative recommender that exploits wordnet-based user profiles for neighborhood formation. *User Modeling and User-Adapted Interaction* **17**(3), 217–255 (2007)
17. Delgado, J., Ishii, N.: Memory-based weighted majority prediction for recommender systems. In: *Proc. of the ACM SIGIR'99 Workshop on Recommender Systems* (1999)
18. Deshpande, M., Karypis, G.: Item-based top-N recommendation algorithms. *ACM Transaction on Information Systems* **22**(1), 143–177 (2004)
19. Fous, F., Renders, J.M., Pirotte, A., Saerens, M.: Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering* **19**(3), 355–369 (2007)
20. Fous, F., Yen, L., Pirotte, A., Saerens, M.: An experimental investigation of graph kernels on a collaborative recommendation task. In: *ICDM '06: Proc. of the 6th Int. Conf. on Data Mining*, pp. 863–868. IEEE Computer Society, Washington, DC, USA (2006)
21. Freund, Y., Iyer, R.D., Schapire, R.E., Singer, Y.: An efficient boosting algorithm for combining preferences. In: *ICML '98: Proc. of the 15th Int. Conf. on Machine Learning*, pp. 170–178. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
22. Gobel, F., Jagers, A.: Random walks on graphs. *Stochastic Processes and Their Applications* **2**, 311–336 (1974)
23. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval* **4**(2), 133–151 (2001)
24. Golub, G.H., Van Loan, C.F.: *Matrix computations* (3rd ed.). Johns Hopkins University Press (1996)
25. Good, N., Schafer, J.B., Konstan, J.A., Borchers, A., Sarwar, B., Herlocker, J., Riedl, J.: Combining collaborative filtering with personal agents for better recommendations. In: *AAAI '99/IAAI '99: Proc. of the 16th National Conf. on Artificial Intelligence*, pp. 439–446. American Association for Artificial Intelligence, Menlo Park, CA, USA (1999)
26. Gori, M., Pucci, A.: Itemrank: a random-walk based scoring algorithm for recommender engines. In: *Proc. of the 2007 IJCAI Conf.*, pp. 2766–2771 (2007)

27. Grcar, M., Fortuna, B., Mladenic, D., Grobelnik, M.: k-NN versus SVM in the collaborative filtering framework. *Data Science and Classification* pp. 251–260 (2006). URL <http://db.cs.ualberta.ca/webkdd05/proc/paper25-mladenic.pdf>
28. Herlocker, J., Konstan, J.A., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.* **5**(4), 287–310 (2002)
29. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: *SIGIR '99: Proc. of the 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pp. 230–237. ACM, New York, NY, USA (1999)
30. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **22**(1), 5–53 (2004)
31. Hill, W., Stead, L., Rosenstein, M., Furnas, G.: Recommending and evaluating choices in a virtual community of use. In: *CHI '95: Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pp. 194–201. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995)
32. Hofmann, T.: Collaborative filtering via Gaussian probabilistic latent semantic analysis. In: *SIGIR '03: Proc. of the 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pp. 259–266. ACM, New York, NY, USA (2003)
33. Howe, A.E., Forbes, R.D.: Re-considering neighborhood-based collaborative filtering parameters in the context of new data. In: *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pp. 1481–1482. ACM, New York, NY, USA (2008)
34. Huang, Z., Chen, H., Zeng, D.: Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems* **22**(1), 116–142 (2004)
35. Jin, R., Chai, J.Y., Si, L.: An automatic weighting scheme for collaborative filtering. In: *SIGIR '04: Proc. of the 27th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pp. 337–344. ACM, New York, NY, USA (2004)
36. Jin, R., Si, L., Zhai, C.: Preference-based graphic models for collaborative filtering. In: *Proc. of the 19th Annual Conf. on Uncertainty in Artificial Intelligence (UAI-03)*, pp. 329–33. Morgan Kaufmann, San Francisco, CA (2003)
37. Jin, R., Si, L., Zhai, C., Callan, J.: Collaborative filtering with decoupled models for preferences and ratings. In: *CIKM '03: Proc. of the 12th Int. Conf. on Information and Knowledge Management*, pp. 309–316. ACM, New York, NY, USA (2003)
38. Katz, L.: A new status index derived from sociometric analysis. *Psychometrika* **18**(1), 39–43 (1953)
39. Kendall, M., Gibbons, J.D.: *Rank Correlation Methods*, 5 edn. Charles Griffin (1990)
40. Kondor, R.I., Lafferty, J.D.: Diffusion kernels on graphs and other discrete input spaces. In: *ICML '02: Proc. of the Nineteenth Int. Conf. on Machine Learning*, pp. 315–322. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
41. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: GroupLens: applying collaborative filtering to usenet news. *Communications of the ACM* **40**(3), 77–87 (1997)
42. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: *KDD'08: Proceeding of the 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 426–434. ACM, New York, NY, USA (2008)
43. Kunegis, J., Lommatzsch, A., Bauckhage, C.: Alternative similarity functions for graph kernels. In: *Proc. of the Int. Conf. on Pattern Recognition* (2008)
44. Lang, K.: News Weeder: Learning to filter netnews. In: *Proc. of the 12th Int. Conf. on Machine Learning*, pp. 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1995)
45. Last.fm: Music recommendation service (2009). <http://www.last.fm>
46. Li, J., Zaiane, O.R.: Combining usage, content, and structure data to improve Web site recommendation. In: *Proc. of the 5th Int. Conf. on Electronic Commerce and Web Technologies (EC-Web)* (2004)
47. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing* **7**(1), 76–80 (2003)

48. Luo, H., Niu, C., Shen, R., Ullrich, C.: A collaborative filtering framework based on both local user similarity and global user similarity. *Machine Learning* **72**(3), 231–245 (2008)
49. Ma, H., King, I., Lyu, M.R.: Effective missing data prediction for collaborative filtering. In: *SIGIR '07: Proc. of the 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pp. 39–46. ACM, New York, NY, USA (2007)
50. Melville, P., Mooney, R.J., Nagarajan, R.: Content-boosted collaborative filtering for improved recommendations. In: *18th National Conf. on Artificial Intelligence*, pp. 187–192. American Association for Artificial Intelligence, Menlo Park, CA, USA (2002)
51. Miller, B.N., Albert, I., Lam, S.K., Konstan, J.A., Riedl, J.: MovieLens unplugged: experiences with an occasionally connected recommender system. In: *IUI '03: Proc. of the 8th Int. Conf. on Intelligent User Interfaces*, pp. 263–266. ACM, New York, NY, USA (2003)
52. Mobasher, B., Dai, H., Luo, T., Nakagawa, M.: Discovery and evaluation of aggregate usage profiles for Web personalization. *Data Mining and Knowledge Discovery* **6**(1), 61–82 (2002)
53. Mooney, R.J.: Content-based book recommending using learning for text categorization. In: *Proc. of the Fifth ACM Conf. on Digital Libraries*, pp. 195–204. ACM Press (2000)
54. Nakamura, A., Abe, N.: Collaborative filtering using weighted majority prediction algorithms. In: *ICML '98: Proc. of the 15th Int. Conf. on Machine Learning*, pp. 395–403. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
55. Netflix: Online movie rental service (2009). <http://www.netflix.com>
56. Norris, J.R.: *Markov Chains*, 1 edn. Cambridge University Press, Cambridge (1999)
57. Paterek, A.: Improving regularized singular value decomposition for collaborative filtering. In: *Proceedings of the KDD Cup and Workshop (2007)*
58. Pazzani, M., Billsus, D.: Learning and revising user profiles: The identification of interesting Web sites. *Machine Learning* **27**(3), 313–331 (1997)
59. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review* **13**(5-6), 393–408 (1999)
60. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: GroupLens: An open architecture for collaborative filtering of netnews. In: *CSCW '94: Proc. of the 1994 ACM Conf. on Computer Supported Cooperative Work*, pp. 175–186. ACM, New York, NY, USA (1994)
61. Rich, E.: User modeling via stereotypes. *Cognitive Science* **3**(4), 329–354 (1979)
62. Rissanen, J.: Modeling by shortest data description. *Automatica* **14**, 465–471 (1978)
63. Rocchio, J.: *Relevance Feedback in Information Retrieval*. Prentice Hall, Englewood, Cliffs, New Jersey (1971)
64. Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted Boltzmann machines for collaborative filtering. In: *ICML '07: Proceedings of the 24th international conference on Machine learning*, pp. 791–798. ACM, New York, NY, USA (2007)
65. Salton, G. (ed.): *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1988)
66. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: *WWW '01: Proc. of the 10th Int. Conf. on World Wide Web*, pp. 285–295. ACM, New York, NY, USA (2001)
67. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J.T.: Application of dimensionality reduction in recommender systems: A case study. In: *ACM WebKDD Workshop (2000)*
68. Sarwar, B.M., Konstan, J.A., Borchers, A., Herlocker, J., Miller, B., Riedl, J.: Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In: *CSCW '98: Proc. of the 1998 ACM Conf. on Computer Supported Cooperative Work*, pp. 345–354. ACM, New York, NY, USA (1998)
69. Schein, A.L., Popescul, A., Ungar, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: *SIGIR '02: Proc. of the 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pp. 253–260. ACM, New York, NY, USA (2002)
70. Shardanand, U., Maes, P.: Social information filtering: Algorithms for automating “word of mouth”. In: *CHI '95: Proc. of the SIGCHI Conf. on Human factors in Computing Systems*, pp. 210–217. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995)

71. Sheth, B., Maes, P.: Evolving agents for personalized information filtering. In: Proc. of the 9th Conf. on Artificial Intelligence for Applications, pp. 345–352 (1993)
72. Soboroff, I.M., Nicholas, C.K.: Combining content and collaboration in text filtering. In: Proc. of the IJCAI'99 Workshop on Machine Learning for Information Filtering, pp. 86–91 (1999)
73. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Major components of the gravity recommendation system. SIGKDD Exploration Newsletter **9**(2), 80–83 (2007)
74. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Investigation of various matrix factorization methods for large recommender systems. In: Proc. of the 2nd KDD Workshop on Large Scale Recommender Systems and the Netflix Prize Competition (2008)
75. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Scalable collaborative filtering approaches for large recommender systems. Journal of Machine Learning Research (Special Topic on Mining and Learning with Graphs and Relations) **10**, 623–656 (2009)
76. Terveen, L., Hill, W., Amento, B., McDonald, D., Creter, J.: PHOAKS: a system for sharing recommendations. Communications of the ACM **40**(3), 59–62 (1997)
77. Zhang, Y., Callan, J., Minka, T.: Novelty and redundancy detection in adaptive filtering. In: SIGIR '02: Proc. of the 25th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 81–88. ACM, New York, NY, USA (2002)
78. Zitnick, C.L., Kanade, T.: Maximum entropy for collaborative filtering. In: AUAI '04: Proc. of the 20th Conf. on Uncertainty in Artificial Intelligence, pp. 636–643. AUAI Press, Arlington, Virginia, United States (2004)

Chapter 5

Advances in Collaborative Filtering

Yehuda Koren and Robert Bell

Abstract

The collaborative filtering (CF) approach to recommenders has recently enjoyed much interest and progress. The fact that it played a central role within the recently completed Netflix competition has contributed to its popularity. This chapter surveys the recent progress in the field. Matrix factorization techniques, which became a first choice for implementing CF, are described together with recent innovations. We also describe several extensions that bring competitive accuracy into neighborhood methods, which used to dominate the field. The chapter demonstrates how to utilize temporal models and implicit feedback to extend models accuracy. In passing, we include detailed descriptions of some the central methods developed for tackling the challenge of the Netflix Prize competition.

5.1 Introduction

Collaborative filtering (CF) methods produce user specific recommendations of items based on patterns of ratings or usage (e.g., purchases) without need for ex-

Yehuda Koren
Yahoo! Research, e-mail: yehuda@yahoo-inc.com

Robert Bell
AT&T Labs – Research e-mail: rbell@research.att.com

This article includes copyrighted materials, which were reproduced with permission of ACM and IEEE. The original articles are:

R. Bell and Y. Koren, “Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights”, *IEEE International Conference on Data Mining (ICDM’07)*, pp. 43–52, © 2007 IEEE. Reprinted by permission.

Y. Koren, “Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model”, *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, © 2008 ACM, Inc. Reprinted by permission. <http://doi.acm.org/10.1145/1401890.1401944>

ogenous information about either items or users. While well established methods work adequately for many purposes, we present several recent extensions available to analysts who are looking for the best possible recommendations.

The Netflix Prize competition that began in October 2006 has fueled much recent progress in the field of collaborative filtering. For the first time, the research community gained access to a large-scale, industrial strength data set of 100 million movie ratings—attracting thousands of scientists, students, engineers and enthusiasts to the field. The nature of the competition has encouraged rapid development, where innovators built on each generation of techniques to improve prediction accuracy. Because all methods are judged by the same rigid yardstick on common data, the evolution of more powerful models has been especially efficient.

Recommender systems rely on various types of input. Most convenient is high quality *explicit feedback*, where users directly report on their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons.

Because explicit feedback is not always available, some recommenders infer user preferences from the more abundant *implicit feedback*, which indirectly reflects opinion through observing user behavior [22]. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user who purchased many books by the same author probably likes that author. This chapter focuses on models suitable for explicit feedback. Nonetheless, we recognize the importance of implicit feedback, an especially valuable information source for users who do not provide much explicit feedback. Hence, we show how to address implicit feedback within the models as a secondary source of information.

In order to establish recommendations, CF systems need to relate two fundamentally different entities: items and users. There are two primary approaches to facilitate such a comparison, which constitute the two main techniques of CF: *the neighborhood approach* and *latent factor models*. Neighborhood methods focus on relationships between items or, alternatively, between users. An item-item approach models the preference of a user to an item based on ratings of similar items by the same user. Latent factor models, such as matrix factorization (aka, SVD), comprise an alternative approach by transforming both items and users to the same latent factor space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback.

Producing more accurate prediction methods requires deepening their foundations and reducing reliance on arbitrary decisions. In this chapter, we describe a variety of recent improvements to the primary CF modeling techniques. Yet, the quest for more accurate models goes beyond this. At least as important is the identification of all the signals, or features, available in the data. Conventional techniques address the sparse data of user-item ratings. Accuracy significantly improves by also utilizing other sources of information. One prime example includes all kinds of tem-

Y. Koren. "Collaborative Filtering with Temporal Dynamics." *Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 447–456, © 2009 ACM, Inc. Reprinted by permission. <http://doi.acm.org/10.1145/1557019.1557072>

poral effects reflecting the dynamic, time-drifting nature of user-item interactions. No less important is listening to hidden feedback such as which items users chose to rate (regardless of rating values). Rated items are not selected at random, but rather reveal interesting aspects of user preferences, going beyond the numerical values of the ratings.

Section 5.3 surveys matrix factorization techniques, which combine implementation convenience with a relatively high accuracy. This has made them the preferred technique for addressing the largest publicly available dataset - the Netflix data. This section describes the theory and practical details behind those techniques. In addition, much of the strength of matrix factorization models stems from their natural ability to handle additional features of the data, including implicit feedback and temporal information. This section describes in detail how to enhance matrix factorization models to address such features.

Section 5.4 turns attention to neighborhood methods. The basic methods in this family are well known, and to a large extent are based on heuristics. Some recently proposed techniques address shortcomings of neighborhood techniques by suggesting more rigorous formulations, thereby improving prediction accuracy. We continue at Section 5.5 with a more advanced method, which uses the insights of common neighborhood methods, with global optimization techniques typical of factorization models. This method allows lifting the limit on neighborhood size, and also addressing implicit feedback and temporal dynamics. The resulting accuracy is close to that of matrix factorization models, while offering some practical advantages.

Pushing the foundations of the models to their limits reveals surprising links among seemingly unrelated techniques. We elaborate on this in Section 5.6 to show that, at their limits, user-user and item-item neighborhood models may converge to a single model. Furthermore, at that point, both become equivalent to a simple matrix factorization model. The connections reduce the relevance of some previous distinctions such as the traditional broad categorization of matrix factorization as “model based” and neighborhood models as “memory based”.

5.2 Preliminaries

We are given ratings for m users (aka customers) and n items (aka products). We reserve special indexing letters to distinguish users from items: for users u, v , and for items i, j, l . A rating r_{ui} indicates the preference by user u of item i , where high values mean stronger preference. For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation \hat{r}_{ui} for the predicted value of r_{ui} .

The scalar t_{ui} denotes the time of rating r_{ui} . One can use different time units, based on what is appropriate for the application at hand. For example, when time is measured in days, then t_{ui} counts the number of days elapsed since some early

time point. Usually the vast majority of ratings are unknown. For example, in the Netflix data 99% of the possible ratings are missing because a user typically rates only a small portion of the movies. The (u, i) pairs for which r_{ui} is known are stored in the set $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$. Each user u is associated with a set of items denoted by $R(u)$, which contains all the items for which ratings by u are available. Likewise, $R(i)$ denotes the set of users who rated item i . Sometimes, we also use a set denoted by $N(u)$, which contains all items for which u provided an implicit preference (items that he rented/purchased/watched, etc.).

Models for the rating data are learnt by fitting the previously observed ratings. However, our goal is to generalize those in a way that allows us to predict future, unknown ratings. Thus, caution should be exercised to avoid overfitting the observed data. We achieve this by regularizing the learnt parameters, whose magnitudes are penalized. Regularization is controlled by constants which are denoted as: $\lambda_1, \lambda_2, \dots$. Exact values of these constants are determined by cross validation. As they grow, regularization becomes heavier.

5.2.1 Baseline predictors

CF models try to capture the interactions between users and items that produce the different rating values. However, much of the observed rating values are due to effects associated with either users or items, independently of their interaction. A principal example is that typical CF data exhibit large user and item biases – i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others.

We will encapsulate those effects, which do not involve user-item interaction, within the *baseline predictors* (also known as *biases*). Because these predictors tend to capture much of the observed signal, it is vital to model them accurately. Such modeling enables isolating the part of the signal that truly represents user-item interaction, and subjecting it to more appropriate user preference models.

Denote by μ the overall average rating. A baseline prediction for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i \tag{5.1}$$

The parameters b_u and b_i indicate the observed deviations of user u and item i , respectively, from the average. For example, suppose that we want a baseline predictor for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, μ , is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline predictor for Titanic's rating by Joe would be 3.9 stars by calculating $3.7 - 0.3 + 0.5$. In order to estimate b_u and b_i one can solve the least squares problem

$$\min_{b_*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2).$$

Here, the first term $\sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu + b_u + b_i)^2$ strives to find b_u 's and b_i 's that fit the given ratings. The regularizing term $-\lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$ – avoids overfitting by penalizing the magnitudes of the parameters. This least square problem can be solved fairly efficiently by the method of stochastic gradient descent (described in Subsection 5.3.1).

For the Netflix data the mean rating (μ) is 3.6. As for the learned user biases (b_u), their average is 0.044 with standard deviation of 0.41. The average of their absolute values ($|b_u|$) is: 0.32. The learned item biases (b_i) average to -0.26 with a standard deviation of 0.48. The average of their absolute values ($|b_i|$) is 0.43.

An easier, yet somewhat less accurate way to estimate the parameters is by decoupling the calculation of the b_i 's from the calculation of the b_u 's. First, for each item i we set

$$b_i = \frac{\sum_{u \in \mathbf{R}(i)} (r_{ui} - \mu)}{\lambda_2 + |\mathbf{R}(i)|}.$$

Then, for each user u we set

$$b_u = \frac{\sum_{i \in \mathbf{R}(u)} (r_{ui} - \mu - b_i)}{\lambda_3 + |\mathbf{R}(u)|}.$$

Averages are shrunk towards zero by using the regularization parameters, λ_2, λ_3 , which are determined by cross validation. Typical values on the Netflix dataset are: $\lambda_2 = 25, \lambda_3 = 10$.

In Subsection 5.3.3.1, we show how the baseline predictors can be improved by also considering temporal dynamics within the data.

5.2.2 The Netflix data

In order to compare the relative accuracy of algorithms described in this chapter, we evaluated all of them on the Netflix data of more than 100 million date-stamped movie ratings performed by anonymous Netflix customers between November, 1999 and December 2005 [5]. Ratings are integers ranging between 1 and 5. The data spans 17,770 movies rated by over 480,000 users. Thus, on average, a movie receives 5600 ratings, while a user rates 208 movies, with substantial variation around each of these averages. To maintain compatibility with results published by others, we adopt some standards that were set by Netflix. First, quality of the results is usually measured by the root mean squared error (RMSE):

$$\sqrt{\sum_{(u,i) \in \text{TestSet}} (r_{ui} - \hat{r}_{ui})^2 / |\text{TestSet}|}$$

a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. (Consider Chapter 8 for a comprehensive survey of alternative evaluation metrics of recommender systems.)

We report results on a test set provided by Netflix (also known as the Quiz set), which contains over 1.4 million recent ratings. Compared with the training data, the test set contains many more ratings by users that do not rate much and are therefore harder to predict. In a way, this represents real requirements for a CF system, which needs to predict new ratings from older ones, and to equally address all users, not just the heavy raters.

The Netflix data is part of the Netflix Prize competition, where the benchmark is Netflix's proprietary system, Cinematch, which achieved a RMSE of 0.9514 on the test set. The grand prize was awarded to a team that managed to drive this RMSE below 0.8563 (10% improvement) after almost three years of extensive efforts. Achievable RMSE values on the test set lie in a quite compressed range, as evident by the difficulty to win the grand prize. Nonetheless, there is evidence that small improvements in RMSE terms can have a significant impact on the quality of the top few presented recommendations [17, 19].

5.2.3 *Implicit feedback*

This chapter is centered on explicit user feedback. Nonetheless, when additional sources of implicit feedback are available, they can be exploited for better understanding user behavior. This helps to combat data sparseness and can be particularly helpful for users with few explicit ratings. We describe extensions for some of the models to address implicit feedback.

For a dataset such as the Netflix data, the most natural choice for implicit feedback would probably be movie rental history, which tells us about user preferences without requiring them to explicitly provide their ratings. For other datasets, browsing or purchase history could be used as implicit feedback. However, such data is not available to us for experimentation. Nonetheless, a less obvious kind of implicit data does exist within the Netflix dataset. The dataset does not only tell us the rating values, but also *which* movies users rate, regardless of *how* they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This creates a binary matrix, where “1” stands for “rated”, and “0” for “not rated”. While this binary data may not be as informative as other independent sources of implicit feedback, incorporating this kind of implicit data does significantly improve prediction accuracy. The benefit of using the binary data is closely related to the fact that ratings are not missing at random; users deliberately choose which items to rate (see Marlin et al. [21]).

5.3 Matrix factorization models

Latent factor models approach collaborative filtering with the holistic goal to uncover latent features that explain observed ratings; examples include pLSA [15], neural networks [24], Latent Dirichlet Allocation [7], and models that are induced by factorization of the user-item ratings matrix (also known as SVD-based models). Recently, matrix factorization models have gained popularity, thanks to their attractive accuracy and scalability.

In information retrieval, SVD is well established for identifying latent semantic factors [9]. However, applying SVD to explicit ratings in the CF domain raises difficulties due to the high portion of missing values. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier works relied on imputation [16, 26], which fills in missing ratings and makes the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, the data may be considerably distorted due to inaccurate imputation. Hence, more recent works [4, 6, 10, 17, 23, 24, 28] suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model.

In this section we describe several matrix factorization techniques, with increasing complexity and accuracy. We start with the basic model – “SVD”. Then, we show how to integrate other sources of user feedback in order to increase prediction accuracy, through the “SVD++ model”. Finally we deal with the fact that customer preferences for products may drift over time. Product perception and popularity are constantly changing as new selection emerges. Similarly, customer inclinations are evolving, leading them to ever redefine their taste. This leads to a factor model that addresses temporal dynamics for better tracking user behavior.

5.3.1 SVD

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f , such that user-item interactions are modeled as inner products in that space. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, factors might measure obvious dimensions such as comedy vs. drama, amount of action, or orientation to children; less well defined dimensions such as depth of character development or “quirkiness”; or completely uninterpretable dimensions.

Accordingly, each item i is associated with a vector $q_i \in \mathbb{R}^f$, and each user u is associated with a vector $p_u \in \mathbb{R}^f$. For a given item i , the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user u , the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors (again, these may be positive or negative).

The resulting dot product,¹ $q_i^T p_u$, captures the interaction between user u and item i —i.e., the overall interest of the user in characteristics of the item. The final rating is created by also adding in the aforementioned baseline predictors that depend only on the user or item. Thus, a rating is predicted by the rule

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u. \quad (5.2)$$

In order to learn the model parameters (b_u, b_i, p_u and q_i) we minimize the regularized squared error

$$\min_{b_u, b_i, p_u, q_i} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda_4 (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2).$$

The constant λ_4 , which controls the extent of regularization, is usually determined by cross validation. Minimization is typically performed by either stochastic gradient descent or alternating least squares.

Alternating least squares techniques rotate between fixing the p_u 's to solve for the q_i 's and fixing the q_i 's to solve for the p_u 's. Notice that when one of these is taken as a constant, the optimization problem is quadratic and can be optimally solved; see [2, 4].

An easy stochastic gradient descent optimization was popularized by Funk [10] and successfully practiced by many others [17, 23, 24, 28]. The algorithm loops through all ratings in the training data. For each given rating r_{ui} , a prediction (\hat{r}_{ui}) is made, and the associated prediction error $e_{ui} \stackrel{\text{def}}{=} r_{ui} - \hat{r}_{ui}$ is computed. For a given training case r_{ui} , we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda_4 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_4 \cdot p_u)$

When evaluating the method on the Netflix data, we used the following values for the meta parameters: $\gamma = 0.005$, $\lambda_4 = 0.02$. Henceforth, we dub this method “SVD”.

A general remark is in place. One can expect better accuracy by dedicating separate learning rates (γ) and regularization (λ) to each type of learned parameter. Thus, for example, it is advised to employ distinct learning rates to user biases, item biases and the factors themselves. A good, intensive use of such a strategy is described in Takács et al. [29]. When producing exemplary results for this chapter, we did not use such a strategy consistently, and in particular many of the given constants are not fully tuned.

¹ Recall that the dot product between two vectors $x, y \in \mathbb{R}^f$ is defined as: $x^T y = \sum_{k=1}^f x_k \cdot y_k$

5.3.2 SVD++

Prediction accuracy is improved by considering also implicit feedback, which provides an additional indication of user preferences. This is especially helpful for those users that provided much more implicit feedback than explicit one. As explained earlier, even in cases where independent implicit feedback is absent, one can capture a significant signal by accounting for which items users rate, regardless of their rating value. This led to several methods [17, 23, 25] that modeled a user factor by the identity of the items he/she has rated. Here we focus on the SVD++ method [17], which was shown to offer accuracy superior to SVD.

To this end, a second set of item factors is added, relating each item i to a factor vector $y_i \in \mathbb{R}^f$. Those new item factors are used to characterize users based on the set of items that they rated. The exact model is as follows:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} y_j \right) \quad (5.3)$$

The set $\mathbf{R}(u)$ contains the items rated by user u .

Now, a user u is modeled as $p_u + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} y_j$. We use a free user-factors vector, p_u , much like in (5.2), which is learnt from the given explicit ratings. This vector is complemented by the sum $|\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} y_j$, which represents the perspective of implicit feedback. Since the y_j 's are centered around zero (by the regularization), the sum is normalized by $|\mathbf{R}(u)|^{-\frac{1}{2}}$, in order to stabilize its variance across the range of observed values of $|\mathbf{R}(u)|$.

Model parameters are determined by minimizing the associated regularized squared error function through stochastic gradient descent. We loop over all known ratings in \mathcal{K} , computing:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} y_j) - \lambda_6 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_6 \cdot p_u)$
- $\forall j \in \mathbf{R}(u)$:
 $y_j \leftarrow y_j + \gamma \cdot (e_{ui} \cdot |\mathbf{R}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_6 \cdot y_j)$

When evaluating the method on the Netflix data, we used the following values for the meta parameters: $\gamma = 0.007$, $\lambda_5 = 0.005$, $\lambda_6 = 0.015$. It is beneficial to decrease step sizes (the γ 's) by a factor of 0.9 after each iteration. The iterative process runs for around 30 iterations until convergence.

Several types of implicit feedback can be simultaneously introduced into the model by using extra sets of item factors. For example, if a user u has a certain kind of implicit preference to the items in $\mathbf{N}^1(u)$ (e.g., she rented them), and a different type of implicit feedback to the items in $\mathbf{N}^2(u)$ (e.g., she browsed them), we could use the model

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + |N^1(u)|^{-\frac{1}{2}} \sum_{j \in N^1(u)} y_j^{(1)} + |N^2(u)|^{-\frac{1}{2}} \sum_{j \in N^2(u)} y_j^{(2)} \right). \quad (5.4)$$

The relative importance of each source of implicit feedback will be automatically learned by the algorithm by its setting of the respective values of model parameters.

5.3.3 Time-aware factor model

The matrix-factorization approach lends itself well to modeling temporal effects, which can significantly improve its accuracy. Decomposing ratings into distinct terms allows us to treat different temporal aspects separately. Specifically, we identify the following effects that each vary over time: (1) user biases $b_u(t)$, (2) item biases $b_i(t)$, and (3) user preferences $p_u(t)$. On the other hand, we specify static item characteristics, q_i , because we do not expect significant temporal variation for items, which, unlike humans, are static in nature. We start with a detailed discussion of the temporal effects that are contained within the baseline predictors.

5.3.3.1 Time changing baseline predictors

Much of the temporal variability is included within the baseline predictors, through two major temporal effects. The first addresses the fact that an item’s popularity may change over time. For example, movies can go in and out of popularity as triggered by external events such as the appearance of an actor in a new movie. This is manifested in our models by treating the item bias b_i as a function of time. The second major temporal effect allows users to change their baseline ratings over time. For example, a user who tended to rate an average movie “4 stars”, may now rate such a movie “3 stars”. This may reflect several factors including a natural drift in a user’s rating scale, the fact that ratings are given in relationship to other ratings that were given recently and also the fact that the identity of the rater within a household can change over time. Hence, in our models we take the parameter b_u as a function of time. This induces a template for a time sensitive baseline predictor for u ’s rating of i at day t_{ui} :

$$b_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}) \quad (5.5)$$

Here, $b_u(\cdot)$ and $b_i(\cdot)$ are real valued functions that change over time. The exact way to build these functions should reflect a reasonable way to parameterize the involving temporal changes. Our choice in the context of the movie rating dataset demonstrates some typical considerations.

A major distinction is between temporal effects that span extended periods of time and more transient effects. In the movie rating case, we do not expect movie likability to fluctuate on a daily basis, but rather to change over more extended periods. On the other hand, we observe that user effects can change on a daily basis,

reflecting inconsistencies natural to customer behavior. This requires finer time resolution when modeling user-biases compared with a lower resolution that suffices for capturing item-related time effects.

We start with our choice of time-changing item biases $b_i(t)$. We found it adequate to split the item biases into time-based bins, using a constant item bias for each time period. The decision of how to split the timeline into bins should balance the desire to achieve finer resolution (hence, smaller bins) with the need for enough ratings per bin (hence, larger bins). For the movie rating data, there is a wide variety of bin sizes that yield about the same accuracy. In our implementation, each bin corresponds to roughly ten consecutive weeks of data, leading to 30 bins spanning all days in the dataset. A day t is associated with an integer $\text{Bin}(t)$ (a number between 1 and 30 in our data), such that the movie bias is split into a stationary part and a time changing part

$$b_i(t) = b_i + b_{i, \text{Bin}(t)}. \quad (5.6)$$

While binning the parameters works well on the items, it is more of a challenge on the users side. On the one hand, we would like a finer resolution for users to detect very short lived temporal effects. On the other hand, we do not expect enough ratings per user to produce reliable estimates for isolated bins. Different functional forms can be considered for parameterizing temporal user behavior, with varying complexity and accuracy.

One simple modeling choice uses a linear function to capture a possible gradual drift of user bias. For each user u , we denote the mean date of rating by t_u . Now, if u rated a movie on day t , then the associated time deviation of this rating is defined as

$$\text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^\beta.$$

Here $|t - t_u|$ measures the number of days between dates t and t_u . We set the value of β by cross validation; in our implementation $\beta = 0.4$. We introduce a single new parameter for each user called α_u so that we get our first definition of a time-dependent user-bias

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot \text{dev}_u(t). \quad (5.7)$$

This simple linear model for approximating a drifting behavior requires learning two parameters per user: b_u and α_u .

A more flexible parameterization is offered by splines. Let u be a user associated with n_u ratings. We designate k_u time points $\{t_1^u, \dots, t_{k_u}^u\}$ spaced uniformly across the dates of u 's ratings as kernels that control the following function:

$$b_u^{(2)}(t) = b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|} b_{t_l^u}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|}} \quad (5.8)$$

The parameters $b_{t_l^u}^u$ are associated with the control points (or, kernels), and are automatically learned from the data. This way the user bias is formed as a time-weighted combination of those parameters. The number of control points, k_u , balances flexibility and computational efficiency. In our application we set $k_u = n_u^{0.25}$, letting it grow

with the number of available ratings. The constant σ determines the smoothness of the spline; we set $\sigma=0.3$ by cross validation.

So far we have discussed smooth functions for modeling the user bias, which mesh well with *gradual concept drift*. However, in many applications there are *sudden drifts* emerging as “spikes” associated with a single day or session. For example, in the movie rating dataset we have found that multiple ratings a user gives in a single day, tend to concentrate around a single value. Such an effect need not span more than a single day. The effect may reflect the mood of the user that day, the impact of ratings given in a single day on each other, or changes in the actual rater in multi-person accounts. To address such short lived effects, we assign a single parameter per user and day, absorbing the day-specific variability. This parameter is denoted by $b_{u,t}$. Notice that in some applications the basic primitive time unit to work with can be shorter or longer than a day.

In the Netflix movie rating data, a user rates on 40 different days on average. Thus, working with $b_{u,t}$ requires, on average, 40 parameters to describe each user bias. It is expected that $b_{u,t}$ is inadequate as a standalone for capturing the user bias, since it misses all sorts of signals that span more than a single day. Thus, it serves as an additive component within the previously described schemes. The time-linear model (5.7) becomes

$$b_u^{(3)}(t) = b_u + \alpha_u \cdot \text{dev}_u(t) + b_{u,t}. \quad (5.9)$$

Similarly, the spline-based model becomes

$$b_u^{(4)}(t) = b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|} b_{t_l^u}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t-t_l^u|}} + b_{u,t}. \quad (5.10)$$

A baseline predictor on its own cannot yield personalized recommendations, as it disregards all interactions between users and items. In a sense, it is capturing the portion of the data that is less relevant for establishing recommendations. Nonetheless, to better assess the relative merits of the various choices of time-dependent user-bias, we compare their accuracy as standalone predictors. In order to learn the involved parameters we minimize the associated regularized squared error by using stochastic gradient descent. For example, in our actual implementation we adopt rule (5.9) for modeling the drifting user bias, thus arriving at the baseline predictor

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + b_i + b_{i,\text{Bin}(t_{ui})}. \quad (5.11)$$

To learn the involved parameters, b_u , α_u , $b_{u,t}$, b_i and $b_{i,\text{Bin}(t)}$, one should solve

$$\begin{aligned} \min \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - \alpha_u \text{dev}_u(t_{ui}) - b_{u,t_{ui}} - b_i - b_{i,\text{Bin}(t_{ui})})^2 \\ + \lambda_7 (b_u^2 + \alpha_u^2 + b_{u,t_{ui}}^2 + b_i^2 + b_{i,\text{Bin}(t_{ui})}^2). \end{aligned}$$

model	static	mov	linear	spline	linear+	spline+
RMSE	.9799	.9771	.9731	.9714	.9605	.9603

Table 5.1: Comparing baseline predictors capturing main movie and user effects. As temporal modeling becomes more accurate, prediction accuracy improves (lowering RMSE).

Here, the first term strives to construct parameters that fit the given ratings. The regularization term, $\lambda_7(b_u^2 + \dots)$, avoids overfitting by penalizing the magnitudes of the parameters, assuming a neutral 0 prior. Learning is done by a stochastic gradient descent algorithm running 20–30 iterations, with $\lambda_7 = 0.01$.

Table 5.1 compares the ability of various suggested baseline predictors to explain signal in the data. As usual, the amount of captured signal is measured by the root mean squared error on the test set. As a reminder, test cases come later in time than the training cases for the same user, so predictions often involve extrapolation in terms of time. We code the predictors as follows:

- *static*, no temporal effects: $b_{ui} = \mu + b_u + b_i$.
- *mov*, accounting only for movie-related temporal effects: $b_{ui} = \mu + b_u + b_i + b_{i, \text{Bin}(t_{ui})}$.
- *linear*, linear modeling of user biases: $b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_i + b_{i, \text{Bin}(t_{ui})}$.
- *spline*, spline modeling of user biases: $b_{ui} = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-t_l^u|} b_{l_l}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-t_l^u|}} + b_i + b_{i, \text{Bin}(t_{ui})}$.
- *linear+*, linear modeling of user biases and single day effect: $b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u, t_{ui}} + b_i + b_{i, \text{Bin}(t_{ui})}$.
- *spline+*, spline modeling of user biases and single day effect: $b_{ui} = \mu + b_u + \frac{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-d_l|} b_{l_l}^u}{\sum_{l=1}^{k_u} e^{-\sigma|t_{ui}-d_l|}} + b_{u, t_{ui}} + b_i + b_{i, \text{Bin}(t_{ui})}$.

The table shows that while temporal movie effects reside in the data (lowering RMSE from 0.9799 to 0.9771), the drift in user biases is much more influential. The additional flexibility of splines at modeling user effects leads to better accuracy compared to a linear model. However, sudden changes in user biases, which are captured by the per-day parameters, are most significant. Indeed, when including those changes, the difference between linear modeling (“linear+”) and spline modeling (“spline+”) virtually vanishes.

Beyond the temporal effects described so far, one can use the same methodology to capture more effects. A primary example is capturing periodic effects. For example, some products may be more popular in specific seasons or near certain holidays. Similarly, different types of television or radio shows are popular throughout different segments of the day (known as “dayparting”). Periodic effects can be found also on the user side. As an example, a user may have different attitudes or buying patterns during the weekend compared to the working week. A way to model such periodic effects is to dedicate a parameter for the combinations of time periods with items or users. This way, the item bias of (5.6), becomes

$$b_i(t) = b_i + b_{i,\text{Bin}(t)} + b_{i,\text{period}(t)}.$$

For example, if we try to capture the change of item bias with the season of the year, then $\text{period}(t) \in \{\text{fall, winter, spring, summer}\}$. Similarly, recurring user effects may be modeled by modifying (5.9) to be

$$b_u(t) = b_u + \alpha_u \cdot \text{dev}_u(t) + b_{u,t} + b_{u,\text{period}(t)}.$$

However, we have not found periodic effects with a significant predictive power within the movie-rating dataset, thus our reported results do not include those.

Another temporal effect within the scope of basic predictors is related to the changing scale of user ratings. While $b_i(t)$ is a user-independent measure for the merit of item i at time t , users tend to respond to such a measure differently. For example, different users employ different rating scales, and a single user can change his rating scale over time. Accordingly, the raw value of the movie bias is not completely user-independent. To address this, we add a time-dependent scaling feature to the baseline predictors, denoted by $c_u(t)$. Thus, the baseline predictor (5.11) becomes

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,\text{Bin}(t_{ui})}) \cdot c_u(t_{ui}). \quad (5.12)$$

All discussed ways to implement $b_u(t)$ would be valid for implementing $c_u(t)$ as well. We chose to dedicate a separate parameter per day, resulting in: $c_u(t) = c_u + c_{u,t}$. As usual, c_u is the stable part of $c_u(t)$, whereas $c_{u,t}$ represents day-specific variability. Adding the multiplicative factor $c_u(t)$ to the baseline predictor lowers RMSE to 0.9555. Interestingly, this basic model, which captures just main effects disregarding user-item interactions, can explain almost as much of the data variability as the commercial Netflix Cinematch recommender system, whose published RMSE on the same test set is 0.9514 [5].

5.3.3.2 Time changing factor model

In the previous subsection we discussed the way time affects baseline predictors. However, as hinted earlier, temporal dynamics go beyond this, they also affect user preferences and thereby the interaction between users and items. Users change their preferences over time. For example, a fan of the “psychological thrillers” genre may become a fan of “crime dramas” a year later. Similarly, humans change their perception on certain actors and directors. This type of evolution is modeled by taking the user factors (the vector p_u) as a function of time. Once again, we need to model those changes at the very fine level of a daily basis, while facing the built-in scarcity of user ratings. In fact, these temporal effects are the hardest to capture, because preferences are not as pronounced as main effects (user-biases), but are split over many factors.

We modeled each component of the user preferences $p_u(t)^T = (p_{u1}(t), \dots, p_{uf}(t))$ in the same way that we treated user biases. Within the movie-rating dataset, we have

found modeling after (5.9) effective, leading to

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \text{dev}_u(t) + p_{uk,t} \quad k = 1, \dots, f. \quad (5.13)$$

Here p_{uk} captures the stationary portion of the factor, $\alpha_{uk} \cdot \text{dev}_u(t)$ approximates a possible portion that changes linearly over time, and $p_{uk,t}$ absorbs the very local, day-specific variability.

At this point, we can tie all pieces together and extend the SVD++ factor model by incorporating the time changing parameters. The resulting model will be denoted as *timeSVD++*, where the prediction rule is as follows:

$$\hat{r}_{ui} = \mu + b_i(t_{ui}) + b_u(t_{ui}) + q_i^T \left(p_u(t_{ui}) + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} y_j \right) \quad (5.14)$$

The exact definitions of the time drifting parameters $b_i(t)$, $b_u(t)$ and $p_u(t)$ were given in (5.6), (5.9) and (5.13). Learning is performed by minimizing the associated squared error function on the training set using a regularized stochastic gradient descent algorithm. The procedure is analogous to the one involving the original SVD++ algorithm. Time complexity per iteration is still linear with the input size, while wall clock running time is approximately doubled compared to SVD++, due to the extra overhead required for updating the temporal parameters. Importantly, convergence rate was not affected by the temporal parameterization, and the process converges in around 30 iterations.

5.3.4 Comparison

In Table 5.2 we compare results of the three algorithms discussed in this section. First is SVD, the plain matrix factorization algorithm. Second, is the SVD++ method, which improves upon SVD by incorporating a kind of implicit feedback. Finally is *timeSVD++*, which accounts for temporal effects. The three methods are compared over a range of factorization dimensions (f). All benefit from a growing number of factor dimensions that enables them to better express complex movie-user interactions. Note that the number of parameters in SVD++ is comparable to their number in SVD. This is because SVD++ adds only item factors, while complexity of our dataset is dominated by the much larger set of users. On the other hand, *timeSVD++* requires a significant increase in the number of parameters, because of its refined representation of each user factor. Addressing implicit feedback by the SVD++ model leads to accuracy gains within the movie rating dataset. Yet, the improvement delivered by *timeSVD++* over SVD++ is consistently more significant. We are not aware of any single algorithm in the literature that could deliver such accuracy. Further evidence of the importance of capturing temporal dynamics is the fact that a *timeSVD++* model of dimension 10 is already more accurate than

an SVD model of dimension 200. Similarly, a timeSVD++ model of dimension 20 is enough to outperform an SVD++ model of dimension 200.

Model	$f=10$	$f=20$	$f=50$	$f=100$	$f=200$
SVD	.9140	.9074	.9046	.9025	.9009
SVD++	.9131	.9032	.8952	.8924	.8911
timeSVD++	.8971	.8891	.8824	.8805	.8799

Table 5.2: Comparison of three factor models: prediction accuracy is measured by RMSE (lower is better) for varying factor dimensionality (f). For all models, accuracy improves with growing number of dimensions. SVD++ improves accuracy by incorporating implicit feedback into the SVD model. Further accuracy gains are achieved by also addressing the temporal dynamics in the data through the timeSVD++ model.

5.3.4.1 Predicting future days

Our models include day-specific parameters. An apparent question would be how these models can be used for predicting ratings in the future, on new dates for which we cannot train the day-specific parameters? The simple answer is that for those future (untrained) dates, the day-specific parameters should take their default value. In particular for (5.12), $c_u(t_{ui})$ is set to c_u , and $b_{u,t_{ui}}$ is set to zero. Yet, one wonders, if we cannot use the day-specific parameters for predicting the future, why are they good at all? After all, prediction is interesting only when it is about the future. To further sharpen the question, we should mention the fact that the Netflix test sets include many ratings on dates for which we have no other rating by the same user and hence day-specific parameters cannot be exploited.

To answer this, notice that our temporal modeling makes no attempt to capture future changes. All it is trying to do is to capture transient temporal effects, which had a significant influence on past user feedback. When such effects are identified they must be tuned down, so that we can model the more enduring signal. This allows our model to better capture the long-term characteristics of the data, while letting dedicated parameters absorb short term fluctuations. For example, if a user gave many higher than usual ratings on a particular single day, our models discount those by accounting for a possible day-specific good mood, which does not reflect the longer term behavior of this user. This way, the day-specific parameters accomplish a kind of data cleaning, which improves prediction of future dates.

5.3.5 Summary

In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from patterns of item ratings. High correspondence between item

and user factors leads to recommendation of an item to a user. These methods deliver prediction accuracy superior to other published collaborative filtering techniques. At the same time, they offer a memory efficient compact model, which can be trained relatively easy. Those advantages, together with the implementation ease of gradient based matrix factorization model (SVD), made this the method of choice within the Netflix Prize competition.

What makes these techniques even more convenient is their ability to address several crucial aspects of the data. First, is the ability to integrate multiple forms of user feedback. One can better predict user ratings by also observing other related actions by the same user, such as purchase and browsing history. The proposed SVD++ model leverages multiple sorts of user feedback for improving user profiling.

Another important aspect is the temporal dynamics that make users' tastes evolve over time. Each user and product potentially goes through a distinct series of changes in their characteristics. A mere decay of older instances cannot adequately identify communal patterns of behavior in time changing data. The solution we adopted is to model the temporal dynamics along the whole time period, allowing us to intelligently separate transient factors from lasting ones. The inclusion of temporal dynamics proved very useful in improving quality of predictions, more than various algorithmic enhancements.

5.4 Neighborhood models

The most common approach to CF is based on neighborhood models. Chapter 4 provides an extensive survey on this approach. Its original form, which was shared by virtually all earlier CF systems, is user-user based; see [14] for a good analysis. User-user methods estimate unknown ratings based on recorded ratings of like-minded users.

Later, an analogous item-item approach [20, 27] became popular. In those methods, a rating is estimated using known ratings made by the same user on similar items. Better scalability and improved accuracy make the item-item approach more favorable in many cases [2, 27, 28]. In addition, item-item methods are more amenable to explaining the reasoning behind predictions. This is because users are familiar with items previously preferred by them, but do not know those allegedly like-minded users. We focus mostly on item-item approaches, but the same techniques can be directly applied within a user-user approach; see also Subsection 5.5.2.2.

In general, latent factor models offer high expressive ability to describe various aspects of the data. Thus, they tend to provide more accurate results than neighborhood models. However, most literature and commercial systems (e.g., those of Amazon [20] and TiVo [1]) are based on the neighborhood models. The prevalence of neighborhood models is partly due to their relative simplicity. However, there are more important reasons for real life systems to stick with those models. First, they naturally provide intuitive explanations of the reasoning behind recommendations,

which often enhance user experience beyond what improved accuracy may achieve. (More on explaining recommendations appears in Chapter 15 of this book.) Second, they can provide immediate recommendations based on newly entered user feedback.

The structure of this section is as follows. First, we describe how to estimate the similarity between two items, which is a basic building block of most neighborhood techniques. Then, we move on to the widely used similarity-based neighborhood method, which constitutes a straightforward application of the similarity weights. We identify certain limitations of this similarity based approach. As a consequence, in Subsection 5.4.3 we suggest a way to solve these issues, thereby improving prediction accuracy at the cost of a slight increase in computation time.

5.4.1 Similarity measures

Central to most item-item approaches is a similarity measure between items. Frequently, it is based on the Pearson correlation coefficient, ρ_{ij} , which measures the tendency of users to rate items i and j similarly. Since many ratings are unknown, some items may share only a handful of common observed raters. The empirical correlation coefficient, $\hat{\rho}_{ij}$, is based only on the common user support. It is advised to work with residuals from the baseline predictors (the b_{ui} 's; see Section 5.2.1) to compensate for user- and item-specific deviations. Thus the approximated correlation coefficient is given by

$$\hat{\rho}_{ij} = \frac{\sum_{u \in U(i,j)} (r_{ui} - b_{ui})(r_{uj} - b_{uj})}{\sqrt{\sum_{u \in U(i,j)} (r_{ui} - b_{ui})^2 \cdot \sum_{u \in U(i,j)} (r_{uj} - b_{uj})^2}}. \quad (5.15)$$

The set $U(i, j)$ contains the users who rated both items i and j .

Because estimated correlations based on a greater user support are more reliable, an appropriate similarity measure, denoted by s_{ij} , is a shrunk correlation coefficient of the form

$$s_{ij} \stackrel{\text{def}}{=} \frac{n_{ij} - 1}{n_{ij} - 1 + \lambda_8} \rho_{ij}. \quad (5.16)$$

The variable $n_{ij} = |U(i, j)|$ denotes the number of users that rated both i and j . A typical value for λ_8 is 100.

Such shrinkage can be motivated from a Bayesian perspective; see Section 2.6 of Gelman et al. [11]. Suppose that the true ρ_{ij} are independent random variables drawn from a normal distribution,

$$\rho_{ij} \sim N(0, \tau^2)$$

for known τ^2 . The mean of 0 is justified if the b_{ui} account for both user and item deviations from average. Meanwhile, suppose that

$$\hat{\rho}_{ij} | \rho_{ij} \sim N(\rho_{ij}, \sigma_{ij}^2)$$

for known σ_{ij}^2 . We estimate ρ_{ij} by its posterior mean:

$$E(\rho_{ij} | \hat{\rho}_{ij}) = \frac{\tau^2 \hat{\rho}_{ij}}{\tau^2 + \sigma_{ij}^2}$$

the empirical estimator $\hat{\rho}_{ij}$ shrunk a fraction, $\sigma_{ij}^2 / (\tau^2 + \sigma_{ij}^2)$, of the way toward zero.

Formula (5.16) follows from approximating the variance of a correlation by $\sigma_{ij}^2 = 1 / (n_{ij} - 1)$, the value for ρ_{ij} near 0.

Notice that the literature suggests additional alternatives for a similarity measure [27, 28].

5.4.2 Similarity-based interpolation

Here we describe the most popular approach to neighborhood modeling, and apparently also to CF in general. Our goal is to predict r_{ui} – the unobserved rating by user u for item i . Using the similarity measure, we identify the k items rated by u that are most similar to i . This set of k neighbors is denoted by $S^k(i; u)$. The predicted value of r_{ui} is taken as a weighted average of the ratings of neighboring items, while adjusting for user and item effects through the baseline predictors

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in S^k(i; u)} s_{ij} (r_{uj} - b_{uj})}{\sum_{j \in S^k(i; u)} s_{ij}}. \quad (5.17)$$

Note the dual use of the similarities for both identification of nearest neighbors and as the interpolation weights in equation (5.17).

Sometimes, instead of relying directly on the similarity weights as interpolation coefficients, one can achieve better results by transforming these weights. For example, we have found at several datasets that squaring the correlation-based similarities is helpful. This leads to a rule like: $\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in S^k(i; u)} s_{ij}^2 (r_{uj} - b_{uj})}{\sum_{j \in S^k(i; u)} s_{ij}^2}$. Toscher et al. [31] discuss more sophisticated transformations of these weights.

Similarity-based methods became very popular because they are intuitive and relatively simple to implement. They also offer the following two useful properties:

1. *Explainability.* The importance of explaining automated recommendations is widely recognized [13, 30]; see also Chapter 15. Users expect a system to give a reason for its predictions, rather than presenting “black box” recommendations. Explanations not only enrich the user experience, but also encourage users to interact with the system, fix wrong impressions and improve long-term accuracy. The neighborhood framework allows identifying which of the past user actions are most influential on the computed prediction.

2. *New ratings.* Item-item neighborhood models can provide updated recommendations immediately after users enter new ratings. This includes handling new users as soon as they provide feedback to the system, without needing to re-train the model and estimate new parameters. This assumes that relationships between items (the s_{ij} values) are stable and barely change on a daily basis. Notice that for items new to the system we do have to learn new parameters. Interestingly, this asymmetry between users and items meshes well with common practices: systems need to provide immediate recommendations to new users (or new ratings by old users) who expect quality service. On the other hand, it is reasonable to require a waiting period before recommending items new to the system.

However, standard neighborhood-based methods raise some concerns:

1. The similarity function (s_{ij}), which directly defines the interpolation weights, is arbitrary. Various CF algorithms use somewhat different similarity measures, trying to quantify the elusive notion of user- or item-similarity. Suppose that a particular item is predicted perfectly by a subset of the neighbors. In that case, we would want the predictive subset to receive all the weight, but that is impossible for bounded similarity scores like the Pearson correlation coefficient.
2. Previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item i and a neighbor $j \in S^k(i; u)$ is computed independently of the content of $S^k(i; u)$ and the other similarities: s_{il} for $l \in S^k(i; u) - \{j\}$. For example, suppose that our items are movies, and the neighbors set contains three movies that are highly correlated with each other (e.g., sequels such as “Lord of the Rings 1–3”). An algorithm that ignores the similarity of the three movies when determining their interpolation weights, may end up essentially triple counting the information provided by the group.
3. By definition, the interpolation weights sum to one, which may cause overfitting. Suppose that an item has no useful neighbors rated by a particular user. In that case, it would be best to ignore the neighborhood information, staying with the more robust baseline predictors. Nevertheless, the standard neighborhood formula uses a weighted average of ratings for the uninformative neighbors.
4. Neighborhood methods may not work well if variability of ratings differs substantially among neighbors.

Some of these issues can be fixed to a certain degree, while others are more difficult to solve within the basic framework. For example, the third item, dealing with the sum-to-one constraint, can be alleviated by using the following prediction rule:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in S^k(i; u)} s_{ij} (r_{uj} - b_{uj})}{\lambda_9 + \sum_{j \in S^k(i; u)} s_{ij}} \quad (5.18)$$

The constant λ_9 penalizes the neighborhood portion when there is not much neighborhood information, e.g., when $\sum_{j \in S^k(i; u)} s_{ij} \ll \lambda_9$. Indeed, we have found that setting an appropriate value of λ_9 leads to accuracy improvements over (5.17). Nonetheless, the whole framework here is not justified by a formal model. Thus,

we strive for better results with a more fundamental approach, as we describe in the following.

5.4.3 Jointly derived interpolation weights

In this section we describe a more accurate neighborhood model that overcomes the difficulties discussed above, while retaining known merits of item-item models. As above, we use the similarity measure to define neighbors for each prediction. However, we search for optimum interpolation weights without regard to values of the similarity measure.

Given a set of neighbors $S^k(i;u)$ we need to compute *interpolation weights* $\{\theta_{ij}^u | j \in S^k(i;u)\}$ that enable the best prediction rule of the form

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in S^k(i;u)} \theta_{ij}^u (r_{uj} - b_{uj}). \quad (5.19)$$

Typical values of k (number of neighbors) lie in the range of 20–50; see [2]. During this subsection we assume that baseline predictors have already been removed. Hence, we introduce a notation for the residual ratings: $z_{ui} \stackrel{\text{def}}{=} r_{ui} - b_{ui}$. For notational convenience assume that the items in $S^k(i;u)$ are indexed by $1, \dots, k$.

We seek a formal computation of the interpolation weights that stems directly from their usage within prediction rule (5.19). As explained earlier, it is important to derive all interpolation weights simultaneously to account for interdependencies among the neighbors. We achieve these goals by defining a suitable optimization problem.

5.4.3.1 Formal model

To start, we consider a hypothetical dense case, where all users but u rated both i and *all* its neighbors in $S^k(i;u)$. In that case, we could learn the interpolation weights by modeling the relationships between item i and its neighbors through a least squares problem

$$\min_{\theta^u} \sum_{v \neq u} \left(z_{vi} - \sum_{j \in S^k(i;u)} \theta_{ij}^u z_{vj} \right)^2. \quad (5.20)$$

Notice that the only unknowns here are the θ_{ij}^u 's. The optimal solution to the least squares problem (5.20) is found by differentiation as a solution of a linear system of equations. From a statistics viewpoint, it is equivalent to the result of a linear regression (without intercept) of z_{vi} on the z_{vj} for $j \in S^k(i;u)$. Specifically, the optimal weights are given by

$$Aw = b. \quad (5.21)$$

Here, $w \in \mathbb{R}^k$ is an unknown vector such that w_j stands for the sought coefficient θ_{ij}^u . A is a $k \times k$ matrix defined as

$$A_{jl} = \sum_{v \neq u} z_{vj} z_{vl}. \quad (5.22)$$

Similarly the vector $b \in \mathbb{R}^k$ is given by

$$b_j = \sum_{v \neq u} z_{vj} z_{vi}. \quad (5.23)$$

For a sparse ratings matrix there are likely to be very few users who rated i and all its neighbors $S^k(i; u)$. Accordingly, it would be unwise to base A and b as given in (5.22)–(5.23) only on users with complete data. Even if there are enough users with complete data for A to be nonsingular, that estimate would ignore a large proportion of the information about pairwise relationships among ratings by the same user. However, we can still estimate A and b , up to the same constant, by averaging over the given pairwise support, leading to the following reformulation:

$$\bar{A}_{jl} = \frac{\sum_{v \in U(j,l)} z_{vj} z_{vl}}{|U(j,l)|} \quad (5.24)$$

$$\bar{b}_j = \frac{\sum_{v \in U(i,j)} z_{vj} z_{vi}}{|U(i,j)|} \quad (5.25)$$

As a reminder, $U(j, l)$ is the set of users who rated both j and l .

This is still not enough to overcome the sparseness issue. The elements of \bar{A}_{jl} or \bar{b}_j may differ by orders of magnitude in terms of the number of users included in the average. As discussed previously, averages based on relatively low support (small values of $|U(j, l)|$) can generally be improved by shrinkage towards a common value. Specifically, we compute a baseline value that is defined by taking the average of all possible \bar{A}_{jl} values. Let us denote this baseline value by avg ; its precise computation is described in the next subsection. Accordingly, we define the corresponding $k \times k$ matrix \hat{A} and the vector $\hat{b} \in \mathbb{R}^k$:

$$\hat{A}_{jl} = \frac{|U(j,l)| \cdot \bar{A}_{jl} + \beta \cdot avg}{|U(j,l)| + \beta} \quad (5.26)$$

$$\hat{b}_j = \frac{|U(i,j)| \cdot \bar{b}_j + \beta \cdot avg}{|U(i,j)| + \beta} \quad (5.27)$$

The parameter β controls the extent of the shrinkage. A typical value would be $\beta = 500$.

Our best estimate for A and b are \hat{A} and \hat{b} , respectively. Therefore, we modify (5.21) so that the interpolation weights are defined as the solution of the linear system

$$\hat{A}w = \hat{b}. \quad (5.28)$$

The resulting interpolation weights are used within (5.19) in order to predict r_{ui} .

This method addresses all four concerns raised in Subsection 5.4.2. First, interpolation weights are derived directly from the ratings, not based on any similarity measure. Second, the interpolation weights formula explicitly accounts for relationships among the neighbors. Third, the sum of the weights is not constrained to equal one. If an item (or user) has only weak neighbors, the estimated weights may all be very small. Fourth, the method automatically adjusts for variations among items in their means or variances.

5.4.3.2 Computational issues

Efficient computation of an item-item neighborhood method requires pre-computing certain values associated with each item-item pair for rapid retrieval. First, we need a quick access to all item-item similarities, by pre-computing all s_{ij} values, as explained in Subsection 5.4.1.

Second, we pre-compute all possible entries of \hat{A} and \hat{b} . To this end, for each two items i and j , we compute

$$\bar{A}_{ij} = \frac{\sum_{v \in \mathbf{U}(i,j)} z_{vi} z_{vj}}{|\mathbf{U}(i,j)|}.$$

Then, the aforementioned baseline value avg , which is used in (5.26)-(5.27), is taken as the average entry of the pre-computed $n \times n$ matrix \bar{A} . In fact, we recommend using two different baseline values, one by averaging the non-diagonal entries of \bar{A} and another one by averaging the generally-larger diagonal entries, which have an inherently higher average because they sum only non-negative values. Finally, we derive a full $n \times n$ matrix \hat{A} from \bar{A} by (5.26), using the appropriate value of avg . Here, the non-diagonal average is used when deriving the non-diagonal entries of \hat{A} , whereas the diagonal average is used when deriving the diagonal entries of \hat{A} .

Because of symmetry, it is sufficient to store the values of s_{ij} and \hat{A}_{ij} only for $i \geq j$. Our experience shows that it is enough to allocate one byte for each individual value, so the overall space required for n items is exactly $n(n+1)$ bytes.

Pre-computing all possible entries of matrix \hat{A} saves the otherwise lengthy time needed to construct entries on the fly. After quickly retrieving the relevant entries of \hat{A} , we can compute the interpolation weights by solving a $k \times k$ system of equations (5.28) using a standard linear solver. However, a modest increase in prediction accuracy was achieved when constraining w to be nonnegative through a quadratic program [2]. Solving the system of equations is an overhead over the basic neighborhood method described in Subsection 5.4.2. For typical values of k (between 20 and 50), the extra time overhead is comparable to the time needed for computing the k nearest neighbors, which is common to neighborhood-based approaches. Hence, while the method relies on a much more detailed computation of the interpolation weights compared to previous methods, it does not significantly increase running time; see [2].

5.4.4 Summary

Collaborative filtering through neighborhood-based interpolation is probably the most popular way to create a recommender system. Three major components characterize the neighborhood approach: (1) data normalization, (2) neighbor selection, and (3) determination of interpolation weights.

Normalization is essential to collaborative filtering in general, and in particular to the more local neighborhood methods. Otherwise, even more sophisticated methods are bound to fail, as they mix incompatible ratings pertaining to different unnormalized users or items. We described a suitable approach to data normalization, based around baseline predictors.

Neighborhood selection is another important component. It is directly related to the employed similarity measure. Here, we emphasized the importance of shrinking unreliable similarities, in order to avoid detection of neighbors with a low rating support.

Finally, the success of neighborhood methods depends on the choice of the interpolation weights, which are used to estimate unknown ratings from neighboring known ones. Nevertheless, most known methods lack a rigorous way to derive these weights. We showed how the interpolation weights can be computed as a global solution to an optimization problem that precisely reflects their role.

5.5 Enriching neighborhood models

Most neighborhood methods are local in their nature – concentrating on only a small subset of related ratings. This contrasts with matrix factorization, which casts a very wide net to try to characterize items and users. It appears that accuracy can be improved by employing this global viewpoint, which motivates the methods of this section. We suggest a new neighborhood model drawing on principles of both classical neighborhood methods and matrix factorization models. Like other neighborhood models, the building stones here are item-item relations (or, alternatively, user-user relations), which provide the system some practical advantages discussed earlier. At the same time, much like matrix factorization, the model is centered around a global optimization framework, which improves accuracy by considering the many weak signals existing in the data.

The main method, which is described in Subsection 5.5.1, allows us to enrich the model with implicit feedback data. In addition, it facilitates two new possibilities. First is a factorized neighborhood model, as described in Subsection 5.5.2, bringing great improvements in computational efficiency. Second is a treatment of temporal dynamics, leading to better prediction accuracy, as described in Subsection 5.5.3.

5.5.1 A global neighborhood model

In this subsection, we introduce a neighborhood model based on global optimization. The model offers an improved prediction accuracy, by offering the aforementioned merits of the model described in Subsection 5.4.3, with additional advantages that are summarized as follows:

1. No reliance on arbitrary or heuristic item-item similarities. The new model is cast as the solution to a global optimization problem.
2. Inherent overfitting prevention or “risk control”: the model reverts to robust baseline predictors, unless a user entered sufficiently many relevant ratings.
3. The model can capture the totality of weak signals encompassed in all of a user’s ratings, not needing to concentrate only on the few ratings for most similar items.
4. The model naturally allows integrating different forms of user input, such as explicit and implicit feedback.
5. A highly scalable implementation (Section 5.5.2) allows linear time and space complexity, thus facilitating both item-item and user-user implementations to scale well to very large datasets.
6. Time drifting aspects of the data can be integrated into the model, thereby improving its accuracy; see Subsection 5.5.3.

5.5.1.1 Building the model

We gradually construct the various components of the model, through an ongoing refinement of our formulations. Previous models were centered around *user-specific* interpolation weights – θ_{ij}^u in (5.19) or $s_{ij}/\sum_{j \in S^k(i;u)} s_{ij}$ in (5.17) – relating item i to the items in a user-specific neighborhood $S^k(i;u)$. In order to facilitate global optimization, we would like to abandon such user-specific weights in favor of global item-item weights independent of a specific user. The weight from j to i is denoted by w_{ij} and will be learned from the data through optimization. An initial sketch of the model describes each rating r_{ui} by the equation

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij}. \quad (5.29)$$

This rule starts with the crude, yet robust, baseline predictors (b_{ui}). Then, the estimate is adjusted by summing over *all* ratings by u .

Let us consider the interpretation of the weights. Usually the weights in a neighborhood model represent interpolation coefficients relating unknown ratings to existing ones. Here, we adopt a different viewpoint, that enables a more flexible usage of the weights. We no longer treat weights as interpolation coefficients. Instead, we take weights as part of adjustments, or *offsets*, added to the baseline predictors. This way, the weight w_{ij} is the extent by which we increase our baseline prediction of r_{ui} based on the observed value of r_{uj} . For two related items i and j , we expect w_{ij} to

be high. Thus, whenever a user u rated j higher than expected ($r_{uj} - b_{uj}$ is high), we would like to increase our estimate for u 's rating of i by adding $(r_{uj} - b_{uj})w_{ij}$ to the baseline prediction. Likewise, our estimate will not deviate much from the baseline by an item j that u rated just as expected ($r_{uj} - b_{uj}$ is around zero), or by an item j that is not known to be predictive on i (w_{ij} is close to zero).

This viewpoint suggests several enhancements to (5.29). First, we can use the form of binary user input, which was found beneficial for factorization models. Namely, analyzing which items were rated regardless of rating value. To this end, we add another set of weights, and rewrite (5.29) as

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathbf{R}(u)} [(r_{uj} - b_{uj})w_{ij} + c_{ij}]. \quad (5.30)$$

Similarly, one could employ here another set of implicit feedback, $\mathbf{N}(u)$ —e.g., the set of items rented or purchased by the user—leading to the rule

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j \in \mathbf{N}(u)} c_{ij}. \quad (5.31)$$

Much like the w_{ij} 's, the c_{ij} 's are offsets added to the baseline predictor. For two items i and j , an implicit preference by u for j leads us to adjust our estimate of r_{ui} by c_{ij} , which is expected to be high if j is predictive on i .

Employing global weights, rather than user-specific interpolation coefficients, emphasizes the influence of missing ratings. In other words, a user's opinion is formed not only by what he rated, but also by what he did not rate. For example, suppose that a movie ratings dataset shows that users that rate "Shrek 3" high also gave high ratings to "Shrek 1–2". This will establish high weights from "Shrek 1–2" to "Shrek 3". Now, if a user did not rate "Shrek 1–2" at all, his predicted rating for "Shrek 3" will be penalized, as some necessary weights cannot be added to the sum.

For prior models (5.17) and (5.19) that interpolated $r_{ui} - b_{ui}$ from $\{r_{uj} - b_{uj} | j \in \mathbf{S}^k(i; u)\}$, it was necessary to maintain compatibility between the b_{ui} values and the b_{uj} values. However, here we do not use interpolation, so we can decouple the definitions of b_{ui} and b_{uj} . Accordingly, a more general prediction rule would be: $\hat{r}_{ui} = \tilde{b}_{ui} + \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj})w_{ij} + c_{ij}$. The constant \tilde{b}_{ui} can represent predictions of r_{ui} by other methods such as a latent factor model. Here, we suggest the following rule that was found to work well:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j \in \mathbf{R}(u)} [(r_{uj} - b_{uj})w_{ij} + c_{ij}] \quad (5.32)$$

Importantly, the b_{uj} 's remain constants, which are derived as explained in Section 5.2.1. However, the b_u 's and b_i 's become parameters that are optimized much like the w_{ij} 's and c_{ij} 's.

We have found that it is beneficial to normalize sums in the model leading to the form

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathbf{R}(u)|^{-\alpha} \sum_{j \in \mathbf{R}(u)} [(r_{uj} - b_{uj})w_{ij} + c_{ij}]. \quad (5.33)$$

The constant α controls the extent of normalization. A non-normalized rule ($\alpha = 0$), encourages greater deviations from baseline predictions for users that provided many ratings (high $|\mathbf{R}(u)|$). On the other hand, a fully normalized rule, eliminates the effect of number of ratings on deviations from baseline predictions. In many cases it would be a good practice for recommender systems to have greater deviation from baselines for users that rate a lot. This way, we take more risk with well modeled users that provided much input. For such users we are willing to predict quirkier and less common recommendations. At the same time, we are less certain about the modeling of users that provided only a little input, in which case we would like to stay with safe estimates close to the baseline values. Our experience with the Netflix dataset shows that best results are achieved with $\alpha = 0.5$, as in the prediction rule

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} [(r_{uj} - b_{uj})w_{ij} + c_{ij}]. \quad (5.34)$$

As an optional refinement, complexity of the model can be reduced by pruning parameters corresponding to unlikely item-item relations. Let us denote by $\mathbf{S}^k(i)$ the set of k items most similar to i , as determined by e.g., a similarity measure s_{ij} or a natural hierarchy associated with the item set. Additionally, we use $\mathbf{R}^k(i; u) \stackrel{\text{def}}{=} \mathbf{R}(u) \cap \mathbf{S}^k(i)$.² Now, when predicting r_{ui} according to (5.34), it is expected that the most influential weights will be associated with items similar to i . Hence, we replace (5.34) with

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathbf{R}^k(i; u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}^k(i; u)} [(r_{uj} - b_{uj})w_{ij} + c_{ij}]. \quad (5.35)$$

When $k = \infty$, rule (5.35) coincides with (5.34). However, for other values of k it offers the potential to significantly reduce the number of variables involved.

5.5.1.2 Parameter Estimation

Prediction rule 5.35 allows fast online prediction. More computational work is needed at a pre-processing stage where parameters are estimated. A major design goal of the new neighborhood model was facilitating an efficient global optimization procedure, which prior neighborhood models lacked. Thus, model parameters are learned by solving the regularized least squares problem associated with (5.35):

² Notational clarification: With other neighborhood models it was beneficial to use $\mathbf{S}^k(i; u)$, which denotes the k items most similar to i among those rated by u . Hence, if u rated at least k items, we will always have $|\mathbf{S}^k(i; u)| = k$, regardless of how similar those items are to i . However, $|\mathbf{R}^k(i; u)|$ is typically smaller than k , as some of those items most similar to i were not rated by u .

$$\min_{b_*, w_*, c_*} \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - |\mathbf{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}^k(i;u)} ((r_{uj} - b_{uj})w_{ij} + c_{ij}) \right)^2 + \lambda_{10} \left(b_u^2 + b_i^2 + \sum_{j \in \mathbf{R}^k(i;u)} w_{ij}^2 + c_{ij}^2 \right) \quad (5.36)$$

An optimal solution of this convex problem can be obtained by least square solvers, which are part of standard linear algebra packages. However, we have found that the following simple stochastic gradient descent solver works much faster. Let us denote the prediction error, $r_{ui} - \hat{r}_{ui}$, by e_{ui} . We loop through all known ratings in \mathcal{K} . For a given training case r_{ui} , we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_{10} \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_{10} \cdot b_i)$
- $\forall j \in \mathbf{R}^k(i;u) :$

$$w_{ij} \leftarrow w_{ij} + \gamma \cdot \left(|\mathbf{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_{10} \cdot w_{ij} \right)$$

$$c_{ij} \leftarrow c_{ij} + \gamma \cdot \left(|\mathbf{R}^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_{10} \cdot c_{ij} \right)$$

The meta-parameters γ (step size) and λ_{10} are determined by cross-validation. We used $\gamma = 0.005$ and $\lambda_{10} = 0.002$ for the Netflix data. Another important parameter is k , which controls the neighborhood size. Our experience shows that increasing k always benefits the accuracy of the results on the test set. Hence, the choice of k should reflect a tradeoff between prediction accuracy and computational cost. In Subsection 5.5.2 we will describe a factored version of the model that eliminates this tradeoff by allowing us to work with the most accurate $k = \infty$ while lowering running time.

A typical number of iterations throughout the training data is 15–20. As for time complexity per iteration, let us analyze the most accurate case where $k = \infty$, which is equivalent to using prediction rule (5.34). For each user u and item $i \in \mathbf{R}(u)$ we need to modify $\{w_{ij}, c_{ij} | j \in \mathbf{R}(u)\}$. Thus the overall time complexity of the training phase is $O(\sum_u |\mathbf{R}(u)|^2)$.

5.5.1.3 Comparison of accuracy

Experimental results on the Netflix data with the globally optimized neighborhood model, henceforth dubbed GlobalNgr, are presented in Figure 5.1. We studied the model under different values of parameter k . The solid black curve with square symbols shows that accuracy monotonically improves with rising k values, as root mean squared error (RMSE) falls from 0.9139 for $k = 250$ to 0.9002 for $k = \infty$. (Notice that since the Netflix data contains 17,770 movies, $k = \infty$ is equivalent to $k = 17,769$, where all item-item relations are explored.) We repeated the experiments without using the implicit feedback, that is, dropping the c_{ij} parameters from our

model. The results depicted by the solid black curve with X's show a significant decline in estimation accuracy, which widens as k grows. This demonstrates the value of incorporating implicit feedback into the model.

For comparison we provide the results of the two previously described neighborhood models. First is a similarity-based neighborhood model (in Subsection 5.4.2), which is the most popular CF method in the literature. We denote this model as CorNgr. Second is the more accurate model described in Subsection 5.4.3, which will be denoted as JointNgr. For both these two models, we tried to pick optimal parameters and neighborhood sizes, which were 20 for CorNgr, and 50 for JointNgr. The results are depicted by the dotted and dashed lines, respectively. It is clear that the popular CorNgr method is noticeably less accurate than the other neighborhood models. On the opposite side, GlobalNgr delivers more accurate results even when compared with JointNgr, as long as the value of k is at least 500. Notice that the k value (the x -axis) is irrelevant to the previous models, as their different notion of neighborhood makes neighborhood sizes incompatible. Yet, we observed that while the performance of GlobalNgr keeps improving as more neighbors are added, this was not true with the two other models. For CorNgr and JointNgr, performance peaks with a relatively small number of neighbors and declines thereafter. This may be explained by the fact that in GlobalNgr, parameters are directly learned from the data through a formal optimization procedure that facilitates using many more parameters effectively.

Finally, let us consider running time. Previous neighborhood models require very light pre-processing, though, JointNgr [2] requires solving a small system of equations for each provided prediction. The new model does involve pre-processing where parameters are estimated. However, online prediction is immediate by following rule (5.35). Pre-processing time grows with the value of k . Figure 5.2 shows typical running times per iteration on the Netflix data, as measured on a single processor 3.4GHz Pentium 4 PC.

5.5.2 A factorized neighborhood model

In the previous subsection we presented a more accurate neighborhood model, which is based on prediction rule (5.34) with training time complexity $O(\sum_u |R(u)|^2)$ and space complexity $O(m + n^2)$. (Recall that m is the number of users, and n is the number of items.) We could improve time and space complexity by sparsifying the model through pruning unlikely item-item relations. Sparsification was controlled by the parameter $k \leq n$, which reduced running time and allowed space complexity of $O(m + nk)$. However, as k gets lower, the accuracy of the model declines as well. In addition, sparsification required relying on an external, less natural, similarity measure, which we would have liked to avoid. Thus, we will now show how to retain the accuracy of the full dense prediction rule (5.34), while significantly lowering time and space complexity.

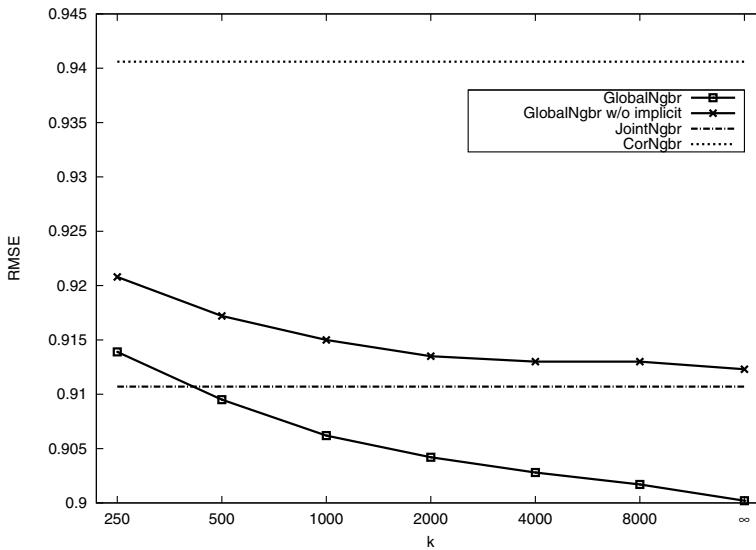


Fig. 5.1: Comparison of neighborhood-based models. Accuracy is measured by RMSE on the Netflix test set, so lower values indicate better performance. We measure the accuracy of the globally optimized model (GlobalNgr) with and without implicit feedback. RMSE is shown as a function of varying values of k , which dictates the neighborhood size. The accuracy of two other models is shown as two horizontal lines; for each we picked an optimal neighborhood size.

5.5.2.1 Factoring item-item relationships

We factor item-item relationships by associating each item i with three vectors: $q_i, x_i, y_i \in \mathbb{R}^f$. This way, we confine w_{ij} to be $q_i^T x_j$. Similarly, we impose the structure $c_{ij} = q_i^T y_j$. Essentially, these vectors strive to map items into an f -dimensional latent factor space where they are measured against various aspects that are revealed automatically by learning from the data. By substituting this into (5.34) we get the following prediction rule:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} [(r_{uj} - b_{uj}) q_i^T x_j + q_i^T y_j] \quad (5.37)$$

Computational gains become more obvious by using the equivalent rule

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(|\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj}) x_j + y_j \right). \quad (5.38)$$

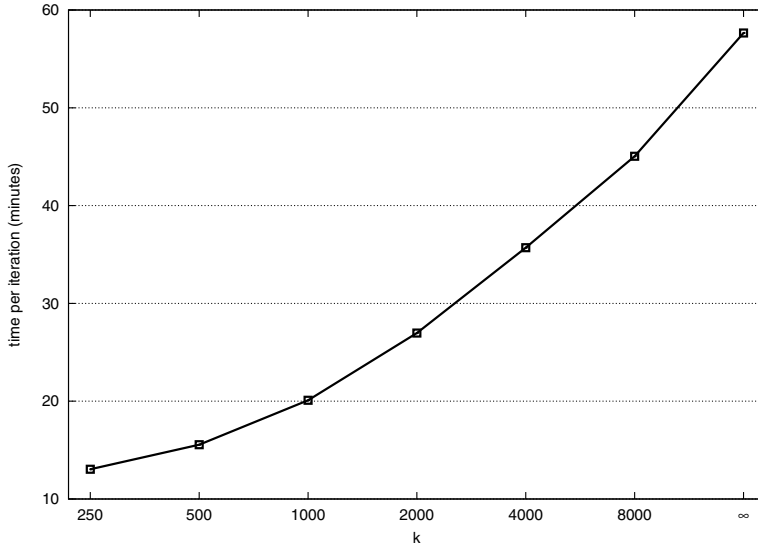


Fig. 5.2: Running time per iteration of the globally optimized neighborhood model, as a function of the parameter k .

Notice that the bulk of the rule $(|\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj})x_j + y_j)$ depends only on u while being independent of i . This leads to an efficient way to learn the model parameters. As usual, we minimize the regularized squared error function associated with (5.38)

$$\min_{q_*, x_*, y_*, b_*} \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - q_i^T \left(|\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj})x_j + y_j \right) \right)^2 + \lambda_{11} \left(b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j \in \mathbf{R}(u)} \|x_j\|^2 + \|y_j\|^2 \right). \quad (5.39)$$

Optimization is done by a stochastic gradient descent scheme, which is described in the following pseudo code:

```

LearnFactorizedNeighborhoodModel(Known ratings:  $r_{ui}$ , rank:  $f$ )
% For each item  $i$  compute  $q_i, x_i, y_i \in \mathbb{R}^f$ 
% which form a neighborhood model
Const #Iterations = 20,  $\gamma = 0.002, \lambda = 0.04$ 
% Gradient descent sweeps:
for count = 1, ..., #Iterations do
  for  $u = 1, \dots, m$  do
    % Compute the component independent of  $i$ :
     $p_u \leftarrow |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj})x_j + y_j$ 
    sum  $\leftarrow 0$ 
    for all  $i \in \mathbf{R}(u)$  do
       $\hat{r}_{ui} \leftarrow \mu + b_u + b_i + q_i^T p_u$ 
       $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$ 
      % Accumulate information for gradient steps on  $x_i, y_i$ :
      sum  $\leftarrow$  sum +  $e_{ui} \cdot q_i$ 
      % Perform gradient step on  $q_i, b_u, b_i$ :
       $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$ 
       $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$ 
       $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$ 
    for all  $i \in \mathbf{R}(u)$  do
      % Perform gradient step on  $x_i$ :
       $x_i \leftarrow x_i + \gamma \cdot (|\mathbf{R}(u)|^{-\frac{1}{2}} \cdot (r_{ui} - b_{ui}) \cdot \text{sum} - \lambda \cdot x_i)$ 
      % Perform gradient step on  $y_i$ :
       $y_i \leftarrow y_i + \gamma \cdot (|\mathbf{R}(u)|^{-\frac{1}{2}} \cdot \text{sum} - \lambda \cdot y_i)$ 
  return  $\{q_i, x_i, y_i | i = 1, \dots, n\}$ 

```

The time complexity of this model is linear with the input size, $O(f \cdot \sum_u (|\mathbf{R}(u)|))$, which is significantly better than the non-factorized model that required time $O(\sum_u |\mathbf{R}(u)|^2)$. We measured the performance of the model on the Netflix data; see Table 5.3. Accuracy is improved as we use more factors (increasing f). However, going beyond 200 factors could barely improve performance, while slowing running time. Interestingly, we have found that with $f \geq 200$ accuracy negligibly exceeds the best non-factorized model (with $k = \infty$). In addition, the improved time complexity translates into a big difference in wall-clock measured running time. For example, the time-per-iteration for the non-factorized model (with $k = \infty$) was close to 58 minutes. On the other hand, a factorized model with $f = 200$ could complete an iteration in 14 minutes without degrading accuracy at all.

The most important benefit of the factorized model is the reduced space complexity, which is $O(m + nf)$ – linear in the input size. Previous neighborhood models required storing all pairwise relations between items, leading to a quadratic space complexity of $O(m + n^2)$. For the Netflix dataset which contains 17,770 movies, such quadratic space can still fit within core memory. Some commercial recommenders process a much higher number of items. For example, an online movie

rental service like Netflix is currently offering over 100,000 titles. Music download shops offer even more titles. Such more comprehensive systems with data on 100,000s items eventually need to resort to external storage in order to fit the entire set of pairwise relations. However, as the number of items is growing towards millions, as in the Amazon item-item recommender system, which accesses stored similarity information for several million catalog items [20], designers must keep a sparse version of the pairwise relations. To this end, only values relating an item to its top- k most similar neighbors are stored thereby reducing space complexity to $O(m+nk)$. However, a sparsification technique will inevitably degrade accuracy by missing important relations, as demonstrated in the previous section. In addition, identification of the top k most similar items in such a high dimensional space is a non-trivial task that can require considerable computational efforts. All these issues do not exist in our factorized neighborhood model, which offers a linear time and space complexity without trading off accuracy.

#factors	50	100	200	500
RMSE	0.9037	0.9013	0.9000	0.8998
time/iteration	4.5 min	8 min	14 min	34 min

Table 5.3: Performance of the factorized item-item neighborhood model. The models with ≥ 200 factors slightly outperform the non-factorized model, while providing much shorter running time.

The factorized neighborhood model resembles some latent factor models. The important distinction is that here we factorize the item-item relationships, rather than the ratings themselves. The results reported in Table 5.3 are comparable to those of the widely used SVD model, but not as good as those of SVD++; see Section 5.3. Nonetheless, the factorized neighborhood model retains the practical advantages of traditional neighborhood models discussed earlier—the abilities to explain recommendations and to immediately reflect new ratings.

As a side remark, we would like to mention that the decision to use three separate sets of factors was intended to give us more flexibility. Indeed, on the Netflix data this allowed achieving most accurate results. However, another reasonable choice could be using a smaller set of vectors, e.g., by requiring: $q_i = x_i$ (implying symmetric weights: $w_{ij} = w_{ji}$).

5.5.2.2 A user-user model

A user-user neighborhood model predicts ratings by considering how like-minded users rated the same items. Such models can be implemented by switching the roles of users and items throughout our derivation of the item-item model. Here, we would like to concentrate on a user-user model, which is dual to the item-item model of

(5.34). The major difference is replacing the w_{ij} weights relating item pairs, with weights relating user pairs:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathbf{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathbf{R}(i)} (r_{vi} - b_{vi}) w_{uv} \quad (5.40)$$

The set $\mathbf{R}(i)$ contains all the users who rated item i . Notice that here we decided to not account for implicit feedback. This is because adding such feedback was not very beneficial for the user-user model when working with the Netflix data.

User-user models can become useful in various situations. For example, some recommenders may deal with items that are rapidly replaced, thus making item-item relations very volatile. On the other hand, a stable user base enables establishment of long term relationships between users. An example of such a case is a recommender system for web articles or news items, which are rapidly changing by their nature; see, e.g., [8]. In such cases, systems centered around user-user relations are more appealing.

In addition, user-user approaches identify different kinds of relations that item-item approaches may fail to recognize, and thus can be useful on certain occasions. For example, suppose that we want to predict r_{ui} , but none of the items rated by user u is really relevant to i . In this case, an item-item approach will face obvious difficulties. However, when employing a user-user perspective, we may find a set of users similar to u , who rated i . The ratings of i by these users would allow us to improve prediction of r_{ui} .

The major disadvantage of user-user models is computational. Since typically there are many more users than items, pre-computing and storing all user-user relations, or even a reasonably sparsified version thereof, is overly expensive or completely impractical. In addition to the high $O(m^2)$ space complexity, the time complexity for optimizing model (5.40) is also much higher than its item-item counterpart, being $O(\sum_i |\mathbf{R}(i)|^2)$ (notice that $|\mathbf{R}(i)|$ is expected to be much higher than $|\mathbf{R}(u)|$). These issues have rendered user-user models as a less practical choice.

A factorized model. All those computational differences disappear by factorizing the user-user model along the same lines as in the item-item model. Now, we associate each user u with two vectors $p_u, z_u \in \mathbb{R}^f$. We assume the user-user relations to be structured as: $w_{uv} = p_u^T z_v$. Let us substitute this into (5.40) to get

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathbf{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathbf{R}(i)} (r_{vi} - b_{vi}) p_u^T z_v. \quad (5.41)$$

Once again, an efficient computation is achieved by including the terms that depends on i but are independent of u in a separate sum, so the prediction rule is presented in the equivalent form

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T |\mathbf{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathbf{R}(i)} (r_{vi} - b_{vi}) z_v. \quad (5.42)$$

In a parallel fashion to the item-item model, all parameters are learned in linear time $O(f \cdot \sum_i |\mathbf{R}(i)|)$. The space complexity is also linear with the input size being $O(n + mf)$. This significantly lowers the complexity of the user-user model compared to previously known results. In fact, running time measured on the Netflix data shows that now the user-user model is even faster than the item-item model; see Table 5.4. We should remark that unlike the item-item model, our implementation of the user-user model did not account for implicit feedback, which probably led to its shorter running time. Accuracy of the user-user model is significantly better than that of the widely-used correlation-based item-item model that achieves RMSE=0.9406 as reported in Figure 5.1. Furthermore, accuracy is slightly better than the variant of the item-item model, which also did not account for implicit feedback (yellow curve in Figure 5.1). This is quite surprising given the common wisdom that item-item methods are more accurate than user-user ones. It appears that a well implemented user-user model can match speed and accuracy of an item-item model. However, our item-item model could significantly benefit by accounting for implicit feedback.

#factors	50	100	200	500
RMSE	0.9119	0.9110	0.9101	0.9093
time/iteration	3 min	5 min	8.5 min	18 min

Table 5.4: Performance of the factorized user-user neighborhood model.

Fusing item-item and user-user models. Since item-item and user-user models address different aspects of the data, overall accuracy is expected to improve by combining predictions of both models. Such an approach was previously suggested and was shown to improve accuracy; see, e.g. [4, 32]. However, past efforts were based on blending the item-item and user-user predictions during a post-processing stage, after each individual model was trained independently of the other model. A more principled approach optimizes the two models simultaneously, letting them know of each other while parameters are being learned. Thus, throughout the entire training phase each model is aware of the capabilities of the other model and strives to complement it. Our approach, which states the neighborhood models as formal optimization problems, allows doing that naturally. We devise a model that sums the item-item model (5.37) and the user-user model (5.41), leading to

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} [(r_{uj} - b_{uj}) q_u^T x_j + q_u^T y_j] \\ & + |\mathbf{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathbf{R}(i)} (r_{vi} - b_{vi}) p_u^T z_v. \end{aligned} \quad (5.43)$$

Model parameters are learned by stochastic gradient descent optimization of the associated squared error function. Our experiments with the Netflix data show that prediction accuracy is indeed better than that of each individual model. For example,

with 100 factors the obtained RMSE is 0.8966, while with 200 factors the obtained RMSE is 0.8953.

Here we would like to comment that our approach allows integrating the neighborhood models also with completely different models in a similar way. For example, in [17] we showed an integrated model that combines the item-item model with a latent factor model (SVD++), thereby achieving improved prediction accuracy with RMSE below 0.887. Therefore, other possibilities with potentially better accuracy should be explored before considering the integration of item-item and user-user models.

5.5.3 Temporal dynamics at neighborhood models

One of the advantages of the item-item model based on global optimization (Subsection 5.5.1), is that it enables us to capture temporal dynamics in a principled manner. As we commented earlier, user preferences are drifting over time, and hence it is important to introduce temporal aspects into CF models.

When adapting rule (5.34) to address temporal dynamics, two components should be considered separately. First component, $\mu + b_i + b_u$, corresponds to the baseline predictor portion. Typically, this component explains most variability in the observed signal. Second component, $|\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj}) w_{ij} + c_{ij}$, captures the more informative signal, which deals with user-item interaction. As for the baseline part, nothing changes from the factor model, and we replace it with $\mu + b_i(t_{ui}) + b_u(t_{ui})$, according to (5.6) and (5.9). However, capturing temporal dynamics within the interaction part requires a different strategy.

Item-item weights (w_{ij} and c_{ij}) reflect inherent item characteristics and are not expected to drift over time. The learning process should capture unbiased long term values, without being too affected from drifting aspects. Indeed, the time changing nature of the data can mask much of the longer term item-item relationships if not treated adequately. For instance, a user rating both items i and j high within a short time period, is a good indicator for relating them, thereby pushing higher the value of w_{ij} . On the other hand, if those two ratings are given five years apart, while the user's taste (if not her identity) could considerably change, this provides less evidence of any relation between the items. On top of this, we would argue that those considerations are pretty much user-dependent; some users are more consistent than others and allow relating their longer term actions.

Our goal here is to distill accurate values for the item-item weights, despite the interfering temporal effects. First we need to parameterize the decaying relations between two items rated by user u . We adopt exponential decay formed by the function $e^{-\beta_u \Delta t}$, where $\beta_u > 0$ controls the user specific decay rate and should be learned from the data. We also experimented with other decay forms, like the more computationally-friendly $(1 + \beta_u \Delta t)^{-1}$, which resulted in about the same accuracy, with an improved running time.

This leads to the prediction rule

$$\hat{r}_{ui} = \mu + b_i(t_{ui}) + b_u(t_{ui}) + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - b_{uj})w_{ij} + c_{ij}). \quad (5.44)$$

The involved parameters, $b_i(t_{ui}) = b_i + b_{i, \text{Bin}(t_{ui})}$, $b_u(t_{ui}) = b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u, t_{ui}}$, β_u , w_{ij} and c_{ij} , are learned by minimizing the associated regularized squared error

$$\begin{aligned} & \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_i - b_{i, \text{Bin}(t_{ui})} - b_u - \alpha_u \text{dev}_u(t_{ui}) - b_{u, t_{ui}} - \right. \\ & \quad \left. |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - b_{uj})w_{ij} + c_{ij}) \right)^2 + \\ & \quad \lambda_{12} \left(b_i^2 + b_{i, \text{Bin}(t_{ui})}^2 + b_u^2 + \alpha_u^2 + b_{u, t}^2 + w_{ij}^2 + c_{ij}^2 \right). \end{aligned} \quad (5.45)$$

Minimization is performed by stochastic gradient descent. We run the process for 25 iterations, with $\lambda_{12} = 0.002$, and step size (learning rate) of 0.005. An exception is the update of the exponent β_u , where we are using a much smaller step size of 10^{-7} . Training time complexity is the same as the original algorithm, which is: $O(\sum_u |\mathbf{R}(u)|^2)$. One can tradeoff complexity with accuracy by sparsifying the set of item-item relations as explained in Subsection 5.5.1.

As in the factor case, properly considering temporal dynamics improves the accuracy of the neighborhood model within the movie ratings dataset. The RMSE decreases from 0.9002 [17] to 0.8885. To our best knowledge, this is significantly better than previously known results by neighborhood methods. To put this in some perspective, this result is even better than those reported by using hybrid approaches such as applying a neighborhood approach on residuals of other algorithms [2, 23, 31]. A lesson is that addressing temporal dynamics in the data can have a more significant impact on accuracy than designing more complex learning algorithms.

We would like to highlight an interesting point. Let u be a user whose preferences are quickly drifting (β_u is large). Hence, old ratings by u should not be very influential on his status at the current time t . One could be tempted to decay the weight of u 's older ratings, leading to "instance weighting" through a cost function like

$$\begin{aligned} & \sum_{(u,i) \in \mathcal{K}} e^{-\beta_u \cdot |t - t_{ui}|} \left(r_{ui} - \mu - b_i - b_{i, \text{Bin}(t_{ui})} - b_u - \alpha_u \text{dev}_u(t_{ui}) - \right. \\ & \quad \left. b_{u, t_{ui}} - |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} ((r_{uj} - b_{uj})w_{ij} + c_{ij}) \right)^2 + \lambda_{12}(\dots). \end{aligned}$$

Such a function is focused at the *current* state of the user (at time t), while de-emphasizing past actions. We would argue against this choice, and opt for equally weighting the prediction error at all past ratings as in (5.45), thereby modeling *all* past user behavior. Therefore, equal-weighting allows us to exploit the signal at each of the past ratings, a signal that is extracted as item-item weights. Learning those

weights would equally benefit from all ratings by a user. In other words, we can deduce that two items are related if users rated them similarly within a short time frame, even if this happened long ago.

5.5.4 Summary

This section follows a less traditional neighborhood based model, which unlike previous neighborhood methods is based on formally optimizing a global cost function. The resulting model is no longer localized, considering relationships between a small set of strong neighbors, but rather considers all possible pairwise relations. This leads to improved prediction accuracy, while maintaining some merits of the neighborhood approach such as explainability of predictions and ability to handle new ratings (or new users) without re-training the model.

The formal optimization framework offers several new possibilities. First, is a factorized version of the neighborhood model, which improves its computational complexity while retaining prediction accuracy. In particular, it is free from the quadratic storage requirements that limited past neighborhood models.

Second addition is the incorporation of temporal dynamics into the model. In order to reveal accurate relations among items, a proposed model learns how influence between two items rated by a user decays over time. Much like in the matrix factorization case, accounting for temporal effects results in a significant improvement in predictive accuracy.

5.6 Between neighborhood and factorization

This chapter was structured around two different approaches to CF: factorization and neighborhood. Each approach evolved from different basic principles, which led to distinct prediction rules. We also argued that factorization can lead to somewhat more accurate results, while neighborhood models may have some practical advantages. In this section we will show that despite those differences, the two approaches share much in common. After all, they are both *linear models*.

Let us consider the SVD model of Subsection 5.3.1, based on

$$\hat{r}_{ui} = q_i^T p_u. \quad (5.46)$$

For simplicity, we ignore here the baseline predictors, but one can easily reintroduce them or just assume that they were subtracted from all ratings at an earlier stage.

We arrange all item-factors within the $n \times f$ matrix $Q = [q_1 q_2 \dots q_n]^T$. Similarly, we arrange all user-factors within the $m \times f$ matrix $P = [p_1 p_2 \dots p_m]^T$. We use the $n_u \times f$ matrix $Q[u]$ to denote the restriction of Q to the items rated by u , where $n_u = |\mathbf{R}(u)|$. Let the vector $r_u \in \mathbb{R}^{n_u}$ contain the ratings given by u ordered as in

$Q[u]$. Now, by activating (5.46) on all ratings given by u , we can reformulate it in a matrix form

$$\hat{r}_u = Q[u]p_u \quad (5.47)$$

For $Q[u]$ fixed, $\|r_u - Q[u]p_u\|_2$ is minimized by

$$p_u = (Q[u]^T Q[u])^{-1} Q[u]^T r_u$$

In practice, we will regularize with $\lambda \geq 0$ to get

$$p_u = (Q[u]^T Q[u] + \lambda I)^{-1} Q[u]^T r_u.$$

By substituting p_u in (5.47) we get

$$\hat{r}_u = Q[u](Q[u]^T Q[u] + \lambda I)^{-1} Q[u]^T r_u. \quad (5.48)$$

This expression can be simplified by introducing some new notation. Let us denote the $f \times f$ matrix $(Q[u]^T Q[u] + \lambda I)^{-1}$ as W^u , which should be considered as a weighting matrix associated with user u . Accordingly, the weighted similarity between items i and j from u 's viewpoint is denoted by $s_{ij}^u = q_i^T W^u q_j$. Using this new notation and (5.48) the predicted preference of u for item i by SVD is rewritten as

$$\hat{r}_{ui} = \sum_{j \in R(u)} s_{ij}^u r_{uj}. \quad (5.49)$$

We reduced the SVD model into a linear model that predicts preferences as a linear function of past actions, weighted by item-item similarity. Each past action receives a separate term in forming the prediction \hat{r}_{ui} . This is equivalent to an item-item neighborhood model. Quite surprisingly, we transformed the matrix factorization model into an item-item model, which is characterized by:

- Interpolation is made from *all* past user ratings, not only from those associated with items most similar to the current one.
- The weight relating items i and j is factorized as a product of two vectors, one related to i and the other to j .
- Item-item weights are subject to a user-specific normalization, through the matrix W^u .

Those properties support our findings on how to best construct a neighborhood model. First, we showed in Subsection 5.5.1 that best results for neighborhood models are achieved when the neighborhood size (controlled by constant k) is maximal, such that all past user ratings are considered. Second, in Subsection 5.5.2 we touted the practice of factoring item-item weights. As for the user-specific normalization, we used a simpler normalizer: $n_u^{-0.5}$. It is likely that SVD suggests a more fundamental normalization by the matrix W^u , which would work better. However, computing W^u would be expensive in practice. Another difference between our suggested item-item model and the one implied by SVD is that we chose to work with asymmetric weights ($w_{ij} \neq w_{ji}$), whereas in the SVD-induced rule: $s_{ij}^u = s_{ji}^u$.

In the derivation above we showed how SVD induces an equivalent item-item technique. In a fully analogous way, it can induce an equivalent user-user technique, by expressing q_i as a function of the ratings and user factors. This brings us to three equivalent models: SVD, item-item and user-user. Beyond linking SVD with neighborhood models, this also shows that user-user and item-item approaches, once well designed, are equivalent.

This last relation (between user-user and item-item approaches) can also be approached intuitively. Neighborhood models try to relate users to new items by following chains of user-item adjacencies. Such adjacencies represent preference- or rating-relations between the respective users and items. Both user-user and item-item models act by following exactly the same chains. They only differ in which “shortcuts” are exploited to speed up calculations. For example, recommending itemB to user1 would follow the chain user1–itemA–user2–itemB (user1 rated itemA, which was also rated by user2, who rated itemB). A user-user model follows such a chain with pre-computed user-user similarities. This way, it creates a “shortcut” that bypasses the sub-chain user1–itemB–user2, replacing it with a similarity value between user1 and user2. Analogously, an item-item approach follows exactly the same chain, but creates an alternative “shortcut”, replacing the sub-chain itemA–user2–itemB with an itemA–itemB similarity value.

Another lesson here is that the distinction that deems neighborhood models as “memory based”, while taking matrix factorization and the likes as “model based” is not always appropriate, at least not when using accurate neighborhood models that are model-based as much as SVD. In fact, the other direction is also true. The better matrix factorization models, such as SVD++, are also following memory-based habits, as they sum over all memory stored ratings when doing the online prediction; see rule (5.3). Hence, the traditional separation between “memory based” and “model based” techniques is not appropriate for categorizing the techniques surveyed in this chapter.

So far, we concentrated on relations between neighborhood models and matrix factorization models. However, in practice it may be beneficial to break these relations, and to augment factorization models with sufficiently different neighborhood models that are able to complement them. Such a combination can lead to improved prediction accuracy [3, 17]. A key to achieve this is by using the more localized neighborhood models (those of Section 5.4, rather than those of Section 5.5), where the number of considered neighbors is limited. The limited number of neighbors might not be the best way to construct a standalone neighborhood model, but it makes the neighborhood model different enough from the factorization model in order to add a local perspective that the rather global factorization model misses.

References

1. Ali, K., and van Stam, W., “TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture”, *Proc. 10th ACM SIGKDD Int. Conference on Knowledge*

- Discovery and Data Mining*, pp. 394–401, 2004.
2. Bell, R., and Koren, Y., “Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights”, *IEEE International Conference on Data Mining (ICDM’07)*, pp. 43–52, 2007.
 3. Bell, R., and Koren, Y., “Lessons from the Netflix Prize Challenge”, *SIGKDD Explorations* **9** (2007), 75–79.
 4. Bell, R.M., Koren, Y., and Volinsky, C., “Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems”, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
 5. Bennet, J., and Lanning, S., “The Netflix Prize”, *KDD Cup and Workshop*, 2007. www.netflixprize.com.
 6. Canny, J., “Collaborative Filtering with Privacy via Factor Analysis”, *Proc. 25th ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR’02)*, pp. 238–245, 2002.
 7. Blei, D., Ng, A., and Jordan, M., “Latent Dirichlet Allocation”, *Journal of Machine Learning Research* **3** (2003), 993–1022.
 8. Das, A., Datar, M., Garg, A., and Rajaram, S., “Google News Personalization: Scalable Online Collaborative Filtering”, *WWW’07*, pp. 271–280, 2007.
 9. Deerwester, S., Dumais, S., Furnas, G.W., Landauer, T.K. and Harshman, R., “Indexing by Latent Semantic Analysis”, *Journal of the Society for Information Science* **41** (1990), 391–407.
 10. Funk, S., “Netflix Update: Try This At Home”, <http://sifter.org/~simon/journal/20061211.html>, 2006.
 11. Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D.B., *Bayesian Data Analysis*, Chapman and Hall, 1995.
 12. Goldberg, D., Nichols, D., Oki, B.M., and Terry, D., “Using Collaborative Filtering to Weave an Information Tapestry”, *Communications of the ACM* **35** (1992), 61–70.
 13. Herlocker, J.L., Konstan, J.A., and Riedl, J., “Explaining Collaborative Filtering Recommendations”, *Proc. ACM Conference on Computer Supported Cooperative Work*, pp. 241–250, 2000.
 14. Herlocker, J.L., Konstan, J.A., Borchers, A., and Riedl, J., “An Algorithmic Framework for Performing Collaborative Filtering”, *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
 15. Hofmann, T., “Latent Semantic Models for Collaborative Filtering”, *ACM Transactions on Information Systems* **22** (2004), 89–115.
 16. Kim, D., and Yum, B., “Collaborative Filtering Based on Iterative Principal Component Analysis”, *Expert Systems with Applications* **28** (2005), 823–830.
 17. Koren, Y., “Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model”, *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.
 18. Koren, Y., “Collaborative Filtering with Temporal Dynamics.” *Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 447–456, 2009.
 19. Koren, Y., “Factor in the Neighbors: Scalable and Accurate Collaborative Filtering”, *ACM Transactions on Knowledge Discovery from Data (TKDD)*,4(2010):1-24.
 20. Linden, G., Smith, B., and York, J., “Amazon.com Recommendations: Item-to-Item Collaborative Filtering”, *IEEE Internet Computing* **7** (2003), 76–80.
 21. Marlin, B.M., Zemel, R.S., Roweis, S., and Slaney, M., “Collaborative Filtering and the Missing at Random Assumption”, *Proc. 23rd Conference on Uncertainty in Artificial Intelligence*, 2007.
 22. Oard, D.W., and Kim, J., “Implicit Feedback for Recommender Systems”, *Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31–36, 1998.
 23. Paterek, A., “Improving Regularized Singular Value Decomposition for Collaborative Filtering”, *Proc. KDD Cup and Workshop*, 2007.

24. Salakhutdinov, R., Mnih, A., and Hinton, G., “Restricted Boltzmann Machines for Collaborative Filtering”, *Proc. 24th Annual International Conference on Machine Learning*, pp. 791–798, 2007.
25. Salakhutdinov, R., and Mnih, A., “Probabilistic Matrix Factorization”, *Advances in Neural Information Processing Systems 20 (NIPS’07)*, pp. 1257–1264, 2008.
26. Sarwar, B.M., Karypis, G., Konstan, J.A., and Riedl, J., “Application of Dimensionality Reduction in Recommender System – A Case Study”, *WEBKDD’2000*.
27. Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., “Item-based Collaborative Filtering Recommendation Algorithms”, *Proc. 10th International Conference on the World Wide Web*, pp. 285–295, 2001.
28. Takács G., Pilászy I., Németh B. and Tikk, D., “Major Components of the Gravity Recommendation System”, *SIGKDD Explorations* **9** (2007), 80–84.
29. Takács G., Pilászy I., Németh B. and Tikk, D., “Matrix Factorization and Neighbor based Algorithms for the Netflix Prize Problem”, *Proc. 2nd ACM conference on Recommender Systems (RecSys’08)*, pp.267–274, 2008.
30. Tintarev, N., and Masthoff, J., “A Survey of Explanations in Recommender Systems”, *ICDE’07 Workshop on Recommender Systems and Intelligent User Interfaces*, 2007.
31. Toscher, A., Jahrer, M., and Legenstein, R., “Improved Neighborhood-Based Algorithms for Large-Scale Recommender Systems”, *KDD’08 Workshop on Large Scale Recommenders Systems and the Netflix Prize*, 2008.
32. Wang, J., de Vries, A.P., and Reinders, M.J.T., “Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion”, *Proc. 29th ACM SIGIR Conference on Information Retrieval*, pp. 501–508, 2006.

Chapter 6

Developing Constraint-based Recommenders

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach and Markus Zanker

6.1 Introduction

Traditional recommendation approaches (content-based filtering [48] and collaborative filtering[40]) are well-suited for the recommendation of quality&taste products such as books, movies, or news. However, especially in the context of products such as cars, computers, apartments, or financial services those approaches are not the best choice (see also Chapter 11). For example, apartments are not bought very frequently which makes it rather infeasible to collect numerous ratings for one specific item (exactly such ratings are required by collaborative recommendation algorithms). Furthermore, users of recommender applications would not be satisfied with recommendations based on years-old item preferences (exactly such preferences would be exploited in this context by content-based filtering algorithms).

Knowledge-based recommender technologies help to tackle these challenges by exploiting explicit user requirements and deep knowledge about the underlying product domain [11] for the calculation of recommendations. Those systems heavily concentrate on knowledge sources that are not exploited by collaborative filtering and content-based filtering approaches. Compared to collaborative filtering and content-based filtering, knowledge-based recommenders have no cold-start problems since requirements are directly elicited within a recommendation session. However, no advantage without disadvantage, knowledge-based recommenders suffer from the so-called knowledge acquisition bottleneck in the sense that knowledge

Alexander Felfernig (contact author)
Graz University of Technology e-mail: alexander.felfernig@ist.tugraz.at

Gerhard Friedrich
University Klagenfurt e-mail: gerhard.friedrich@uni-klu.ac.at

Dietmar Jannach
TU Dortmund e-mail: dietmar.jannach@tu-dortmund.de

Markus Zanker
University Klagenfurt e-mail: markus.zanker@uni-klu.ac.at

engineers must work hard to convert the knowledge possessed by domain experts into formal, executable representations.

There are two basic specifics of knowledge-based recommenders: case-based [3, 4, 36] and constraint-based recommenders [11, 13].¹ In terms of used knowledge both are similar: user requirements are collected, repairs for inconsistent requirements are automatically proposed in situations where no solutions could be found [12, 13, 43], and recommendation results are explained. The major difference lies in the way solutions are calculated [11]. Case-based recommenders determine recommendations on the basis of similarity metrics whereas constraint-based recommenders predominantly exploit predefined *recommender knowledge bases* that contain explicit rules about how to relate customer requirements with item features. In this chapter we will focus on an overview of constraint-based recommendation technologies. For a detailed review of case-based recommender technologies the reader is referred to [3, 4, 36].

A *recommender knowledge base* of a constraint-based recommender system (see [16]) typically is defined by two sets of variables (V_C , V_{PROD}) and three different sets of constraints (C_R , C_F , C_{PROD}). Those variables and constraints are the major ingredients of a constraint satisfaction problem [54]. A solution for a constraint satisfaction problem consists of concrete instantiations of the variables such that all the specified constraints are fulfilled (see Section 6.4).

Customer Properties V_C describe possible requirements of customers, i.e., requirements are instantiations of customer properties. In the domain of financial services *willingness to take risks* is an example for a customer property and *willingness to take risks = low* represents a concrete customer requirement.

Product Properties V_{PROD} describe the properties of a given product assortment. Examples for product properties are *recommended investment period*, *product type*, *product name*, or *expected return on investment*.

Constraints C_R are systematically restricting the possible instantiations of customer properties, for example, *short investment periods are incompatible with high risk investments*.

Filter Conditions C_F define the relationship between potential customer requirements and the given product assortment. An example for a filter condition is the following: *customers without experiences in the financial services domain should not receive recommendations which include high-risk products*.

Products Finally, allowed instantiations of product properties are represented by C_{PROD} . C_{PROD} represents one constraint in disjunctive normal form that defines elementary restrictions on the possible instantiations of variables in V_{PROD} .

A simplified recommender knowledge base for the domain of financial services is the following (see Example 6.1).

¹ Utility-based recommenders are often as well categorized as knowledge-based, see for example [4]. For a detailed discussion on utility-based approaches we refer the interested reader to [4, 13].

Example 6.1. Recommender knowledge base ($V_C, V_{PROD}, C_R, C_F, C_{PROD}$)

$V_C = \{kl_c: [\text{expert, average, beginner}] \dots \dots \dots /* level of expertise */$
 $wr_c: [\text{low, medium, high}] \dots \dots \dots /* willingness to take risks */$
 $id_c: [\text{shortterm, mediumterm, longterm}] \dots \dots \dots /* duration of investment */$
 $aw_c: [\text{yes, no}] \dots \dots \dots /* advisory wanted ? */$
 $ds_c: [\text{savings, bonds, stockfunds, singleshares}] \dots \dots /* direct product search */$
 $sl_c: [\text{savings, bonds}] \dots \dots \dots /* type of low-risk investment */$
 $av_c: [\text{yes, no}] \dots \dots \dots /* availability of funds */$
 $sh_c: [\text{stockfunds, singlshares}] \dots \dots \dots /* type of high-risk investment */ \}$

$V_{PROD} = \{name_p: [\text{text}] \dots \dots \dots /* name of the product */$
 $er_p: [1..40] \dots \dots \dots /* expected return rate */$
 $ri_p: [\text{low, medium, high}] \dots \dots \dots /* risk level */$
 $mniv_p: [1..14] \dots \dots \dots /* minimum investment period of product in years */$
 $inst_p: [\text{text}] \dots \dots \dots /* financial institute */ \}$

$C_R = \{CR_1: wr_c = \text{high} \rightarrow id_c \neq \text{shortterm},$
 $CR_2: kl_c = \text{beginner} \rightarrow wr_c \neq \text{high} \}$

$C_F = \{CF_1: id_c = \text{shortterm} \rightarrow mniv_p < 3,$
 $CF_2: id_c = \text{mediumterm} \rightarrow mniv_p \geq 3 \wedge mniv_p < 6,$
 $CF_3: id_c = \text{longterm} \rightarrow mniv_p \geq 6,$
 $CF_4: wr_c = \text{low} \rightarrow ri_p = \text{low},$
 $CF_5: wr_c = \text{medium} \rightarrow ri_p = \text{low} \vee ri_p = \text{medium},$
 $CF_6: wr_c = \text{high} \rightarrow ri_p = \text{low} \vee ri_p = \text{medium} \vee ri_p = \text{high},$
 $CF_7: kl_c = \text{beginner} \rightarrow ri_p \neq \text{high},$
 $CF_8: sl_c = \text{savings} \rightarrow name_p = \text{savings},$
 $CF_9: sl_c = \text{bonds} \rightarrow name_p = \text{bonds} \}$

$C_{PROD} = \{C_{PROD}_1: name_p = \text{savings} \wedge er_p = 3 \wedge ri_p = \text{low} \wedge mniv_p = 1 \wedge inst_p = A;$
 $C_{PROD}_2: name_p = \text{bonds} \wedge er_p = 5 \wedge ri_p = \text{medium} \wedge mniv_p = 5 \wedge inst_p = B;$
 $C_{PROD}_3: name_p = \text{equity} \wedge er_p = 9 \wedge ri_p = \text{high} \wedge mniv_p = 10 \wedge inst_p = B \}$

On the basis of such a recommender knowledge base and a given set of customer requirements we are able to calculate recommendations. The task of identifying a set of products fitting a customer's wishes and needs is denoted as *recommendation task* (see Definition 6.1).

Definition 6.1. A *recommendation task* can be defined as a constraint satisfaction problem ($V_C, V_{PROD}, C_C \cup C_F \cup C_R \cup C_{PROD}$) where V_C is a set of variables representing possible customer requirements and V_{PROD} is a set of variables describing product properties. C_{PROD} is a set of constraints describing product instances, C_R is a set of constraints describing possible combinations of customer requirements, and C_F (also called filter conditions) is a set of constraints describing the relationship

between customer requirements and product properties. Finally, C_C is a set of unary constraints representing concrete customer requirements.

Example 6.2. Based on the recommender knowledge base of Example 6.1, the definition of a concrete recommendation task could be completed with the following set of requirements $C_C = \{wr_c = low, kl_c = beginner, id_c = shortterm, sl_c = savings\}$.

Based on the definition of a recommendation task, we can introduce the notion of a solution (*consistent recommendation*) for a recommendation task.

Definition 6.2. An assignment of the variables in V_C and V_{PROD} is denoted as *consistent recommendation* for a recommendation task $(V_C, V_{PROD}, C_C \cup C_F \cup C_R \cup C_{PROP})$ iff it does not violate any of the constraints in $C_C \cup C_F \cup C_R \cup C_{PROD}$.

Example 6.3. A consistent recommendation with regard to the recommender knowledge base of Example 6.1 and the customer requirements defined in Example 6.2 is $kl_c = beginner, wr_c = low, id_c = shortterm, sl_c = savings, name_p = savings, er_p = 3, ri_p = low, mniv_p = 1, inst_p = A$.

In addition to the recommender knowledge base we have to define the intended behavior of the recommender user interface. In order to support intuitive dialogs, a recommender interface must be adaptive (see Section 3). There exist different alternatives to describe the intended behavior of recommender user interfaces. For example, dialogs can be modeled explicitly in the form of finite state models [20] or can be structured even more flexibly in a form where users themselves are enabled to select interesting properties they would like to specify [37].

In this chapter we will focus on the first alternative: recommendation dialogs are modeled explicitly in the form in finite state models [20]. Transitions between the states are represented as acceptance criteria on the user input. For example, an expert ($kl_c = expert$) who is not interested in a recommendation session regarding financial services ($aw_c = no$) is automatically forwarded to q_4 (search interface that supports the specification of technical product features). Figure 6.1 depicts a finite state model of the intended behavior of a financial services recommender application.

The remainder of this chapter is organized as follows. In Section 6.2 we give an overview of knowledge acquisition concepts for the development of recommender knowledge bases and recommender process definitions. In Section 6.3 we introduce major techniques for guiding and actively supporting the user in a recommendation dialog. A short overview of approaches to solve recommendation tasks is given in Section 6.4. In Section 6.5 we discuss successful applications of constraint-based recommender technologies. In Section 6.6 we present future research issues in constraint-based recommendation. With Section 6.7 we conclude the chapter.

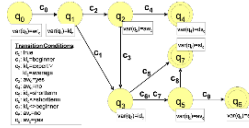


Fig. 6.1: Recommender user interface description: a simple example recommendation process for financial services. The process starts in state q_0 , and, depending on the user’s knowledge level, is forwarded to either state q_2 or state q_3 . In the final state (one of the states q_4, q_6, q_7) the recommended items are presented. Each state q_i has an assigned customer property $var(q_i)$ that represents a question to be asked in this state.

6.2 Development of Recommender Knowledge Bases

The major precondition for successfully applying constraint-based technologies in commercial settings are technologies that actively support knowledge engineers and domain experts in the development and maintenance of recommender applications and thus help to limit knowledge acquisition bottlenecks as much as possible. Due to very limited programming skills of domain experts, there typically is a discrepancy between knowledge engineers and domain experts in terms of knowledge base development and maintenance know-how [13]. Thus domain experts are solely responsible for knowledge provision but not for the formalization into a corresponding executable representation (recommender knowledge base).

The major goal of the commercially available *CWAdvisor* environment presented in [13] is to reduce the above mentioned knowledge acquisition bottleneck: it supports autonomous knowledge base development and maintenance processes for domain experts. In the following sections we will present parts of the *CWAdvisor* environment for demonstration purposes. The *CWAdvisor* knowledge acquisition environment (*CWAdvisor Designer*) takes into account major *design principles* that are crucial for effective knowledge acquisition and maintenance [8, 13].

- First, *rapid prototyping* processes support the principle of *concreteness* where the user can immediately inspect the effects of introduced changes to explanation texts, properties of products, images, recommender process definitions, and recommendation rules. This functionality is implemented in the form of templates that enable a direct translation of graphically defined model properties into a corresponding executable recommender application.
- Second, changes to all the mentioned information units can be performed on a graphical level. This functionality is very important to make knowledge acquisition environments more accessible to domain experts without a well-grounded technical education. Domain experts are protected from programming details - an approach that follows the principle of a strict *separation of application logic and implementation details*.

- Third, an integrated testing and debugging environment supports the principle of *immediate feedback* in the sense that erroneous definitions in the recommender knowledge base and the recommendation process are automatically detected and reported (end-user debugging support). Thus, knowledge bases are maintained in a structured way and not deployed in a productive environment until all test cases specified for the knowledge base are fulfilled. As a direct consequence, domain experts develop a higher trust level since erroneous recommendations become the exception of the rule.

Figure 6.2 provides examples for major modeling concepts supported by the *CWAdvisor* recommender development environment [13]. This environment can be used for the design of a recommender knowledge base (see Example 6.2), i.e., customer properties (V_C), product properties (V_{PROD}), constraints (C_R), filter conditions (C_F), and the product assortment (C_{Prod}) can be specified on a graphical level. Figure 6.2 (upper part) depicts an interface for the design of filter conditions (C_F) whereas the lower part represents an interface for the context-oriented specification of compatibility constraints. Figure 6.3 shows the *CWAdvisor* Process Designer user interface. This component enables the graphical design of recommendation processes. Given such a process definition, the recommender application can be automatically generated (see, e.g., Figure 6.4).

Sometimes recommendation processes are faulty, for example, the transition conditions between the states are defined in a way that does not allow the successful completion of a recommendation session. If we would change the transition condition $c_1 : kl_c = \text{beginner}$ in Figure 6.1 to $c'_1 : kl_c = \text{expert}$, users who have nearly no knowledge about the financial services domain would not be forwarded to any of the following states (q_2 or q_3). For more complex process definitions, the manual identification and repair of such faults is tedious and error-prone. In [20] an approach is presented which helps to automatically detect and repair such faulty statements. It is based on the concepts of model-based diagnosis [20] that help to locate minimal sets of faulty transition conditions.

In addition to a graphical process definition, *CWAdvisor* Designer supports the automated generation of test cases (input sequences including recommended products) [16]. On the one hand, such test cases can be exploited for the purpose of regression testing, for example, before the recommender application is deployed in the production environment. On the other hand, test cases can be used for debugging faulty recommender knowledge bases (if some of the test cases are not fulfilled) and faulty process definitions (e.g., when the recommender process gets stuck).

The basic principle of recommender knowledge base debugging [10, 12, 13, 16] will now be shown on the basis of Example 6.4.² Readers interested in the automated debugging of faulty recommender process definitions are referred to [20]. A typical approach to identify faults in a recommender knowledge base is to test the knowledge base with a set of examples (test cases) $e_i \in E$. For simplicity, let us assume that $e_1 : wr_c = \text{high} \wedge rr_c \geq 9\%$ is the only example provided by domain experts up to now. Testing $e_1 \cup C_R$ results in the empty solution set due to the fact

² For simplicity, we omit the specification of V_{PROD} , C_F , and C_{PROD} .

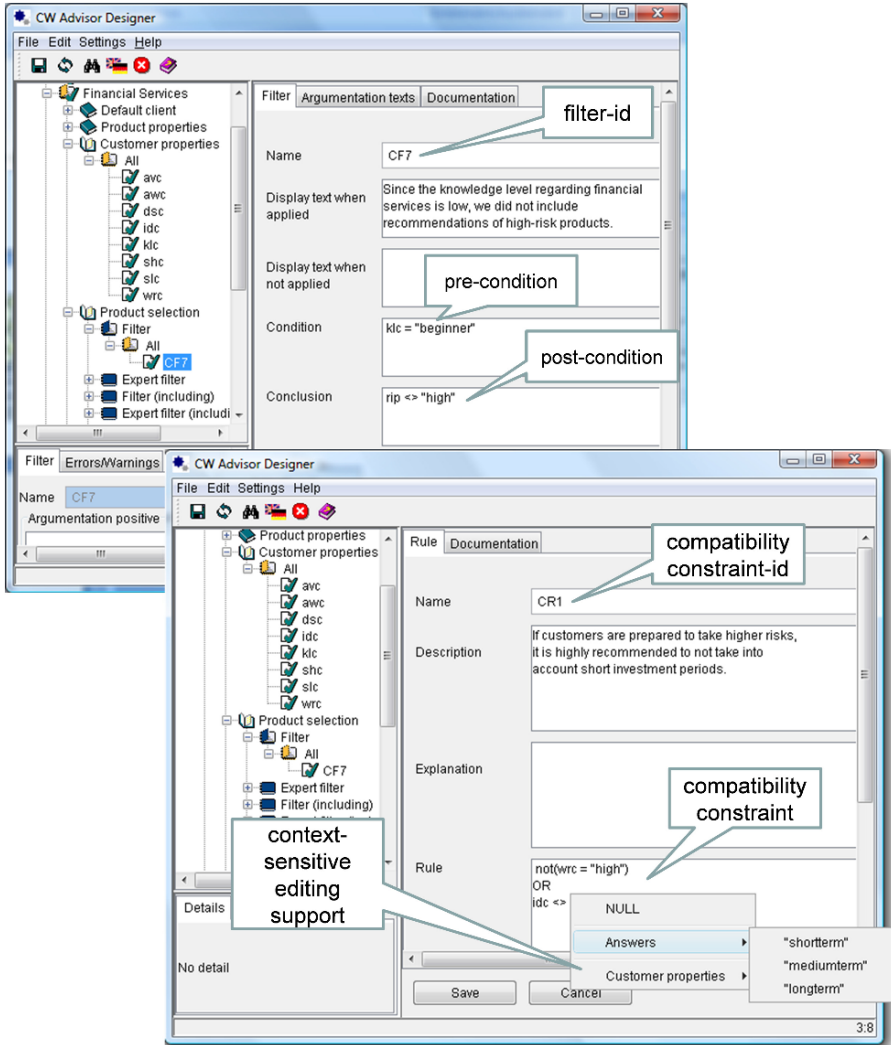


Fig. 6.2: CWAdvisor Designer Environment. Filter constraints (conditions) as well as compatibility constraints can be defined in a context-sensitive editing environment.

that e_1 is inconsistent with C_R . A more detailed look at the example shows that the constraints CR_2, CR_3 are inconsistent with e_1 . CR_2, CR_3 is denoted as *conflict set* [33, 45] that can be resolved (under the minimality assumption) by simply deleting one of its elements. For example, if we delete CR_3 from C_R , the consistency of $e_1 \cup C_R$ is restored. The calculation of conflict sets can be realized using the conflict detection algorithm proposed by [33], the automated resolution of conflicts is shown in detail in [20].

Example 6.4. Faulty Recommender knowledge base ($V_C, V_{PROD}, C_R, C_F, C_{PROD}$)

$V_C = \{rr_c: [1-3\%, 4-6\%, 7-9\%, 9\%] \dots\dots\dots /* \text{return rate} */$
 $wr_c: [low, medium, high] \dots\dots\dots /* \text{willingness to take risks} */$
 $id_c: [shortterm, mediumterm, longterm] \dots\dots\dots /* \text{duration of investment} */ \}$

$C_R = \{CR_1: wr_c = medium \rightarrow id_c \neq shortterm$
 $CR_2: wr_c = high \rightarrow id_c = long$
 $CR_3: id_c = long \rightarrow rr_c = 4 - 6\% \vee rr_c = 7 - 9\%$
 $CR_4: rr_c \geq 9\% \rightarrow wr_c = high$
 $CR_5: rr_c = 7 - 9\% \rightarrow wr_c \neq low \}$

$V_{PROD} = \{ \} \quad C_F = \{ \} \quad C_{PROD} = \{ \}$

Experiences from commercial projects in domains such as financial services [18], electronic equipments [13], or e-tourism [74] clearly point out the importance of the above mentioned principles regarding the design of knowledge acquisition and maintenance environments. Within the scope of user studies [10] significant time savings in development and maintenance processes have been detected due to the availability of a graphical development, test, and automated debugging environment. Experiences from the financial services domain [18] show that initially knowledge bases have to be developed within the scope of a cooperation between domain experts and technical experts (knowledge engineers). Thereafter, most development and maintenance requests are directly processed by the domain experts (e.g., updates in product tables, adaptations of constraints, or recommender process definitions).

6.3 User Guidance in Recommendation Processes

As constraint-based recommender systems operate on the basis of explicit statements about the current customer’s needs and wishes, the knowledge about these user requirements has to be made available to the system before recommendations can be made. The general options for such a *requirements elicitation process* in increasing order of implementation complexity include the following.

1. Session-independent customer profiles: users enter their preferences and interests in their user profile by, for example, specifying their general areas of inter-

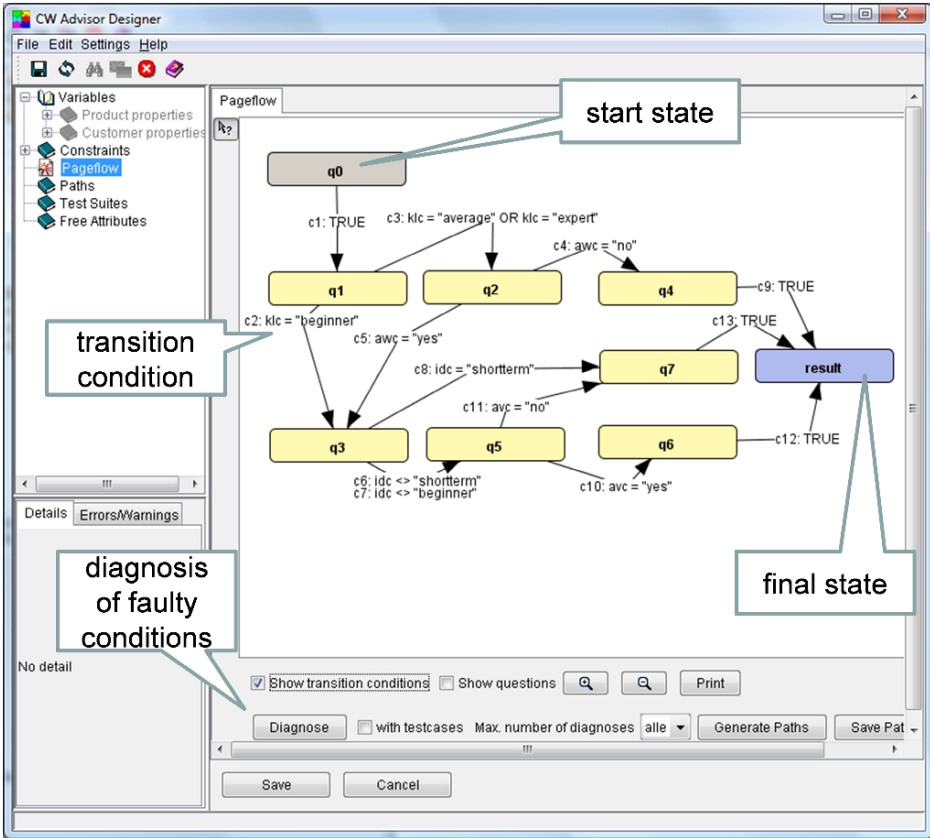


Fig. 6.3: CWAdvisor Designer Environment. Recommendation processes are specified on a graphical level and can be automatically translated into a corresponding executable representation. Faulty transition conditions can be identified automatically on the basis of model-based diagnosis [20].

est (see also Chapter 22). This is a common approach in web portals or social networking platforms.

2. Static fill-out forms per session: customers fill out a static web-based form every time they use the recommender system. Such interfaces are easy to implement and web users are well-acquainted with such interfaces, which are often used on web shops search for items.
3. Conversational recommendation dialogs: the recommender system incrementally acquires the user’s preferences in an interactive dialog, based on, for example, “critiquing” [8] (see also Chapter 13), “wizard-like“ and form-based preference elicitation dialogs [13], natural-language interaction [59] or a combination of these techniques.

In the context of constraint-based recommendation, particularly this last type of preference elicitation plays an important role and will be in the focus of this chapter, because recommendation in complex domains such as financial services [18] or electronic consumer goods [25] often induces a significant cognitive load on the end user interacting with the system. Thus, adequate user interfaces are required to make sure that the system is usable for a broad community of online users.

Of course, the static information available in some user-specified customer profile can also be an input source for a constraint-based recommender. The integration of such general profile information (including particularly demographic information) into the recommendation process is straightforward. In many cases, however, this information is rather unspecific and broad so that the utility of these information pieces is limited for an in-detail knowledge-based recommendation process.

Static fill-out forms for some applications work well for the above-mentioned reasons. However, in knowledge-intensive domains, for which constraint-based recommenders are often built, this approach might be too simplistic, particularly because the online user community can be heterogeneous with respect to their technical background, so that it is inappropriate to ask all users the same set of questions or at the same level of technical detail [25].

Finally, we will also not focus on natural language interaction in this chapter as only few examples such as [59] exist, that use a (complementing) natural language recommender system user interface. Despite the advances in the field of Natural Language Processing and although human-like virtual advisors can be found as an add-on to different web sites, they are barely used for recommending items to users today, for which there are different reasons. First, such dialogs are often user-driven, i.e., the user is expected to actively ask questions. In complex domains, however, in particular novice users are not capable of formulating such questions about, for example, the right medium-term investment strategy. In addition, the knowledge-acquisition effort for such systems is relatively high, as the system should also be capable of conducting casual conversation. Finally, end users often attribute more intelligence to such human-like avatars than is warranted which carries the risk of leaving them disappointed after interacting with the system.

Critiquing Critiquing is a popular interaction style for knowledge-based recommender systems, which was first proposed in [6] in the context of *Case-Based Reasoning* (CBR) approaches to conversational recommendation. The idea is to present individual items (instances), for example, digital cameras or financial products, to the user who can then interactively give feedback in terms of critiques on individual features. A user might, for instance, ask for a financial product with a “shorter investment period” or a “lower risk”. This recommend-review-revise cycle is repeated until the desired item is found. Note that although this method was developed for CBR recommendation approaches³, it can also be applied to constraint-based recommendation, as the critiques can be directly translated into additional constraints that reflect the user’s directional preferences on some feature.

³ The general idea of exploring a database by criticizing successive examples is in fact much older and was already proposed in the early 1980s in an information retrieval context [55].

When compared with detailed search forms that can be found on many online shops, the critiquing interaction style has the advantage that it supports the user in interactively exploring the item space. Moreover, the approach, which is often also called *tweaking*, is relatively easy to understand also for novice users. Developing a critiquing application, however, requires some domain knowledge, for example, about the set of features the user can give feedback, suitable increment values for number-valued attributes or logical orderings of attributes with enumeration domains. In addition, when mappings from customer needs to product features are needed, additional engineering effort is required.

The basic critiquing scheme was later on extended to also support *compound critiques* [44, 52], where users can give feedback on several features in a single interaction cycle. In the domain of financial services, a user could therefore ask for a product that has lower risk and a longer investment horizon in one step, thus decreasing the number of required interaction cycles. While some sort of pre-designed compound critiques were already possible in the initial proposal from [6], it is argued in [44] that the set of possible critiques should be dynamically determined depending on the remaining items in the current user's item space and in particular on the level of variation among these remaining items. The results of experimental evaluations show that such compound critiques can help to significantly reduce the number of required interaction cycles, thus making the whole interaction process more efficient. In addition, the experiments indicate that compound critiques (if limited to a size that is still understandable to the user) can also help the user understand the logic of the recommendations generated by the system.

Recent developments in critiquing include the use of elaborate visual interfaces [62], the application of the approach in mobile recommender systems [74], or the evaluation of critiquing styles regarding decision accuracy and cognitive effort [20].

Personalized preference elicitation dialogs Another form of acquiring the user's wishes and needs for a constraint-based recommender system is to rely on explicitly modeled and adaptive preference elicitation dialogs. Such dialog models can for instance be expressed using a *dialog grammar* [2] or by using finite-state automaton as done in the *CWAdvisor* system [20, 13].

In the later system, the end user is guided by a "virtual advisor" through a series of questions about the particular needs and requirements before a recommendation is displayed, see Figure 6.4 for an example dialog. In contrast to static fill-out forms, the set of questions is personalized, i.e., depending on the current situation and previous user answers, a different set of questions (probably also using a different technical or non-technical language [29]) will be asked by the system.

In the *CWAdvisor* system, the required user interface adaptation is based on manually-engineered personalization rules and on an explicit dialog model in the form of a finite-state automaton as shown in Figure 6.1. Thus, a method is chosen that represents a compromise between fill-out forms to which web users are well-acquainted and fully free conversation as provided by approaches based on Natural Language Processing.

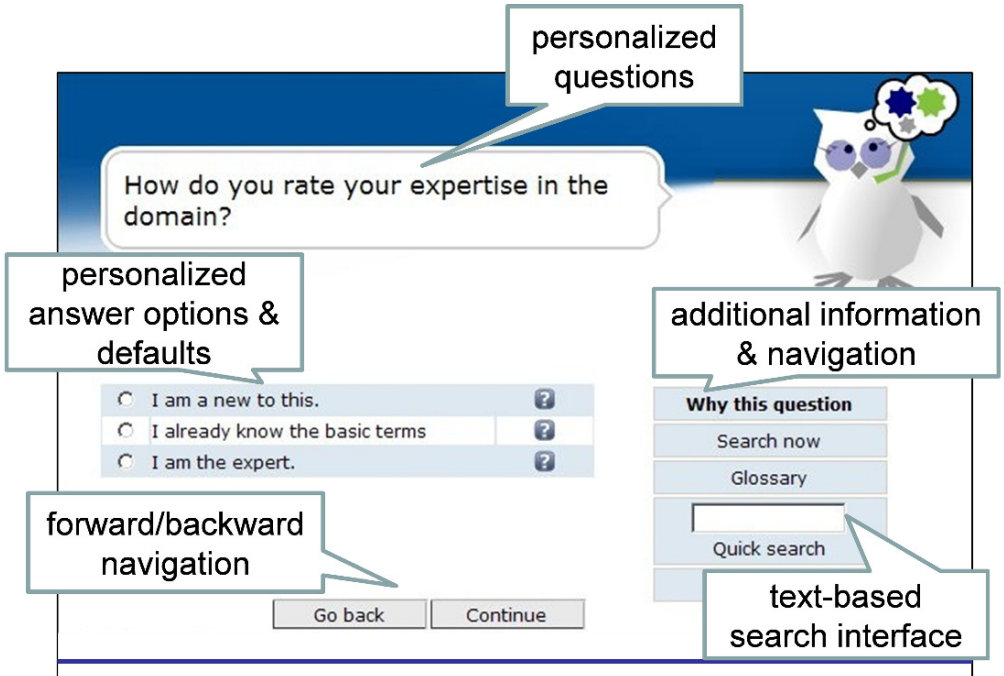


Fig. 6.4: Interactive and personalized preference elicitation example. Customers specify their preferences by answering questions.

Technically, the vertices of the finite-state automaton in Figure 6.1 are annotated with logical expressions over the constraint variables that are used to capture the user requirements. The process of developing the dialog and personalization model is supported in the *CWAdvisor* system by an end-user oriented graphical process modeling editor. At run time, the interaction-handling component of the framework collects the user inputs and evaluates the transition conditions in order to decide how to continue the dialog, see [13] for more details.

Beside the personalization of the dialog, different other forms of adaptation on the level of content, interaction and presentation are implemented in the system [30] in order to support the design of preference elicitation and explanation dialogs that support the end user in the best possible way.

While highly-dynamic and adaptive web applications can be valuable in terms of ease-of-use and user experience, the technical realization and in particular the maintenance of such flexible user interfaces for a constraint-based recommender can be challenging. The main problem in this context are the strong interrelationships between the “model”, the “view” and the control logic of such applications: consider, for instance, the situation, where the dialog model should be extended with a new question (variable), a new answer option (new variable domain), or whole dialog page (new dialog automaton state). In each case, the web pages that represent

the “view” of the recommender application, have to be adapted accordingly. Therefore, toolkits for developing personalized preference elicitation processes, have to provide mechanisms to at least partially automate the process of updating the user interface, see [30] for details of the template-based approach in *CWAdvisor*.

Dealing with unfulfillable or too loose user requirements The issue of the development of the user interface is not the only challenging problem in the context of personalized preference elicitation in constraint-based recommenders. In the following, we will sketch further aspects that have to be dealt with in practical applications of this technology (see also Chapter 15 and Chapter 16).

In constraint-based recommenders, the situation can easily arise that no item in the catalog fulfills all the constraints of the user. During an interactive recommendation session, a message such as “no matching product found” is however highly undesirable. The question therefore arises, how to deal with such a situation that can also occur in CBR-based recommenders that in many cases at least initially rely on some query mechanism to retrieve an initial set of cases from the product catalog (case base). One possible approach proposed in the context of CBR-based recommenders is based on *query relaxation* [39, 43, 47, 23, 27]. In the context of CBR recommenders, the set of recommendable items are conceptually stored in a database table; the case retrieval process consists of sending a conjunctive query Q (of user requirements) to this case base. Query relaxation then refers to finding a (maximal) subquery Q' of the original query Q that returns at least one item.

The general idea of query relaxation techniques can also be applied in constraint-based recommendation. Consider Example 6.5 (adapted from [27]), where the catalog of four items C_{PROD} is shown in tabular form in Figure 6.5.

$name_p$	sl_p (type of low risk inv.)	ri_p (associated risk)	$mniv_p$ (min. investment period)	er_p (expected return)	$inst_p$ (financial institute)
p1	stockfunds	medium	4	5 %	ABank
p2	singleshares	high	3	5 %	ABank
p3	stockfunds	medium	2	4 %	BInvest
p4	singleshares	high	4	5 %	CMutual

Fig. 6.5: Example item catalog (financial services).

Example 6.5. Query Relaxation

For sake of clarity and simplicity of the example, let us assume that the customer can directly specify the desired properties of the investment product on an “expert screen” of the advisory application. The set of corresponding customer properties V_c thus contains sl_c (investment type), ri_c (risk class), $minimum_return_c$ (minimum value for expected return) and $investment_duration_c$ (desired investment duration). The filter constraints (conditions) in this example simple map customer requirements from C_c to item features, i.e., $CF_1 : sl_c = sl_p$, $CF_2 : ri_c = ri_p$, $CF_3 : investment_duration_c \geq mniv_p$, $CF_4 : er_p \geq minimum_return_c$

Let the concrete customer requirements C_C be as follows: $\{sl_c = \textit{singleshares}, ri_c = \textit{medium}, \textit{investment_duration}_c = 3, \textit{minimum_return}_c = 5\}$.

As can be easily seen, no item in the catalog (see Figure 6.5) fulfills all relevant constraints in the given task, i.e., no consistent recommendation can be found for the recommendation task. When following a “constraint relaxation“ approach, the goal now consists of finding a maximal subset of the constraints of C_F , for which a recommendation can be found. The maximization criterion is typically chosen because the constraints directly relate to customer requirements, i.e., the more constraints can be retained, the better the retrieved items will match these requirements.

While this problem of finding consistency-establishing subsets of C_F seems to be not too complex at a first glance, in practical settings, computational effectiveness becomes an issue. Given a constraint base consisting of n constraints, the number of possible subsets is 2^n . Since real-world recommender systems have to serve many users in parallel and typically the acceptable response time is about one second, naive subset probing is not appropriate.

Different techniques have therefore been proposed to solve this problem more efficiently. In [39], for instance, an incremental mixed-initiative to recovery from failing queries in a CBR recommender was suggested. In [47], a relaxation method based on manually-defined feature hierarchies was proposed, which despite its incomplete nature has shown to be an effective help in a travel recommender system. Finally, in [27] and [26] a set of complete algorithms for the query relaxation problem in constraint-based recommenders was developed. The algorithms not only support the computation of minimum relaxations in linear time (at the cost of a preprocessing step and slightly increased memory requirements) but also the computation of relaxations that lead to “at least n ” remaining items. In addition, also a conflict-directed algorithm for interactive and incremental query relaxation was proposed which makes use of recent conflict-detection technology [33].

The main idea of the linear-time constraint relaxation technique can be sketched as follows. Instead of testing combinations of constraints, the relevant constraints are evaluated individually, resulting in a data structure that assigns to every constraint the list of catalog items that fulfill the constraint, see Figure 6.6.

ID	Product p1	Product p2	Product p3	Product p4
CF_1	0	1	0	1
CF_2	1	0	1	0
CF_3	0	1	1	0
CF_4	1	1	0	1

Fig. 6.6: Evaluating the subqueries individually. For example, product p1 is filtered out by the filter condition CF_1 under the assumption that $sl_c = \textit{singleshares}$.

The table should be interpreted as follows. Constraint CF_1 on the type of investment (single shares) in line 1 of the table would filter out products $p1$ and $p3$.

Given this table, it can be easily determined, which constraints of a given set C_F have to be relaxed to have a certain product in the result set, i.e., consistent with the constraints and the user requirements. For example, in order to have p_1 in the result set, the constraints CF_1 and CF_3 of C_F have to be relaxed. Let us call this a “product-specific relaxation” for p_1 . The main idea of the method from [27] is that the overall “best” relaxation for given products C_{PROD} , filter conditions C_F and a given set of concrete requirements C_C has to be among the product-specific relaxations. Thus, it is sufficient to scan the set of product-specific relaxations, i.e., no further constraint solving step is required in this phase.

In the example, the relaxation of constraint CF_2 is optimal, when the number of relaxed constraints determines the best choice as only one customer requirement has to be given up. All other relaxations require at least two constraints to be ignored, which can be simply determined by counting the number of zeros in each column. Note that the number of involved constraints is only one possible optimization criterion. Other optimization criteria that take additional “costs of compromise” per constraint into account can also be implemented based on this technique as long as the cost function’s value is monotonically increasing with the size of the relaxation.

Technically, the computation of product-specific relaxations can be done very efficiently based on bit-set operations [27]. In addition, the computation can partially also be done in advance in the start-up phase of the recommender.

Suggesting alternatives for unfulfillable requirements In some application domains, the automated or interactive relaxation of individual constraints alone may be not suffice as a means to help the user out of a situation, in which his or her requirements cannot be fulfilled. Consider, for instance, a situation where the recommender in an interactive relaxation scenario proposes a set of alternatives of constraints to be relaxed. Let us assume that the user accepts one of the proposals, i.e., agrees to relax the constraints related to two variables of V_C , for example, A and B . If, however, the values of A and B are particularly important to him (or mandatory), he will later on put different constraints on these variables. These new values can, however, again cause an inconsistency with the other requirements of the user. This might finally lead to an undesirable situation, in which the user ends up in trying out different values but gets no clear advise, which values to take to receive a consistent recommendation.

Overall, it would be thus desirable, if the system could immediately come up with suggestions for new values for A and B , for which it is guaranteed that some items remain in the result set when the user’s other requirements are also further taken into account.

Let us first consider the basic CBR-style case retrieval problem setting as used in [39, 43, 47], in which constraints are directly placed on item features. The constraints in this example shall be $\{sl_p = \text{singleshares}, ri_p = \text{medium}, minv_p < 3, er_p \geq 5\}$. Again, no item fulfills these requirements.

In such a setting, the detailed information about the catalog items can be used to compute a set of suggestions for alternative constraints (“repairs”) on individual features. Based on this information, the system could – instead of only proposing the

user to relax the constraints on the investment type and on the investment duration – inform the user that “if the single shares requirement is abandoned and the minimum investment duration is set to 4” one or more items will be found. Thus, the user will be prevented from (unsuccessfully) trying a minimum investment duration of 3.

In this example, the calculation of such alternative values can be accomplished by the system by choosing one relaxation alternative (investment duration and investment type) and searching the catalog for items that fulfill the remaining constraints. The values for the investment duration and the investment type (e.g., of product 1 in Figure 6.5) can be directly taken as suggestions for the end user [19] [14].

While this approach seems intuitive and simple, in practical applications the following problems have to be dealt with.

- The number of possible repairs. In realistic scenarios, the number of possible repair alternatives is typically very large as for every possible relaxation – there might be already many of them – various solutions exist. In practice, however, end users cannot be confronted with more than a few overall alternatives. Thus, the problem exists to select and prioritize the repair alternatives.
- The size/length of the repair proposals. Repair suggestions that contain alternative values for more than three features are not easy to understand for end users.
- Computational complexity for non-trivial constraints. When only simple constraints on product features are allowed, the information from the item catalog can help to determine possible repairs as described above. In constraint-based systems such as *CWAdvisor*, however, the definition of constraints that relate often qualitative user needs to (technical) product features is possible. Consequently, also the repair suggestions must relate to user requirements, which means that the search space of possible repair alternatives is determined by the domains of the user-related variables. In addition, determining whether or not a specific combination of user requirements (i.e., a repair alternative) leads to a non-empty result set, requires a probably costly catalog query.

In order to address these issues at least to some extent, the *CWAdvisor* system uses a combination of query relaxation and different search heuristics and additional domain-specific knowledge for the calculation of repair suggestions in a financial services application [17].

The method implemented in the system interleaves the search for relaxations with a bounded search for repair alternatives. The possible relaxations are determined in increasing order of their cardinality. For each relaxation, repair alternatives are determined by varying the values of the variables that are involved in the relaxed constraints. The selection of alternative values can for instance be guided by a “nearness” heuristic that is based on an externally or implicitly defined order of the values. Thus, when varying for instance a user requirement of “at least 5 % expected return”, the neighboring value of “4 %” is evaluated, assuming that such an alternative will be more acceptable for the end user than an even stronger relaxation. In order to avoid too many similar repair suggestions, the algorithm can be parameterized with several threshold values that, for example, determine the number

of repairs for a relaxation, the maximum size of a relaxation and so forth. Overall, anecdotal evidence in the financial service domain indicates that such a repair feature, even if it is based on heuristics, is well-appreciated by end users as a means for shortening the required dialog length.

Query tightening Beside having no item in the result set, having *too many* items in the result set is also not desirable in an interactive recommender. In many real-world applications the user is informed that “too many items have been found” and that more precise search constraints have to be specified. Often, only the first few results are displayed (as to, e.g., avoid long page loading times). Such a selection may however not be optimal for the current user, since the selection is often simply based on the alphabetic order of the catalog entries.

In order to better support the user also in this situation, in [48] an *Interactive Query Management* approach for CBR recommenders is proposed, that also includes techniques for “query tightening”. The proposed tightening algorithm takes as an input a query Q and its large result set and selects – on the basis of information-theoretic considerations and the entropy measure – three features that are presented to the user as proposals to refine the query.

Overall, an evaluation of *Interactive Query Management* within a travel recommender system that implemented both query relaxation and query tightening [50], revealed that the relaxation feature was well-appreciated by end users. With respect to the tightening functionality, the evaluation indicated that query tightening was not that important to end users who were well capable of refining their queries by themselves. Thus, in [41] a different feature selection method was proposed, that also takes a probabilistic model of feature popularity into account. An evaluation showed that in certain situations the method of [41] is preferable since it is better accepted by end users as a means to further refine their queries.

6.4 Calculating Recommendations

Following our characterization of a recommendation task (see Definition 1), we will now discuss corresponding problem solving approaches. Typical approaches to solve a recommendation task are *constraint satisfaction algorithms* [54] and *conjunctive database queries* [46].

Constraint Satisfaction Solutions for *constraint satisfaction problems* are calculated on the basis of search algorithms that use different combinations of *backtracking* and *constraint propagation* - the basic principle of both concepts will be explained in the following.

Backtracking. In each step, backtracking chooses a variable and assigns all the possible values to this variable. It checks the consistency of the assignment with the already existing assignments and defined set of constraints. If all the possible values of the current variable are inconsistent with the existing assignments and the constraints, the constraint solver backtracks which means that the previously

instantiated variable is selected again. In the case that a consistent assignment has been identified, a recursive activation of the backtracking algorithm is performed and the next variable is selected [54].

Constraint Propagation. The major disadvantage of pure backtracking-based search is "trashing" where parts of the search space are revisited although no solution exists in these parts. In order to make constraint solving more efficient, constraint propagation techniques have been introduced. These techniques try to modify an existing constraint satisfaction problem such that the search space can be reduced significantly. The methods try to create a state of *local consistency* that guarantees consistent instantiations among groups of variables. The mentioned modification steps turn an existing constraint satisfaction problem into an equivalent one. A well known type of local consistency is *arc consistency* [54] which states that for two variables X and Y there must not exist a value in the domain of Y which does not have a corresponding consistent value in X. Thus, arc consistency is a directed concept which means that if X is arc consistent with Y, the reverse must not necessarily be the case.

When using a constraint solver, constraints are typically represented in the form of expressions of the corresponding programming language. Many of the existing constraint solvers are implemented on the basis of Java (see, for example, *jacop.osolpro.com*).

Conjunctive Database Queries Solutions to *conjunctive queries* are calculated on the basis of database queries that try to retrieve items which fulfill all of the defined customer requirements. For details on the database technologies and the execution of queries on database tables see, for example, [46].

Ranking Items Given a recommendation task, both constraint solvers and database engines try to identify a set of items that fulfill the given customer requirements. Typically, we have to deal with situations where more than one item is part of a recommendation result. In such situations the items (products) in the result set have to be ranked. In both cases (constraint solvers and database engines), we can apply the concepts of multi-attribute utility theory (MAUT) [56] that helps to determine a ranking for each of the items in the result set. Examples for the application of MAUT can be found in [13].

An alternative to the application of *MAUT in combination with conjunctive queries* are *probabilistic databases* [35] which allow a direct specification of ranking criteria within a query. Example 6.6 shows such a query which selects all products that fulfill the criteria in the WHERE clause and orders the result conform to a similarity metric (defined in the ORDER BY clause). Finally, instead of combining the mentioned *standard constraint solvers with MAUT*, we can represent a recommendation task in the form of soft constraints where the importance (preference) for each combination of variable values is determined on the basis of a corresponding utility operation (for details see, for example, [1]).

Example 6.6. Queries in probabilistic databases

```
Result = SELECT *      /* calculate a solution */
FROM Products        /* select items from "Products" */
```



```

WHERE  $x_1=a_1$  and  $x_2=a_2$  /* "must" criteria */
ORDER BY score(abs( $x_3-a_3$ ), ..., abs( $x_m-a_m$ )) /* similarity-based utility function */
STOP AFTER N; /* at most N items in the solution (result set) */

```

6.5 Experiences from Projects and Case Studies

The *CWAdvisor* system has been commercialized in 2002 and since then more than 35 different applications have been instantiated and fielded. They have been applied in commercial domains ranging from financial services [17] to electronic consumer goods or tourism applications [32] as well as to application domains that are considered rather untypical for recommender systems such as providing counseling services on business plans [28] or supporting software engineers in selecting appropriate software estimation methods [43].

Based on this installation base different forms of *empirical research* have been conducted that try to assess the impact and business value of knowledge based recommender systems as well as to identify opportunities for advancing their state-of-the-art. In the following we will differentiate them based on their study design into *user studies*, *evaluations on historical data* and *case studies of productive systems*.

Experimental user studies simulate real user interactions and research the acceptance or rejection of different hypotheses. [15] conducted a study to evaluate the impact of specific functionalities of conversational knowledge-based recommenders like explanations, proposed repair actions or product comparisons. The study assigned users randomly to different versions of the recommender system that varied the functionalities and applied pre- and post-interaction surveys to identify users' level of knowledge in the domain, their trust in the system or the perceived competence of the recommender. Quite interestingly, the study showed that study participants appreciate these specific functionalities as they increase their perceived level of knowledge in the domain and their trust in the system's recommendations.

The COHAVE project initiated a line of research that investigated how psychological theories might be applied to explain users' behavior in online choice situations. For instance, asymmetric dominance effects arise if proposed itemsets contain decoy products that are dominated by other products due to their relative similarity but a lower overall utility. Several user studies in domains such as electronic consumer goods, tourism and financial services showed, that knowing about these effects a recommender can increase the conversion rate of some specific items as well as a users confidence in the buying decision.

Algorithm evaluations on historical datasets are off-line experimentations [25]. A dataset that contains past user transactions is split into a training and testing set. Consequently, the training set is exploited to learn a model or tune algorithm's parameters in order to enable the recommender to predict the historic outcomes of the user sessions contained in the testing set. Such an evaluation scenario enables comparative research on algorithm performance. While collaborative and content-

based recommendation paradigms have been extensively evaluated in the literature, comparing knowledge-based recommendation algorithms with other recommendation paradigms received only few attention in the past. One reason is that they are hard to compare, because they require different types of algorithm input: collaborative filtering typically exploits user ratings while constraint-based recommender systems require explicit user requirements, catalog data and domain knowledge. Consequently, datasets that contain all these types of input data - like the *Entree* dataset provided by Burke [14] - would allow such comparisons, however they are very rare. One of the few is described in [61]. The dataset stems from a retailer offering premium cigars and includes implicit ratings signifying users' purchase actions, users' requirements input to a conversational recommender and a product catalog with detailed item descriptions. Therefore, offline experiments compared knowledge-based algorithm variants that exploited user requirements with content-based and collaborative algorithms working on ratings. One of the interesting results were that knowledge-based recommenders did not perform worse in terms of serendipity measured by the catalog coverage metric than collaborative filtering. This is especially true if a constraint-based recommender is cascaded with a utility-based item ranking scheme like the *CWAdvisor* system. However, collaborative filtering does better in terms of accuracy, if there are 10 and more ratings known from users. Nevertheless, an evaluation of a knowledge-based recommender always measures the quality of the encoded knowledge base *and* the inferencing itself.

Another study was instrumented in [60] that focuses on explicit user requirements as the sole input for personalization mechanisms. It compares different hybridization variants between knowledge-based and collaborative algorithms, where collaborative filtering interprets explicit requirements as a form of rating. Result sets of knowledge-based recommenders turn out to be very precise, if users formulated some specific requirements. However, when only few constraints apply and result sets are large the ranking function is not always able to identify the best matching items. In contrast, collaborative filtering learns the relationships between requirements and actually purchased items. Therefore, the study shows that a cascading strategy where the knowledge-based recommender removes candidates based on hard criteria and a collaborative algorithm does the ranking does best.

Consequently, in [57] a meta-level hybridization approach between knowledge-based and collaborative filtering was proposed and validated. There collaborative filtering learns constraints that map users' requirements onto catalog properties of purchased items and feeds them as input into a knowledge-based recommender that acts as the principal component. Offline experiments on historical data provided initial evidence that such an approach is able to outperform the knowledge base elicited from the domain experts with respect to algorithm's accuracy. Based on these first promising results further research on automatically extracting constraints from historic transaction data will take place.

Case studies on productive systems are the most realistic form of evaluation because users act under real-world conditions and possess an intrinsic motivation to use the system. In [13] experiences from two commercial projects in the domains

of financial services and electronic consumer goods are reported. In the latter domain a conversational recommender for digital cameras has been fielded that was utilized by more than 200.000 online shoppers at a large Austrian price comparison platform. Replies to an online questionnaire supported the hypothesis that advisor applications help users to better orientate themselves when being confronted with large sets of choices. A significantly higher share of users successfully completed their product search when using the conversational recommender compared to those that did not use it. Installations of knowledge-based recommenders in the financial services domain follow a different business model as they support sales agents while interacting with their prospective clients. Empirical surveys among sales representatives figured out that time savings when interacting with clients are a big advantage which in turn allows sales staff to identify sales opportunities [13, 17].

In [58] a case study researches how the application of a knowledge-based conversational sales recommender on a Webshop for Cuban cigars affects online shoppers behavior. Therefore the sales records in the period before and after introducing the recommender were analyzed. One interesting finding of this study is that the list of top ranked items in the two periods differs considerably. In fact items that were infrequently sold in the period before but very often recommended by the system experienced a very high demand. Thus the relative increase of items was positively correlated with how often the recommender proposed these items. The advice given by recommendation applications is followed by users and leads to online conversions. This confirms the results of user studies like [15] that were initially discussed. Finally, another evaluation of a knowledge-based recommender in the tourism domain was conducted to compare conversion rates, i.e., the share of users that turned into bookers, between users and non-users of the interactive sales guide [59]. This study strongly empirically confirms that the probability of users issuing a booking request is more than twice as high for those having interacted with the interactive travel advisor than for the others.

Thus, based on these results we are able to summarize that constraint-based recommendation has been successfully deployed in several commercial application domains and is well accepted by their users.

6.6 Future Research Issues

On the one hand constraint-based recommender systems have proven their utility in many fielded applications on the other hand we can identify several challenges for improvements. Such improvements will lead to enhancing the *quality* for users, the *broadness* of the application fields, and the *development* of recommender software.

Automated product data extraction A constraint-based recommender is only as good as its knowledge base. Consequently, the knowledge base has to be correct, complete, and up-to-date in order to guarantee high quality recommendations. This implies significant maintenance tasks, especially in those domains where data and

recommendation knowledge changes frequently, for example, electronic consumer products. Currently, maintenance is done by human experts, for example, collecting product data or updating rule-bases. However, in many domains at least product data is accessible for machines on the web. By exploiting the internet as a resource for data and knowledge almost all necessary pieces for many recommender applications could be collected. The major research focuses in this context are the automated extraction of product data from different information sources and the automated detection and adaptation of outdated product data. This includes identifying relevant information sources (for instance, Web pages), extracting product data, and resolving contradictions in those data. A related recent challenge is extracting product information directly from digital multimedia products such as books, CDs, DVDs, and TV programs.

However, the fundamental problem for machines is the presentation of data in the web. Data in the Web is usually presented with the goal that humans can easily access and comprehend information. Unfortunately, the opposite is true for computers which are currently not particularly capable in interpreting visual information. Therefore, a fundamental research question is how we can enable machines such that they can “read” the web similarly as humans do. In fact, this task goes far beyond recommender systems and is a central endeavor of the Semantic Web and on a more general level of Artificial Intelligence. Although it seems that currently this task is far too ambitious to be solved in the near future, we can exploit the particularities of recommendation domains. For example, when dealing with the extraction of product data from the web, we can search for product descriptions in tabular form, extract the data of these product descriptions, and instantiate a product database [31]. Of course the success of such methods depends on the domains. For example in the domain of electronic consumer products like digital cameras the description of cameras follows a common structure (e.g., data-sheets of different brands are very similar) whereas in other domains like holiday resorts product descriptions are mostly expressed by natural language text. It has to be mentioned that instead of an automatic translation of human readable content in machine processable data there is the alternative to provide such machine processable data in addition or instead of human readable content. Indeed strong market forces like internet search engine vendors might offer improved search services if machine processable information is provided. For example, product vendors supply their data in specific formats and benefit by an improved ranking in search results. However, in this scenario search machine vendors dictate which descriptions of which products are available for recommendations purposes which leads to a strong dependency on single authorities. Therefore, the aim to enable computers to read the web as humans do remains an important point on the research agenda.

Community-based knowledge acquisition The cornerstone of constraint-based recommendation is efficient knowledge acquisition and maintenance. This problem has been addressed in the past in different dimensions, the main focus lying on knowledge representation and conceptualization issues as well as on process models for capturing and formalizing a domain expert’s knowledge. Historically, one

main assumption of these approaches was that there shall exist one single point of knowledge formalization and in consequence one (user-oriented) conceptualization and a central knowledge acquisition tool. In most cases in real world, however, the domain knowledge is in the heads of different stakeholders, typical examples being cross-department or cross-organization business rules or new types of applications, in which large user communities are sharing knowledge in an open-innovation , web-based environment. Only recently, with the emergence and spread of Web 2.0 and Semantic Web technologies, the opportunities and also the problems of collaborative knowledge acquisition have again become a topic of interest. With regard to the types of knowledge to be acquired, the main focus of these recent developments, however, is on acquiring “structural” knowledge, i.e., on terms, concepts, and relationships among them. New developments aim at going a step further and target at the collaborative acquisition and refinement of domain-constraints and business rules as they represent the most crucial, frequently updated, and thus costly part in many knowledge-based applications. The main questions to be answered comprise the following: How can we automatically detect and resolve conflicts if knowledge acquisition is distributed between different knowledge contributors? How can we assist the knowledge contributors to acquire knowledge by asking them the “right” questions, i.e., minimizing the interaction needed? How can we generate “good” proposals for changing the knowledge base from different, possibly only partially-defined knowledge chunks, i.e., find plausible (in the eyes of the contributors) changes of the knowledge base?

Usually the term *knowledge acquisition* refers to methods supporting the user to formulate rules, constraints, or other logical descriptions depending on the employed language. This task is complicated in recommender systems since in most cases the output includes a preference relation over the recommended items. Consequently, knowledge acquisition has to support also the formulation, debugging, and testing of such preference descriptions [21].

A further factor which complicates the search for a satisfying knowledge base is the demand for high quality explanations . Explanations in constraint-based recommender systems are generated by exploiting the content of the knowledge base. In fact, different knowledge bases can provide the equivalent input/output behavior with respect to recommendations but show significant differences in their explanatory quality. Consequently, a further important goal of knowledge acquisition is supporting the formulation of comprehensible knowledge bases which serve the user to gain confidence in the recommendations.

Knowledge bases for recommender systems have to be considered as dynamic. Unfortunately this dynamics are not only caused by changing product catalogs but also by shifts of customer preferences. For example, the pixel resolution of digital photos considered to be sufficient for printing an A4 picture changes over time because of higher demands for quality. Consequently, automatic detection of such shifts and supporting a subsequent adaptation of the knowledge base are of great interest.

Validation Successfully developing and maintaining recommender knowledge bases requires intelligent testing environments that can guarantee the recommendations' correctness. Particularly in application areas where a certain recommendation quality must be assured (e.g., financial products) a company employing recommender systems has to be sure about the quality of the recommendation process and its outcome. So, future research must focus on developing mechanisms to automatically configure optimal test suites that both maximize the probability of identifying faulty elements in the recommender knowledge base and minimize the number of test cases needed to achieve this goal. Minimizing the number of test cases is important because domain experts must validate them manually. This validation output fits nicely with supporting knowledge acquisition since any feedback from a knowledge engineer can be exploited for learning recommendation knowledge bases. In particular an interesting research question is to which extend arguments of a user in favor or against a recommendation can be exploited to improve knowledge bases. In [51] an algorithm is described which learns constraints based on arguments why an example (e.g., a product) should be recommended or not.

Recommendation of configurable products and services With the production of the Model T about 100 years ago, Henry Ford revolutionized manufacturing by employing mass production (the efficient production of many identical products). Nowadays, mass production is an outmoded business model, and companies must provide goods and services that fit customers' individual needs. In this context, mass customization – the production of highly variant products and services under mass production pricing conditions – has become the new paradigm. A phenomenon accompanying mass customization is mass confusion, which occurs when items are too numerous and complex for users to survey. Developing recommender technologies that apply to configurable products and services can help tackle mass confusion. For example, recommender technology could be adapted to help the uninformed customer to discover her wishes, needs, and product requirements in a domain of almost unlimited product variants. However, recommendation of configurable products pushes the limits of current recommender technologies. Current techniques assume that items to be recommended can be extensionally represented. But configuration domains frequently offer such a high product variance that the set of all possible configurations can only be intensionally characterized by configuration descriptions. For example, configurable systems may comprise thousand of components and connections. In these domains searching for the most preferred configurations satisfying the customer requirements is a challenging task.

Intelligibility and explanation To be convincing, recommendations must be explained to customers. When they can challenge a recommendation and see why a system recommended a specific product customers will start to trust that system. In general, explanations are provided for outputs of recommender systems and serve a wide spectrum of tasks, for example, increase transparency and trust, persuade a customer, or improve customer satisfaction just to name some. These explanations depend on the state of the recommendation process and the user profile, for example, her aims, desires, and prior knowledge. The vision of future recommender

systems is that pro-actively information is provided to the user such that explanation goals are optimized, i.e., if the recommender recognizes that a customer does not understand the differences between alternative products then explanations of these differences are offered. Conversely, customers with a rich background of a product domain and a clear understanding what they want can be offered a quick jump to a recommendation with a detailed technical justification. Consequently, the research challenge is to create an artificial recommender agent that acts flexibly to the needs of customers. Explanations are a cornerstone in such a general endeavor.

Theories of consumer buying behavior A truly intelligent recommender agent adapts to the user. This implies that the recommender has a model of the user which allows predictions about her reaction depending on the information provided. In particular, if we have a model about the influencing factors of consumer buying behavior then it is possible to reason about the best next actions a recommender agent can take. Therefore, research in recommender technology can greatly benefit from insights of cognitive and decision psychology. One can argue that such “intelligent” behavior of recommender agents is questionable from an ethical point of view. However, every information provided to a customer influences her buying behavior. Therefore, it is important to understand the consequences of communications with the customer thus allowing a more planned design of recommender systems.

Context awareness and ambient intelligence Recommender systems may not only be regarded as simple software tools accessible via a PC but rather as intelligent agents recommending actions in various situations. For example, in future cars artificial assistants will provide advice for various driving tasks, such as, overtaking, turning, or parking. In order to give recommendations in such environments the recommender has to be aware of the situation and the goals of a user. Other typical scenarios are recommendations for tourists during their journeys. In such situations, recommendations depend not only on customer preferences but also on the context, which can include attributes such as time of day, season, weather conditions, and ticket availability. Note, that the mentioned scenarios requires so called ambient intelligence. Not the traditional computer is the only interface to the customer but speech and gesture play an important role for the communication between user and recommender.

Semantic Web The W3C states “The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.” In particular Semantic Web technologies offer methods to relate data in the web. This can be exploited to implement a decentralized web of entities who trust each other or relations between customers and products they rated. Based on such relations between customers or products many improvements are feasible. We already mentioned that the extraction of product data and knowledge acquisition can benefit from the machine readable content descriptions. However, we can go a step further and use the information in the Semantic Web to improve the quality of recommendations [22, 63]. In particular, an agent can consider only those ratings of trustworthy agents in order to avoid intentional misguidance. Further-

more, the Semantic Web allows to integrate data of various sources in the reasoning process. On the one hand this enhances knowledge-based recommendation since knowledge is contributed and maintained by a community on a decentralized computing infrastructure and therefore knowledge-acquisition efforts are shared. However, on the other hand many research questions for this scenario arise: How can the quality of recommendations be guaranteed or at least assessed? How can we assess the trustworthiness and quality of knowledge sources? How can we make sure that for the description of products and services there is a common agreement on the concepts and values used? How can we deal with differences in the meaning of concepts and values? How can we assess not only the correctness of recommendations but also their completeness?

6.7 Summary

In this chapter we provided an overview of major constraint-based recommendation technologies. These technologies are especially applicable to large and potentially complex product assortments where collaborative filtering and content-based filtering technologies have their drawbacks. The usefulness of constraint-based recommendation technologies has been shown in different commercial applications - those applications have been analyzed in this chapter. Finally, to trigger further research in the field, we provide an extensive overview of important future research directions.

References

1. Bistarelli, S., Montanary, U., Rossi, F.: Semiring-based Constraint Satisfaction and Optimization. *Journal of the ACM* **44**, 201–236 (1997)
2. Bridge, D.: Towards Conversational Recommender Systems: a Dialogue Grammar Approach. In: D.W. Aha (ed.) *Proceedings of the EWCBR-02 Workshop on Mixed Initiative CBR*, pp. 9–22 (2002)
3. Bridge, D., Goeker, M., McGinty, L., Smyth, B.: Case-based recommender systems. *Knowledge Engineering Review* **20**(3), 315–320 (2005)
4. Burke, R.: Knowledge-Based Recommender Systems. *Encyclopedia of Library and Information Science* **69**(32), 180–200 (2000)
5. Burke, R.: Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction* **12**(4), 331–370 (2002)
6. Burke, R., Hammond, K., Young, B.: Knowledge-based navigation of complex information spaces. In: *Proceedings of the 13th National Conference on Artificial Intelligence, AAAI'96*, pp. 462–468. AAAI Press (1996)
7. Burke, R., Hammond, K., Young, B.: The FindMe Approach to Assisted Browsing. *IEEE Intelligent Systems* **12**(4), 32–40 (1997)
8. Burnett, M.: HCI research regarding end-user requirement specification: a tutorial. *Knowledge-based Systems* **16**, 341–349 (2003)
9. Chen, L., Pu, P.: Evaluating Critiquing-based Recommender Agents. In: *Proceedings of the 21st National Conference on Artificial Intelligence and the Eighteenth Innovative Ap-*

- lications of Artificial Intelligence Conference, AAAI/IAAI'06, pp. 157–162. AAAI Press, Boston, Massachusetts, USA (2006)
10. Felfernig, A.: Reducing Development and Maintenance Efforts for Web-based Recommender Applications. *Web Engineering and Technology* **3**(3), 329–351 (2007)
 11. Felfernig, A., Burke, R.: Constraint-based recommender systems: technologies and research issues. In: *Proceedings of the 10th International Conference on Electronic Commerce, ICEC'08*, pp. 1–10. ACM, New York, NY, USA (2008)
 12. Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M.: Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence* **152**(2), 213–234 (2004)
 13. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: An integrated environment for the development of knowledge-based recommender applications. *International Journal of Electronic Commerce* **11**(2), 11–34 (2007)
 14. Felfernig, A., Friedrich, G., Schubert, M., Mandl, M., Mairitsch, M., Teppan, E.: Plausible Repairs for Inconsistent Requirements. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09*, pp. 791–796. Pasadena, CA, USA (2009)
 15. Felfernig, A., Gula, B.: An Empirical Study on Consumer Behavior in the Interaction with Knowledge-based Recommender Applications. In: *Proceedings of the 8th IEEE International Conference on E-Commerce Technology (CEC 2006) / Third IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2006)*, p. 37 (2006)
 16. Felfernig, A., Isak, K., Kruggel, T.: Testing Knowledge-based Recommender Systems. *OE-GAI Journal* **4**, 12–18 (2007)
 17. Felfernig, A., Isak, K., Szabo, K., Zachar, P.: The VITA Financial Services Sales Support Environment. In: *Proceedings of the 22nd AAAI Conference on Artificial Intelligence and the 19th Conference on Innovative Applications of Artificial Intelligence, AAAI/IAAI'07*, pp. 1692–1699. Vancouver, Canada (2007)
 18. Felfernig, A., Kiener, A.: Knowledge-based Interactive Selling of Financial Services using FSAdvisor. In: *Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference, AAAI/IAAI'05*, pp. 1475–1482. AAAI Press, Pittsburgh, PA (2005)
 19. Felfernig, A., Mairitsch, M., Mandl, M., Schubert, M., Teppan, E.: Utility-based Repair of Inconsistent Requirements. In: *Proceedings of the 22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligence Systems, IEAAIE 2009*, Springer Lecture Notes on Artificial Intelligence, pp. 162–171. Springer, Taiwan (2009)
 20. Felfernig, A., Shchekotykhin, K.: Debugging user interface descriptions of knowledge-based recommender applications. In: *Proceedings of the 11th International Conference on Intelligent User Interfaces, IUI 2006*, pp. 234–241. ACM Press, New York, NY, USA (2006)
 21. Felfernig, A., Teppan, E., Friedrich, G., Isak, K.: Intelligent debugging and repair of utility constraint sets in knowledge-based recommender applications. In: *Proceedings of the ACM International Conference on Intelligent User Interfaces, IUI 2008*, pp. 217–226 (2008)
 22. Gil, Y., Motta, E., Benjamins, V., Musen, M. (eds.): *The Semantic Web - ISWC 2005*, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6–10, 2005, *Lecture Notes in Computer Science*, vol. 3729. Springer (2005)
 23. Godfrey, P.: Minimization in Cooperative Response to Failing Database Queries. *International Journal of Cooperative Information Systems* **6**(2), 95–149 (1997)
 24. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems* **22**(1), 5–53 (2004)
 25. Jannach, D.: Advisor Suite - A knowledge-based sales advisory system. In: R.L. de Mantaras, L. Saitta (eds.) *Proceedings of European Conference on Artificial Intelligence, ECAI 2004*, pp. 720–724. IOS Press, Valencia, Spain (2004)
 26. Jannach, D.: Techniques for Fast Query Relaxation in Content-based Recommender Systems. In: C. Freksa, M. Kohlhase, K. Schill (eds.) *Proceedings of the 29th German Conference on AI, KI 2006*, pp. 49–63. Springer LNAI 4314, Bremen, Germany (2006)
 27. Jannach, D.: Fast computation of query relaxations for knowledge-based recommenders. *AI Communications* **22**(4), 235–248 (2009)

28. Jannach, D., Bundgaard-Joergensen, U.: SAT: A Web-Based Interactive Advisor For Investor-Ready Business Plans. In: Proceedings of International Conference on e-Business, pp. 99–106 (2007)
29. Jannach, D., Kreutler, G.: Personalized User Preference Elicitation for e-Services. In: Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Services, EEE 2005, pp. 604–611. IEEE Computer Society, Hong Kong (2005)
30. Jannach, D., Kreutler, G.: Rapid Development Of Knowledge-Based Conversational Recommender Applications With Advisor Suite. *Journal of Web Engineering* **6**, 165–192 (2007)
31. Jannach, D., Shchekotykhin, K., Friedrich, G.: Automated Ontology Instantiation from Tabular Web Sources - The AllRight System. *Journal of Web Semantics* **7**(3), 136–153 (2009)
32. Jannach, D., Zanker, M., Fuchs, M.: Constraint-based recommendation in tourism: A multi-perspective case study. *Information Technology and Tourism* **11**(2), 139–156 (2009)
33. Junker, U.: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In: Proceedings of National Conference on Artificial Intelligence, AAAI'04, pp. 167–172. AAAI Press, San Jose (2004)
34. Konstan, J., Miller, N., Maltz, D., Herlocker, J., Gordon, R., Riedl, J.: GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM* **40**(3), 77–87 (1997)
35. Lakshmanan, L., Leone, N., Ross, R., Subrahmanian, V.: ProbView: A Flexible Probabilistic Database System. *ACM Transactions on Database Systems* **22**(3), 419–469 (1997)
36. Lorenzi, F., Ricci, F., Tostes, R., Brasil, R.: Case-based recommender systems: A unifying view. In: *Intelligent Techniques in Web Personalisation*, no. 3169 in *Lecture Notes in Computer Science*, pp. 89–113. Springer (2005)
37. Mahmood, T., Ricci, F.: Learning and adaptivity in interactive recommender systems. In: Proceedings of the 9th International Conference on Electronic Commerce, ICEC'07, pp. 75–84. ACM Press, New York, NY, USA (2007)
38. Maimon, O., Rokach, L. Data Mining by Attribute Decomposition with semiconductors manufacturing case study, in *Data Mining for Design and Manufacturing: Methods and Applications*, D. Braha (ed.), Kluwer Academic Publishers, pp. 311–336 (2001)
39. McSherry, D.: Incremental Relaxation of Unsuccessful Queries. In: P. Funk, P.G. Calero (eds.) Proceedings of the European Conference on Case-based Reasoning, ECCBR 2004, no. 3155 in *Lecture Notes in Artificial Intelligence*, pp. 331–345. Springer (2004)
40. McSherry, D.: Retrieval Failure and Recovery in Recommender Systems. *Artificial Intelligence Review* **24**(3–4), 319–338 (2005)
41. Mirzadeh, N., Ricci, F., Bansal, M.: Feature Selection Methods for Conversational Recommender Systems. In: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service on e-Technology, e-Commerce and e-Service, EEE 2005, pp. 772–777. IEEE Computer Society, Washington, DC, USA (2005)
42. Pazzani, M.: A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review* **13**(5–6), 393–408 (1999)
43. Peischl, B., Nica, M., Zanker, M., Schmid, W.: Recommending effort estimation methods for software project management. In: Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology - WPRRS Workshop, vol. 3, pp. 77–80. Milano, Italy (2009)
44. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Dynamic Critiquing. In: Proceedings of the 7th European Conference on Case-based Reasoning, ECCBR 2004, pp. 763–777. Madrid, Spain (2004)
45. Reiter, R.: A theory of diagnosis from first principles. *Artificial Intelligence* **32**(1), 57–95 (1987)
46. R.Elmasri, Navathe, S.: *Fundamentals of Database Systems*. Addison Wesley (2006)
47. Ricci, F., Mirzadeh, N., Bansal, M.: Supporting User Query Relaxation in a Recommender System. In: Proceedings of the 5th International Conference in E-Commerce and Web-Technologies, EC-Web 2004, pp. 31–40. Zaragoza, Spain (2004)

48. Ricci, F., Mirzadeh, N., Venturini, A.: Intelligent query management in a mediator architecture. In: Proceedings of the 1st International IEEE Symposium on Intelligent Systems, vol. 1, pp. 221–226. Varna, Bulgaria (2002)
49. Ricci, F., Nguyen, Q.: Acquiring and Revising Preferences in a Critique-Based Mobile Recommender System. *IEEE Intelligent Systems* **22**(3), 22–29 (2007)
50. Ricci, F., Venturini, A., Cavada, D., Mirzadeh, N., Blaas, D., Nones, M.: Product Recommendation with Interactive Query Management and Twofold Similarity. In: Proceedings of the 5th International Conference on Case-Based Reasoning, pp. 479–493. Trondheim, Norway (2003)
51. Shchekotykhin, K., Friedrich, G.: Argumentation based constraint acquisition. In: Proceedings of the IEEE International Conference on Data Mining (2009)
52. Smyth, B., McGinty, L., Reilly, J., McCarthy, K.: Compound Critiques for Conversational Recommender Systems. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, WI'04, pp. 145–151. Maebashi, Japan (2004)
53. Thompson, C., Goeker, M., Langley, P.: A Personalized System for Conversational Recommendations. *Journal of Artificial Intelligence Research* **21**, 393–428 (2004)
54. Tsang, E.: *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego (1993)
55. Williams, M., Tou, F.: RABBIT: An interface for database access. In: Proceedings of the ACM '82 Conference, ACM'82, pp. 83–87. ACM, New York, NY, USA (1982)
56. Winterfeldt, D., Edwards, W.: *Decision Analysis and Behavioral Research*. Cambridge University Press (1986)
57. Zanker, M.: A Collaborative Constraint-Based Meta-Level Recommender. In: Proceedings of the 2nd ACM International Conference on Recommender Systems, RecSys 2008, pp. 139–146. ACM Press, Lausanne, Switzerland (2008)
58. Zanker, M., Bricman, M., Gordea, S., Jannach, D., Jessenitschnig, M.: Persuasive online-selling in quality & taste domains. In: Proceedings of the 7th International Conference on Electronic Commerce and Web Technologies, EC-Web 2006, pp. 51–60. Springer, Krakow, Poland (2006)
59. Zanker, M., Fuchs, M., Höpken, W., Tuta, M., Müller, N.: Evaluating Recommender Systems in Tourism - A Case Study from Austria. In: Proceedings of the International Conference on Information and Communication Technologies in Tourism, ENTER 2008, pp. 24–34 (2008)
60. Zanker, M., Jessenitschnig, M.: Case-studies on exploiting explicit customer requirements in recommender systems. *User Modeling and User-Adapted Interaction: The Journal of Personalization Research*, A. Tuzhilin and B. Mobasher (Eds.): Special issue on Data Mining for Personalization **19**(1-2), 133–166 (2009)
61. Zanker, M., Jessenitschnig, M., Jannach, D., Gordea, S.: Comparing recommendation strategies in a commercial context. *IEEE Intelligent Systems* **22**(May/June), 69–73 (2007)
62. Zhang, J., Jones, N., Pu, P.: A visual interface for critiquing-based recommender systems. In: Proceedings of the 9th ACM Conference on Electronic Commerce, EC'08, pp. 230–239. ACM, New York, NY, USA (2008)
63. Ziegler, C.: Semantic Web Recommender Systems. In: Proceedings of the EDBT Workshop, EDBT'04, pp. 78–89 (2004)

Chapter 7

Context-Aware Recommender Systems

Gediminas Adomavicius and Alexander Tuzhilin

Abstract The importance of contextual information has been recognized by researchers and practitioners in many disciplines, including e-commerce personalization, information retrieval, ubiquitous and mobile computing, data mining, marketing, and management. While a substantial amount of research has already been performed in the area of recommender systems, most existing approaches focus on recommending the most relevant items to users without taking into account any additional contextual information, such as time, location, or the company of other people (e.g., for watching movies or dining out). In this chapter we argue that relevant contextual information does matter in recommender systems and that it is important to take this information into account when providing recommendations. We discuss the general notion of context and how it can be modeled in recommender systems. Furthermore, we introduce three different algorithmic paradigms – contextual pre-filtering, post-filtering, and modeling – for incorporating contextual information into the recommendation process, discuss the possibilities of combining several context-aware recommendation techniques into a single unifying approach, and provide a case study of one such combined approach. Finally, we present additional capabilities for context-aware recommenders and discuss important and promising directions for future research.

Gediminas Adomavicius
Department of Information and Decision Sciences
Carlson School of Management, University of Minnesota
e-mail: gedas@umn.edu

Alexander Tuzhilin
Department of Information, Operations and Management Sciences
Stern School of Business, New York University
e-mail: atuzhili@stern.nyu.edu

7.1 Introduction and Motivation

The majority of existing approaches to recommender systems focus on recommending the most relevant items to individual users and do not take into consideration any contextual information, such as time, place and the company of other people (e.g., for watching movies or dining out). In other words, traditionally recommender systems deal with applications having only two types of entities, users and items, and do not put them into a context when providing recommendations.

However, in many applications, such as recommending a vacation package, personalized content on a Web site, or a movie, it may not be sufficient to consider only users and items – it is also important to incorporate the *contextual information* into the recommendation process in order to recommend items to users under certain *circumstances*. For example, using the temporal context, a travel recommender system would provide a vacation recommendation in the winter that can be very different from the one in the summer. Similarly, in the case of personalized content delivery on a Web site, it is important to determine what content needs to be delivered (recommended) to a customer and when. More specifically, on weekdays a user might prefer to read world news when she logs on in the morning and the stock market report in the evening, and on weekends to read movie reviews and do shopping.

These observations are consistent with the findings in behavioral research on consumer decision making in marketing that have established that decision making, rather than being invariant, is contingent on the context of decision making. Therefore, accurate prediction of consumer preferences undoubtedly depends upon the degree to which the recommender system has incorporated the relevant contextual information into a recommendation method.

More recently, companies started incorporating some contextual information into their recommendation engines. For example, when selecting a song for the customer, Sourcetone interactive radio (www.sourcetone.com) takes into the consideration the current mood of the listener (the context) that she specified. In case of music recommenders, some of the contextual information, such as listener's mood, may matter for providing better recommendations. However, it is still not clear if context matters for a broad range of other recommendation applications.

In this chapter we discuss the topic of *context-aware recommender systems* (CARS), address this and several other related questions, and demonstrate that, depending on the application domain and the available data, at least certain contextual information can be useful for providing better recommendations. We also propose three major approaches in which the contextual information can be incorporated into recommender systems, individually examine these three approaches, and also discuss how these three separate methods can be combined into one unified approach. Finally, the inclusion of the contextual information into the recommendation process presents opportunities for richer and more diverse interactions between the end-users and recommender systems. Therefore, in this chapter we also discuss novel flexible interaction capabilities in the form of the recommendation query language for context-aware recommender systems.

The rest of the chapter is organized as follows. Section 7.2 discusses the general notion of context as well as how it can be modeled in recommender systems. Section 7.3 presents three different algorithmic paradigms for incorporating contextual information into the recommendation process. Section 7.4 discusses the possibilities of combining several context-aware recommendation techniques and provides a case study of one such combined approach. Additional important capabilities for context-aware recommender systems are described in Section 7.5, and the conclusions and some opportunities for future work are presented in Section 7.6.

7.2 Context in Recommender Systems

Before discussing the role and opportunities of contextual information in recommender systems, in Section 7.2.1 we start by discussing the general notion of context. Then, in Section 7.2.2, we focus on recommender systems and explain how context is specified and modeled there.

7.2.1 *What is Context?*

Context is a multifaceted concept that has been studied across different research disciplines, including computer science (primarily in artificial intelligence and ubiquitous computing), cognitive science, linguistics, philosophy, psychology, and organizational sciences. In fact, an entire conference – CONTEXT (see, for example, <http://context-07.ruc.dk>) – is dedicated exclusively to studying this topic and incorporating it into various other branches of science, including medicine, law, and business. In reference to the latter, a well-known business researcher and practitioner C. K. Prahalad has stated that “the ability to reach out and touch customers anywhere at anytime means that companies must deliver not just competitive products but also unique, real-time customer experiences shaped by *customer context*” and that this would be the next main issue (“big thing”) for the CRM practitioners [57].

Since context has been studied in multiple disciplines, each discipline tends to take its own idiosyncratic view that is somewhat different from other disciplines and is more specific than the standard generic dictionary definition of context as “conditions or circumstances which affect some thing” [70]. Therefore, there exist many definitions of context across various disciplines and even within specific subfields of these disciplines. Bazire and Brézillon [17] present and examine 150 different definitions of context from different fields. This is not surprising, given the complexity and the multifaceted nature of the concept. As Bazire and Brézillon [17] observe:

“... it is difficult to find a relevant definition satisfying in any discipline. Is context a frame for a given object? Is it the set of elements that have any influence on the object? Is it possible to define context a priori or just state the effects a posteriori? Is it something static

or dynamic? Some approaches emerge now in Artificial Intelligence [...]. In Psychology, we generally study a person doing a task in a given situation. Which context is relevant for our study? The context of the person? The context of the task? The context of the interaction? The context of the situation? When does a context begin and where does it stop? What are the real relationships between context and cognition?"

Since we focus on recommender systems in this paper and since the general concept of context is very broad, we try to focus on those fields that are directly related to recommender systems, such as data mining, e-commerce personalization, databases, information retrieval, ubiquitous and mobile context-aware systems, marketing, and management. We follow Palmisano et al. [54] in this section when describing these areas.

Data Mining. In the data mining community, context is sometimes defined as those events which characterize the life stages of a customer and that can determine a change in his/her preferences, status, and value for a company [18]. Examples of context include a new job, the birth of a child, marriage, divorce, and retirement. Knowledge of this contextual information helps (a) mining patterns pertaining to this particular context by focusing only on the *relevant data*; for example, the data pertaining to the daughter's wedding, or (b) selecting only *relevant results*, i.e., those data mining results that are applicable to the particular context, such as the discovered patterns that are related to the retirement of a person.

E-commerce Personalization. Palmisano et al. [54] use the *intent* of a purchase made by a customer in an e-commerce application as contextual information. Different purchasing intents may lead to different types of behavior. For example, the same customer may buy from the same online account different products for different reasons: a book for improving her personal work skills, a book as a gift, or an electronic device for her hobby. To deal with different purchasing intentions, Palmisano et al. [54] build a separate profile of a customer for each purchasing context, and these separate profiles are used for building separate models predicting customer's behavior in specific contexts and for specific segments of customers. Such contextual segmentation of customers is useful, because it results in better predictive models across different e-commerce applications [54].

Recommender systems are also related to e-commerce personalization, since personalized recommendations of various products and services are provided to the customers. The importance of including and using the contextual information in recommendation systems has been demonstrated in [3], where the authors presented a multidimensional approach that can provide recommendations based on contextual information in addition to the typical information on users and items used in many recommendation applications. It was also demonstrated by Adomavicius et al. [3] that the contextual information does matter in recommender systems: it helps to increase the quality of recommendations in certain settings.

Similarly, Oku et al. [53] incorporate additional contextual dimensions (such as time, companion, and weather) into the recommendation process and use machine learning techniques to provide recommendations in a restaurant recommender system. They empirically show that the context-aware approach significantly outper-

forms the corresponding non-contextual approach in terms of recommendation accuracy and user's satisfaction with recommendations.

Since we focus on the use of context in recommender systems in this paper, we will describe these and similar approaches later in the chapter.

Ubiquitous and mobile context-aware systems. In the literature pertaining to the context-aware systems, context was initially defined as the location of the user, the identity of people near the user, the objects around, and the changes in these elements [63]. Other factors have been added to this definition subsequently. For instance, Brown et al. [23] include the date, the season, and the temperature. Ryan et al. [61] add the physical and conceptual statuses of interest for a user. Dey et al. [33] include the user's emotional status and broaden the definition to any information which can characterize and is relevant to the interaction between a user and an application. Some associate the context with the user [33, 35], while others emphasize how context relates to the application [60, 69]. More recently, a number of other techniques for context-aware systems have been discussed in research literature, including hybrid techniques for mobile applications [59, 71] and graphical models for visual recommendation [20].

This contextual information is crucial for providing a broad range of Location-Based Services (LBSes) to the mobile customers [64]. For example, a Broadway theater may want to recommend heavily discounted theater tickets to the Time Square visitors in New York thirty minutes before the show starts (since these tickets will be wasted anyway after the show begins) and send this information to the visitors' smart phones or other communication devices. Note that time, location, and the type of the communication device (e.g., smart phone) constitute contextual information in this application. Brown et al. [22] introduce another interesting application that allows tourists interactively share their sightseeing experiences with remote users, demonstrating the value that context-aware techniques can provide in supporting social activities.

A survey of context-aware mobile computing research can be found in [30], which discusses different models of contextual information, context-sensing technologies, different possible architectures, and a number of context-aware application examples.

Databases. Contextual capabilities have been added to some of the database management systems by incorporating user preferences and returning different answers to database queries depending on the context in which the queries have been expressed and the particular user preferences corresponding to specific contexts. More specifically, in Stephanidis et al. [66] a set of contextual parameters is introduced and preferences are defined for each combination of regular relational attributes and these contextual parameters. Then Stephanidis et al. [66] present a context-aware extension of SQL to accommodate for such preferences and contextual information. Agrawal et al. [7] present another method for incorporating context and user preferences into query languages and develop methods of reconciling and ranking different preferences in order to expeditiously provide ranked answers to contextual queries. Mokbel and Levandoski [52] describe the context-aware and location-aware

database server CoreDB and discuss several issues related to its implementation, including challenges related to context-aware query operators, continuous queries, multi-objective query processing, and query optimization.

Information Retrieval. Contextual information has been proven to be helpful in information retrieval and access [40], although most existing systems base their retrieval decisions solely on queries and document collections, whereas information about search context is often ignored [9]. The effectiveness of a proactive retrieval system depends on the ability to perform context-based retrieval, generating queries which return context-relevant results [46, 65]. In Web searching, context is considered as the set of topics potentially related to the search term. For instance, Lawrence [45] describes how contextual information can be used and proposes several specialized domain-specific context-based search engines. Integration of context into the Web services composition is suggested by Maamar et al. [51]. Most of the current context-aware information access and retrieval techniques focus on the short-term problems and immediate user interests and requests (such as “find all files created during a spring meeting on a sunny day outside an Italian restaurant in New York”), and are not designed to model long-term user tastes and preferences.

Marketing and Management. Marketing researchers have maintained that the purchasing process is contingent upon the context in which the transaction takes place, since the same customer can adopt different decision strategies and prefer different products or brands depending on the context [19, 50]. According to Lilien et al. [47], “consumers vary in their decision-making rules because of the usage situation, the use of the good or service (for family, for gift, for self) and purchase situation (catalog sale, in-store shelf selection, and sales person aided purchase).” Therefore, accurate predictions of consumer preferences should depend on the degree to which we have incorporated the relevant contextual information. In the marketing literature, context has been also studied in the field of behavioral decision theory. In Lussier and Olshavsky [50], context is defined as a task complexity in the brand choice strategy.

The context is defined in Prahalad [57] as “the precise physical location of a customer at any given time, the exact minute he or she needs the service, and the kind of technological mobile device over which that experience will be received.” Further, Prahalad [57] focuses on the applications where the contextual information is used for delivering “unique, real-time customer experiences” based on this contextual information, as opposed to the delivery of competitive products. Prahalad [57] provides an example about the case when he left his laptop in a hotel in Boston, and was willing to pay significant premiums for the hotel shipping the laptop to him in New York in that particular context (he was in New York and needed the laptop really urgently in that particular situation).

To generalize his statements, Prahalad [57] really distinguishes among the following three dimensions of the contextual information: temporal (when to deliver customer experiences), spatial (where to deliver), and technological (how to deliver). Although Prahalad focuses on the real-time experiences (implying that it is

really the present time, “now”), the temporal dimension can be generalized to the past and the future (e.g., I want to see a movie tomorrow in the evening).

As this section clearly demonstrates, context is a multifaceted concept used across various disciplines, each discipline taking a certain angle and putting its “stamp” on this concept. To bring some “order” to this diversity of views, Dourish [34] introduces taxonomy of contexts, according to which contexts can be classified into the representational and the interactional views. In the *representational* view, context is defined with a predefined set of observable attributes, the structure (or schema, using database terminology) of which does not change significantly over time. In other words, the representational view assumes that the contextual attributes are identifiable and known *a priori* and, hence, can be captured and used within the context-aware applications. In contrast, the *interactional* view assumes that the user behavior is induced by an underlying context, but that the context itself is not necessarily observable. Furthermore, Dourish [34] assumes that different types of actions may give rise to and call for different types of relevant contexts, thus assuming a *bidirectional* relationship between activities and underlying contexts: contexts influence activities and also different activities giving rise to different contexts.

In the next section, we take all these different definitions and approaches to context and adapt them to the idiosyncratic needs of recommender systems. As a result, we will also revise and enhance the prior definitions of context used in recommender systems, including those provided in [3, 53, 72].

7.2.2 Modeling Contextual Information in Recommender Systems

Recommender systems emerged as an independent research area in the mid-1990s, when researchers and practitioners started focusing on recommendation problems that explicitly rely on the notion of ratings as a way to capture user preferences for different items. For example, in case of a movie recommender system, John Doe may assign a rating of 7 (out of 10) for the movie “Gladiator,” i.e., set $R_{movie}(\text{John.Doe}, \text{Gladiator})=7$. The recommendation process typically starts with the specification of the initial set of ratings that is either explicitly provided by the users or is implicitly inferred by the system. Once these initial ratings are specified, a recommender system tries to estimate the rating function R

$$R : User \times Item \rightarrow Rating$$

for the (user, item) pairs that have not been rated yet by the users. Here *Rating* is a totally ordered set (e.g., non-negative integers or real numbers within a certain range), and *User* and *Item* are the domains of users and items respectively. Once the function R is estimated for the whole $User \times Item$ space, a recommender system can recommend the highest-rated item (or k highest-rated items) for each user. We call such systems *traditional* or *two-dimensional* (2D) since they consider only the *User* and *Item* dimensions in the recommendation process.

In other words, in its most common formulation, the recommendation problem is reduced to the problem of estimating ratings for the items that have not been seen by a user. This estimation is usually based on the ratings given by this user to other items, ratings given to this item by other users, and possibly on some other information as well (e.g., user demographics, item characteristics). Note that, while a substantial amount of research has been performed in the area of recommender systems, the vast majority of the existing approaches focus on recommending items to users or users to items and do not take into the consideration any additional contextual information, such as time, place, the company of other people (e.g., for watching movies). Motivated by this, in this chapter we explore the area of *context-aware recommender systems (CARS)*, which deal with modeling and predicting user tastes and preferences by incorporating available contextual information into the recommendation process as explicit additional categories of data. These long-term preferences and tastes are usually expressed as *ratings* and are modeled as the function of not only items and users, but also of the context. In other words, ratings are defined with the *rating function* as

$$R : User \times Item \times Context \rightarrow Rating,$$

where *User* and *Item* are the domains of users and items respectively, *Rating* is the domain of ratings, and *Context* specifies the contextual information associated with the application. To illustrate these concepts, consider the following example.

Example 7.1. Consider the application for recommending movies to users, where users and movies are described as relations having the following attributes:

- **Movie:** the set of all the movies that can be recommended; it is defined as `Movie(MovieID, Title, Length, ReleaseYear, Director, Genre)`.
- **User:** the people to whom movies are recommended; it is defined as `User(UserID, Name, Address, Age, Gender, Profession)`.

Further, the contextual information consists of the following three types that are also defined as relations having the following attributes:

- **Theater:** the movie theaters showing the movies; it is defined as `Theater(TheaterID, Name, Address, Capacity, City, State, Country)`.
- **Time:** the time when the movie can be or has been seen; it is defined as `Time(Date, DayOfWeek, TimeOfWeek, Month, Quarter, Year)`. Here, attribute `DayOfWeek` has values `Mon, Tue, Wed, Thu, Fri, Sat, Sun`, and attribute `TimeOfWeek` has values `“Weekday”` and `“Weekend”`.
- **Companion:** represents a person or a group of persons with whom one can see a movie. It is defined as `Companion(companionType)`, where attribute `companionType` has values `“alone”, “friends”, “girlfriend/boyfriend”, “family”, “co-workers”, and “others”`.

Then the rating assigned to a movie by a person also depends on where and how the movie has been seen, with whom, and at what time. For example,

the type of movie to recommend to college student Jane Doe can differ significantly depending on whether she is planning to see it on a Saturday night with her boyfriend vs. on a weekday with her parents.

As we can see from this example and other cases, the contextual information Context can be of different *types*, each type defining a certain aspect of context, such as time, location (e.g., Theater), companion (e.g., for seeing a movie), purpose of a purchase, etc. Further, each contextual type can have a complicated structure reflecting complex nature of the contextual information. Although this complexity of contextual information can take many different forms, one popular defining characteristic is the *hierarchical* structure of contextual information that can be represented as trees, as is done in most of the context-aware recommender and profiling systems, including [3] and [54]. For instance, the three contexts from Example 1 can have the following hierarchies associated with them: *Theater*: TheaterID \rightarrow City \rightarrow State \rightarrow Country; *Time*: Date \rightarrow DayOfWeek \rightarrow TimeOfWeek, Date \rightarrow Month \rightarrow Quarter \rightarrow Year.¹

Furthermore, we follow the representational view of Dourish [34], as described at the end of Section 7.2.1, and assume that the context is defined with a predefined set of observable attributes, the structure of which does not change significantly over time. Although there are some papers in the literature that take the interactional approach to modeling contextual recommendations, such as [11] that models context through a short-term memory (STM) interactional approach borrowed from psychology, most of the work on context-aware recommender systems follows the representational view. As stated before, we also adopt this representational view in this chapter and assume that there is a predefined finite set of contextual types in a given application and that each of these types has a well-defined structure.

More specifically, we follow Palmisano et al. [54], and also Adomavicius et al. [3] to some extent, in this paper and define the contextual information with a *set* of *contextual dimensions* \mathbf{K} , each contextual dimension K in \mathbf{K} being defined by a set of q attributes $K = (K^1, \dots, K^q)$ having a hierarchical structure and capturing a particular type of context, such as *Time* or *CommunicatingDevice*. The values taken by attribute K^q define *finer* (more *granular*) levels, while K^1 values define *coarser* (less granular) levels of contextual knowledge. For example, Figure 7.1(a) presents a four-level hierarchy for the contextual attribute K specifying the intent of a purchasing transaction in an e-retailer application. While the root (coarsest level) of the hierarchy for K defines purchases in all possible contexts, the next level is defined by attribute $K^1 = \{Personal, Gift\}$, which labels each customer purchase either as a personal purchase or as a gift. At the next, finer level of the hierarchy, “Personal” value of attribute K^1 is further split into a more detailed personal context: personal purchase made for the work-related or other purposes. Similarly, the *Gift* value for K^1 can be split into a gift for a partner or a friend and a gift for parents or others.

¹ For the sake of completeness, we would like to point out that not only the contextual dimensions, but also the traditional User and Item dimensions can have their attributes form hierarchical relationships. For example, the main two dimensions from Example 1 can have the following hierarchies associated with them: *Movie*: MovieID \rightarrow Genre; *User*: UserID \rightarrow Age, UserID \rightarrow Gender, UserID \rightarrow Profession.

Thus, the K^2 level is $K^2 = \{PersonalWork, PersonalOther, GiftPartner/Friend, Gift-Parent/Other\}$. Finally, attribute K^2 can be split into further levels of hierarchy, as shown in Figure 7.1(a).²

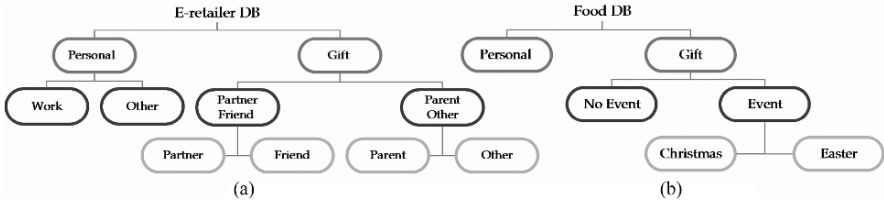


Fig. 7.1: Contextual information hierarchical structure: (a) e-retailer dataset, (b) food dataset [54].

Contextual information was also defined in [3] as follows. In addition to the classical *User* and *Item* dimensions, additional contextual dimensions, such as *Time*, *Location*, etc., were also introduced using the OLAP-based³ *multidimensional data (MD) model* widely used in the data warehousing applications in databases [29, 41]. Formally, let D_1, D_2, \dots, D_n be dimensions, two of these dimensions being *User* and *Item*, and the rest being contextual. Each dimension D_i is a subset of a Cartesian product of some attributes (or fields) $A_{ij}, (j = 1, \dots, k_i)$, i.e., $D_i \subseteq A_{i1} \times A_{i2} \times \dots \times A_{ik_i}$, where each attribute defines a domain (or a set) of values. Moreover, one or several attributes form a *key*, i.e., they uniquely define the rest of the attributes [58]. In some cases, a dimension can be defined by a single attribute, and $k_i = 1$ in such cases. For example, consider the three-dimensional recommendation space $User \times Item \times Time$, where the *User* dimension is defined as $User \subseteq UName \times Address \times Income \times Age$ and consists of a set of users having certain names, addresses, incomes, and being of a certain age. Similarly, the *Item* dimension is defined as $Item \subseteq IName \times Type \times Price$ and consists of a set of items defined by their names, types and the price. Finally, the *Time* dimension can be defined as $Time \subseteq Year \times Month \times Day$ and consists of a list of days from the starting to the ending date (e.g. from January 1, 2003 to December 31, 2003).

Given dimensions D_1, D_2, \dots, D_n , we define the recommendation space for these dimensions as a Cartesian product $S = D_1 \times D_2 \times \dots \times D_n$. Moreover, let *Rating* be a rating domain representing the ordered set of all possible rating values. Then the *rating function* is defined over the space $D_1 \times \dots \times D_n$ as

$$R : D_1 \times \dots \times D_n \rightarrow Rating.$$

² For simplicity and illustration purposes, this figure uses only two-way splits. Obviously, three-way, four-way and, more generally, multi-way splits are also allowed.

³ OLAP stands for OnLine Analytical Processing, which represents a popular approach to manipulation and analysis of data stored in multi-dimensional cube structures and which is widely used for decision support.

For instance, continuing the $User \times Item \times Time$ example considered above, we can define a rating function R on the recommendation space $User \times Item \times Time$ specifying how much user $u \in User$ liked item $i \in Item$ at time $t \in Time$, $R(u, i, t)$.

Visually, ratings $R(d_1, \dots, d_n)$ on the recommendation space $S = D_1 \times D_2 \times \dots \times D_n$ can be stored in a multidimensional cube, such as the one shown in Figure 7.2. For example, the cube in Figure 7.2 stores ratings $R(u, i, t)$ for the recommendation space $User \times Item \times Time$, where the three tables define the sets of users, items, and times associated with $User$, $Item$, and $Time$ dimensions respectively. For example, rating $R(101, 7, 1) = 6$ in Figure 7.2 means that for the user with User ID 101 and the item with Item ID 7, rating 6 was specified during the weekday.

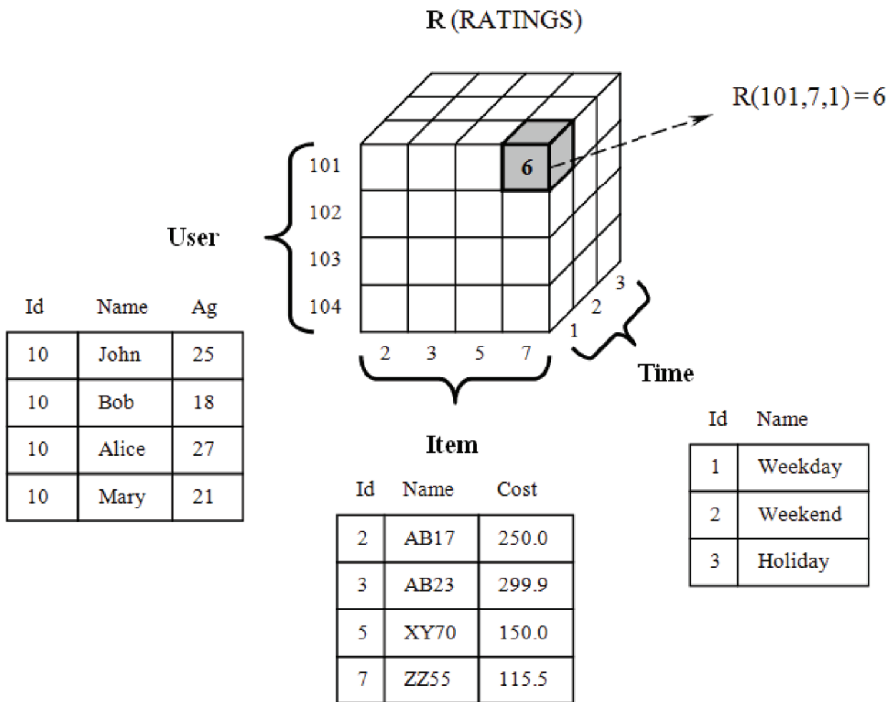


Fig. 7.2: Multidimensional model for the $User \times Item \times Time$ recommendation space.

The rating function R introduced above is usually defined as a partial function, where the initial set of ratings is known. Then, as usual in recommender systems, the goal is to estimate the unknown ratings, i.e., make the rating function R total.

The main difference between the multidimensional (MD) contextual model described above and the previously described contextual model lies in that contextual information in the MD model is defined using classical OLAP hierarchies, whereas the contextual information in the previous case is defined with more general hierar-

chical taxonomies, that can be represented as trees (both balanced and unbalanced), directed acyclic graphs (DAGs), or various other types of taxonomies. Further, the ratings in the MD model are stored in the multidimensional cubes, whereas the ratings in the other contextual model are stored in more general hierarchical structures.

We would also like to point out that not all contextual information might be relevant or useful for recommendation purposes. Consider, for example, a book recommender system. Many types of contextual data could potentially be obtained by such a system from book buyers, including: (a) purpose of buying the book (possible options: for work, for leisure, ...); (b) planned reading time (weekday, weekend, ...); (c) planned reading place (at home, at school, on a plane, ...); (d) the value of the stock market index at the time of the purchase. Clearly some types of contextual information can be more relevant in a given application than some other types. For example, in the previous example, the value of a stock market can be less relevant as contextual information than the purpose of buying a book. There are several approaches to determining the relevance of a given type of contextual information. In particular, the relevance determination can either be done *manually*, e.g., using domain knowledge of the recommender system's designer or a market expert in a given application domain, or *automatically*, e.g., using numerous existing feature selection procedures from machine learning [42], data mining [48], and statistics [28], based on existing ratings data during the data preprocessing phase. The detailed discussion of the specific feature selection procedures is beyond the scope of this paper; in the remainder of this chapter we will assume that only the relevant contextual information is stored in the data.

7.2.3 Obtaining Contextual Information

The contextual information can be obtained in a number of ways, including:

- *Explicitly*, i.e., by directly approaching relevant people and other sources of contextual information and explicitly gathering this information either by asking direct questions or eliciting this information through other means. For example, a website may obtain contextual information by asking a person to fill out a web form or to answer some specific questions before providing access to certain web pages.
- *Implicitly* from the data or the environment, such as a change in location of the user detected by a mobile telephone company. Alternatively, temporal contextual information can be implicitly obtained from the timestamp of a transaction. Nothing needs to be done in these cases in terms of interacting with the user or other sources of contextual information – the source of the implicit contextual information is accessed directly and the data is extracted from it.
- *Inferring* the context using statistical or data mining methods. For example, the household identity of a person flipping the TV channels (husband, wife, son, daughter, etc.) may not be explicitly known to a cable TV company; but it can be inferred with reasonable accuracy by observing the TV programs watched

and the channels visited using various data mining methods. In order to infer this contextual information, it is necessary to build a predictive model (i.e., a classifier) and train it on the appropriate data. The success of inferring this contextual information depends very significantly on the quality of such classifier, and it also varies considerably across different applications. For example, it was demonstrated in [54] that various types of contextual information can be inferred with a reasonably high degree of accuracy in certain applications and using certain data mining methods, such as Naïve Bayes classifiers and Bayesian Networks.

Finally, the contextual information can be “hidden” in the data in some latent form, and we can use it *implicitly* to better estimate the unknown ratings *without explicitly knowing* this contextual information. For instance, in the previous example, we may want to estimate how much a person likes a particular TV program by modeling the member of the household (husband, wife, etc.) watching the TV program as a latent variable. It was also shown in [54] that this deployment of latent variables, such as intent of purchasing a product (e.g., for yourself vs. as a gift, work-related vs. pleasure, etc.), whose true values were unknown but that were explicitly modeled as a part of a Bayesian Network (BN), indeed improved the predictive performance of that BN classifier. Therefore, even without any *explicit* knowledge of the contextual information (e.g., which member of the household is watching the program), recommendation accuracy can still be improved by modeling and inferring this contextual information implicitly using carefully chosen learning techniques (e.g., by using latent variables inside well-designed recommendation models). A similar approach of using latent variables is presented in [11].

As explained in Section 7.2.1, we focus on the representational view of Dourish [34], and assume that the context is defined with a *predefined* set of contextual attributes, the structure of which does not change over time. The implication of this assumption is that we need to identify and acquire contextual information before actual recommendations are made. If the acquisition process of this contextual information is done explicitly or even implicitly, it should be conducted as a part of the overall data collection process. All this implies that the decisions of which contextual information should be relevant and collected for an application should be done at the application design stage and well in advance of the time when actual recommendations are provided.

One methodology of deciding which contextual attributes should be used in a recommendation application (and which should not) is presented in [3]. In particular, Adomavicius et al. [3] propose that a wide range of contextual attributes should be initially selected by the domain experts as possible candidates for the contextual attributes for the application. For example, in a movie recommendation application described in Example 1, we can initially consider such contextual attributes as Time, Theater, Companion, Weather, as well as a broad set of other contextual attributes that can possibly affect the movie watching experiences, as initially identified by the domain experts for the application. Then, after collecting the data, including the rating data and the contextual information, we may apply various types of statistical tests identifying which of the chosen contextual attributes are truly significant

in the sense that they indeed affect movie watching experiences, as manifested by significant deviations in ratings across different values of a contextual attribute. For example, we may apply pairwise t-tests to see if good weather vs. bad weather or seeing a movie alone vs. with a companion significantly affect the movie watching experiences (as indicated by statistically significant changes in rating distributions). This procedure provides an example of screening all the initially considered contextual attributes and filtering out those that do not matter for a particular recommendation application. For example, we may conclude that the Time, Theater and Companion contexts matter, while the Weather context does not in the considered movie recommendation application.

7.3 Paradigms for Incorporating Context in Recommender Systems

The usage of contextual information in recommender systems can be traced to the work by Herlocker and Konstan [36], who hypothesized that the inclusion of knowledge about the user's *task* into the recommendation algorithm in certain applications can lead to better recommendations. For example, if we want to recommend books as gifts for a child, then we might want to specify several books that the child already has (and likes) and provide this information (i.e., a task profile) to the recommender system for calculating new recommendations. Note that this approach operates within the traditional 2D $User \times Item$ space, since the task specification for a specific user consists of a list of sample items; in other words, besides the standard *User* and *Item* dimensions, no additional contextual dimensions are used. However, this approach serves as a successful illustration of how additional relevant information (in the form of user-specified task-relevant item examples) can be incorporated into the standard collaborative filtering paradigm. Further, the use of interest scores assigned to topics has been applied to building contextual user profiles in recommender systems [73].

Different approaches to using contextual information in the recommendation process can be broadly categorized into two groups: (1) recommendation via *context-driven querying and search*, and (2) recommendation via *contextual preference elicitation and estimation*. The context-driven querying and search approach has been used by a wide variety of mobile and tourist recommender systems [2, 27, 68]. Systems using this approach typically use contextual information (obtained either directly from the user, e.g., by specifying current mood or interest, or from the environment, e.g., obtaining local time, weather, or current location) to query or search a certain repository of resources (e.g., restaurants) and present the best matching resources (e.g., nearby restaurants that are currently open) to the user. One of the early examples of this approach is the Cyberguide project [2], which developed several *tour guide* prototypes for different hand-held platforms. Abowd et al. [2] discuss different architectures and features necessary to provide realistic tour guide services to mobile users and, more specifically, the role that the contextual knowl-

edge of the user's current and past locations can play in the recommendation and guiding process. Among the many other examples of context-aware tourist guide systems proposed in research literature we can mention GUIDE [31], INTRIGUE [14], COMPASS [68], and MyMap [32] systems.

The other general approach to using contextual information in the recommendation process, i.e., via contextual preference elicitation and estimation, represents a more recent trend in context-aware recommender systems literature [3, 53, 55, 72]. In contrast to the previously discussed context-driven querying and search approach (where the recommender systems use the current context information and specified current user's interest as queries to search for the most appropriate content), techniques that follow this second approach attempt to model and learn user preferences, e.g., by observing the interactions of this and other users with the systems or by obtaining preference feedback from the user on various previously recommended items. To model users' context-sensitive preferences and generate recommendations, these techniques typically either adopt existing collaborative filtering, content-based, or hybrid recommendation methods to context-aware recommendation settings or apply various intelligent data analysis techniques from data mining or machine learning (such as Bayesian classifiers or support vector machines).

While both general approaches offer a number of research challenges, in the remainder of this chapter we will focus on the second, more recent trend of the contextual preference elicitation and estimation in recommender systems. We do not mention that it is possible to design applications that combine the techniques from both general approaches (i.e., both context-driven querying and search as well as contextual preference elicitation and estimation) into a single system. For example, the UbiquiTO system [27], which implements a mobile tourist guide, provides intelligent adaptation not only based on the specific context information, but also uses various rule-based and fuzzy set techniques to adapt the application content based on the user preferences and interests. Similarly, the News@hand system [26] uses semantic technologies to provide personalized news recommendations that are retrieved using user's concept-based queries or calculated according to a specific user's (or a user group's) profile.

To start the discussion of the contextual preference elicitation and estimation techniques, note that, in its general form, a traditional 2-dimensional (2D) ($User \times Item$) recommender system can be described as a *function*, which takes partial user preference data as its *input* and produces a list of recommendations for each user as an *output*. Accordingly, Figure 7.3 presents a general overview of the traditional 2D recommendation process, which includes three components: data (input), 2D recommender system (function), and recommendation list (output). Note that, as indicated in Figure 7.3, after the recommendation function is defined (or constructed) based on the available data, recommendation list for any given user u is typically generated by using the recommendation function on user u and all candidate items to obtain a predicted rating for each of the items and then by ranking all items according to their predicted rating value. Later in this section, we will discuss how the use of contextual information in each of those three components gives rise to three different paradigms for context-aware recommender systems.

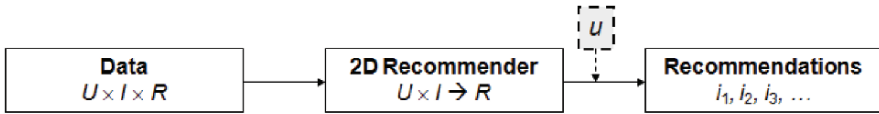


Fig. 7.3: General components of the traditional recommendation process.

As mentioned in Section 7.2.2, traditional recommender systems are built based on the knowledge of *partial user preferences*, i.e., user preferences for some (often limited) set of items, and the input data for traditional recommender systems is typically based on the records of the form $\langle user, item, rating \rangle$. In contrast, context-aware recommender systems are built based on the knowledge of *partial contextual user preferences* and typically deal with data records of the form $\langle user, item, context, rating \rangle$, where each specific record includes not only how much a given user liked a specific item, but also the contextual information in which the item was consumed by this user (e.g., $Context = Saturday$). Also, in addition to the descriptive information about users (e.g., demographics), items (e.g., item features), and ratings (e.g., multi-criteria rating information), context-aware recommender systems may also make use of additional context attributes, such as context hierarchies (e.g., $Saturday \rightarrow Weekend$) mentioned in Section 7.2.2. Based on the presence of this additional contextual data, several important questions arise: How contextual information should be reflected when modeling user preferences? Can we reuse the wealth of knowledge in traditional (non-contextual) recommender systems to generate context-aware recommendations? We will explore these questions in this chapter in more detail.

In the presence of available contextual information, following the diagrams in Figure 7.4, we start with the data having the form $U \times I \times C \times R$, where C is additional contextual dimension and end up with a list of contextual recommendations i_1, i_2, i_3, \dots for each user. However, unlike the process in Figure 7.3, which does not take into account the contextual information, we can apply the information about the current (or desired) context c at various stages of the recommendation process. More specifically, the context-aware recommendation process that is based on contextual user preference elicitation and estimation can take one of the three forms, based on which of the three components the context is used in, as shown in Figure 7.4:

- *Contextual pre-filtering* (or contextualization of recommendation input). In this recommendation paradigm (presented in Figure 7.4a), contextual information drives data selection or data construction for that specific context. In other words, information about the current context c is used for selecting or constructing the relevant set of data records (i.e., ratings). Then, ratings can be predicted using any traditional 2D recommender system on the selected data.
- *Contextual post-filtering* (or contextualization of recommendation output). In this recommendation paradigm (presented in Figure 7.4b), contextual information is initially ignored, and the ratings are predicted using any traditional 2D

recommender system on the *entire* data. Then, the resulting set of recommendations is adjusted (*contextualized*) for each user using the contextual information.

- *Contextual modeling* (or contextualization of recommendation function). In this recommendation paradigm (presented in Figure 7.4c), contextual information is used directly in the modeling technique as part of rating estimation.

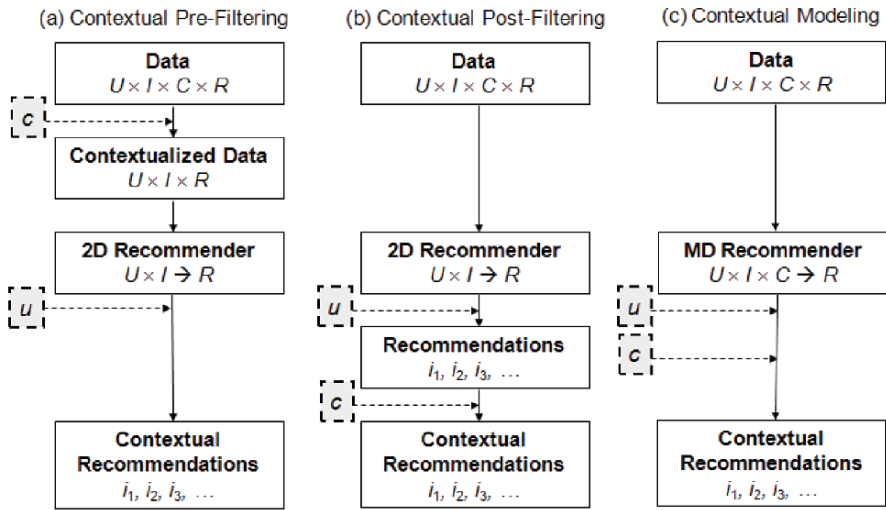


Fig. 7.4: Paradigms for incorporating context in recommender systems.

In the remainder of this section we will discuss these three approaches in detail.

7.3.1 Contextual Pre-Filtering

As shown in Figure 7.4a, the contextual pre-filtering approach uses contextual information to select or construct the most relevant 2D (*User × Item*) data for generating recommendations. One major advantage of this approach is that it allows deployment of any of the numerous traditional recommendation techniques previously proposed in the literature [5]. In particular, in one possible use of this approach, context *c* essentially serves as a *query* for selecting (filtering) relevant ratings data. An example of a contextual data filter for a movie recommender system would be: if a person wants to see a movie on Saturday, *only* the Saturday rating data is used to recommend movies. Note that this example represents an *exact pre-filter*. In other words, the data filtering query has been constructed using exactly the specified context.

For example, following the contextual pre-filtering paradigm, Adomavicius et al. [3] proposed a *reduction-based approach*, which reduces the problem of multidimensional

mensional (MD) contextual recommendations to the standard 2D $User \times Item$ recommendation space. Therefore, as with any contextual pre-filtering approach, one important benefit of the reduction-based approach is that all the previous research on 2D recommender systems is directly applicable in the MD case after the reduction is done. In particular, let $R_{User \times Item}^D: U \times I \rightarrow Rating$ be any 2D rating estimation function that, given existing ratings D (i.e., D contains records $\langle user, item, rating \rangle$ for each of the known, user-specified ratings), can calculate a prediction for any rating, e.g., $R_{User \times Item}^D(John, StarWars)$. Then, a 3-dimensional rating prediction function supporting the context of time can be defined similarly as $R_{User \times Item \times Time}^D: U \times I \times T \rightarrow Rating$, where D contains records $\langle user, item, time, rating \rangle$ for the user-specified ratings. Then the 3-dimensional prediction function can be expressed through a 2D prediction function in several ways, including:

$$\forall (u, i, t) \in U \times I \times T, R_{User \times Item \times Time}^D(u, i, t) = R_{User \times Item}^{D[Time=t]}(User, Item, Rating)(u, i).$$

Here $[Time = t]$ denotes a simple contextual pre-filter, and $D[Time = t](User, Item, Rating)$ denotes a rating dataset obtained from D by selecting only the records where $Time$ dimension has value t and keeping only the values for $User$ and $Item$ dimensions, as well as the value of the rating itself. I.e., if we treat a dataset of 3-dimensional ratings D as a relation, then $D[Time = t](User, Item, Rating)$ is simply another relation obtained from D by performing two relational operations: selection and, subsequently, projection.

However, the exact context sometimes can be too narrow. Consider, for example, the context of watching a movie with a girlfriend in a movie theater on Saturday or, i.e., $c = (Girlfriend, Theater, Saturday)$. Using this exact context as a data filtering query may be problematic for several reasons. First, certain aspects of the overly specific context may not be significant. For example, user's movie watching preferences with a girlfriend in a theater on Saturday may be exactly the same as on Sunday, but different from Wednesday's. Therefore, it may be more appropriate to use a more general context specification, i.e., *Weekend* instead of *Saturday*. And second, exact context may not have enough data for accurate rating prediction, which is known as the "sparsity" problem in recommender systems literature. In other words, the recommender system may not have enough data points about the past movie watching preferences of a given user with a girlfriend in a theater on Saturday.

Context generalization. Adomavicius et al. [3] introduce the notion of *generalized pre-filtering*, which allows to generalize the data filtering query obtained based on a specified context. More formally, let's define $c' = (c'_1, \dots, c'_k)$ to be a generalization of context $c = (c_1, \dots, c_k)$ if and only if $c_i \rightarrow c'_i$ for every $i = 1, \dots, k$ in the corresponding context hierarchy. Then, c' (instead of c) can be used as a data query to obtain contextualized ratings data.

Following the idea of context generalization, Adomavicius et al. [3] proposed to use not a simple pre-filter $[Time = t]$, which represents the exact context t of the rating (u, i, t) , but rather a generalized pre-filter $[Time \in S_t]$, where S_t denotes some

superset of context t . Here S_t is called a *contextual segment* [3]. For example, if we would like to predict how much John Doe would like to see the “Gladiator” movie on Monday, i.e., to calculate $R_{User \times Item \times Time}^D(JohnDoe, Gladiator, Monday)$, we could use not only other user-specified *Monday* ratings for prediction, but *Weekday* ratings in general. In other words, for every (u, i, t) where $t \in Weekday$, we can predict the rating as $R_{User \times Item \times Time}^D(u, i, t) = R_{User \times Item}^{D[Time \in Weekday]}(User, Item, AGGR(Rating))(u, i)$. More generally, in order to estimate some rating $R(u, i, t)$, we can use some specific contextual segment S_t as: $R_{User \times Item \times Time}^D(u, i, t) = R_{User \times Item}^{D[Time \in S_t]}(User, Item, AGGR(Rating))(u, i)$.

Note, that we have used the $AGGR(Rating)$ notation in the above expressions, since there may be several user-specified ratings with the same *User* and *Item* values for different *Time* instances in dataset D belonging to some contextual segment S_t (e.g., different ratings for *Monday* and *Tuesday*, all belonging to segment *Weekday*). Therefore, we have to aggregate these values using some aggregation function, e.g., averaging, when reducing the dimensionality of the recommendation space. The above 3-dimensional reduction-based approach can be extended to a general pre-filtering method reducing an arbitrary n -dimensional recommendation space to an m -dimensional one (where $m < n$). In this chapter we will assume that $m = 2$ because traditional recommendation algorithms are only designed for the two-dimensional $User \times Item$ case. Note, that there typically exist multiple different possibilities for context generalization, based on the context taxonomy and the desired context granularity. For example, let’s assume that we have the following contextual taxonomies (*is-a* or *belongs-to* relationships) that can be derived from context hierarchies:

- *Company*: Girlfriend \rightarrow Friends \rightarrow NotAlone \rightarrow AnyCompany;
- *Place*: Theater \rightarrow AnyPlace;
- *Time*: Saturday \rightarrow Weekend \rightarrow AnyTime.

Then, the following are just several examples of possible generalizations c' of the above-mentioned context $c = (Girlfriend, Theater, Saturday)$:

- $c' = (Girlfriend, AnyPlace, Saturday)$;
- $c' = (Friends, Theater, AnyTime)$;
- $c' = (NotAlone, Theater, Weekend)$;

Therefore, choosing the “right” generalized pre-filter becomes an important problem. One option is to use a manual, expert-driven approach; e.g., always generalize specific days of week into more general *Weekday* or *Weekend*. Another option is to use a more automated approach, which could empirically evaluate the predictive performance of the recommender system on contextualized input datasets obtained from each generalized pre-filter, and then would automatically choose the pre-filter with best performance. An interesting and important research issue is how to deal with potential computational complexity of this approach due to context granularity; in other words, in cases of applications with highly granular contexts, there may exist a very large number of possible context generalizations, for which exhaustive search techniques would not be practical. For such cases, effective greedy approaches would need to be developed. Among the related work, Jiang

and Tuzhilin [39] examine optimal levels of granularity of customer segments in order to maximize predictive performance of segmentation methods. Applicability of these techniques in the context-aware recommender systems settings constitutes an interesting problem for future research.

Also note that the reduction-based approach is related to the problems of building local models in machine learning and data mining [10]. Rather than building the global rating estimation model utilizing all the available ratings, the reduction-based approach builds a *local* rating estimation model that uses only the ratings pertaining to the user-specified criteria in which a recommendation is made (e.g., morning). It is important to know if a local model generated by the reduction-based approach outperforms the global model of the traditional 2D technique, where all the information associated with the contextual dimensions is simply ignored. For example, it is possible that it is better to use the contextual pre-filtering to recommend movies to see in the movie theaters on weekends, but use the traditional 2D technique for movies to see at home on VCRs. This is the case because the reduction-based approach, on the one hand, focuses recommendations on a *particular segment* and builds a local prediction model for this segment, but, on the other hand, computes these recommendations based on a *smaller* number of points limited to the considered segment. This tradeoff between having *more relevant* data for calculating an unknown rating based only on the ratings with the same or similar context and having *fewer* data points used in this calculation belonging to a particular segment (i.e., the *sparsity* effect) explains why the reduction-based recommendation method can outperform traditional 2D recommendation techniques on some segments and underperform on others. Which of these two trends dominates on a particular segment may depend on the application domain and on the specifics of the available data. Based on this observation, Adomavicius et al. [3] propose to combine a number of contextual pre-filters with the traditional 2D technique (i.e., as a default filter, where no filtering is done); this approach will be described as a case study in Section 7.4.

Among some recent developments, Ahn et al. [8] use a technique similar to the contextual pre-filtering to recommend advertisements to mobile users by taking into account user location, interest, and time, and Lombardi et al. [49] evaluate the effect of contextual information using a pre-filtering approach on the data obtained from an online retailer. Also, Baltrunas and Ricci [16] take a somewhat different approach to contextual pre-filtering in proposing and evaluating the benefits of the *item splitting* technique, where each item is split into several fictitious items based on the different contexts in which these items can be consumed. Similarly to the item splitting idea, Baltrunas and Amatriain [15] introduce the idea of *micro-profiling* (or user splitting), which splits the user profile into several (possibly overlapping) sub-profiles, each representing the given user in a particular context. The predictions are done using these contextual micro-profiles instead of a single user model. Note that these data construction techniques fit well under the contextual pre-filtering paradigm, because they are following the same basic idea (as the data filtering techniques described earlier) – using contextual information to reduce the problem of multidimensional recommendations to the standard 2D *User* \times *Item* space, which then allows to use any traditional recommendation techniques for rating prediction.

7.3.2 Contextual Post-Filtering

As shown in Figure 7.4b, the contextual post-filtering approach ignores context information in the input data when generating recommendations, i.e., when generating the ranked list of all candidate items from which any number of top- N recommendations can be made, depending on specific values of N . Then, the contextual post-filtering approach adjusts the obtained recommendation list for each user using contextual information. The recommendation list adjustments can be made by:

- *Filtering* out recommendations that are irrelevant (in a given context), or
- *Adjusting* the ranking of recommendations on the list (based on a given context).

For example, in a movie recommendation application, if a person wants to see a movie on a weekend, and on weekends she only watches comedies, the system can filter out all non-comedies from the recommended movie list. More generally, the basic idea for contextual post-filtering approaches is to analyze the contextual preference data for a given user in a given context to find specific item usage patterns (e.g., user Jane Doe watches only comedies on weekends) and then use these patterns to adjust the item list, resulting in more “contextual” recommendations, as depicted in Figure 7.5.

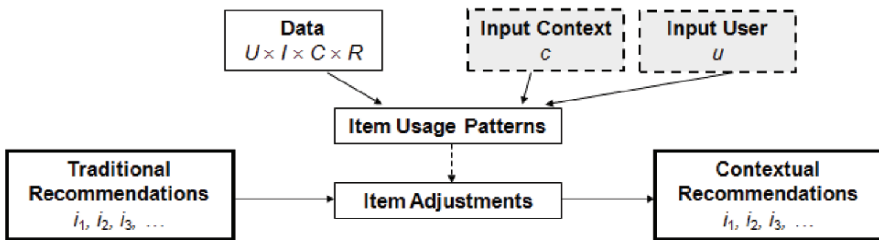


Fig. 7.5: Final phase of the contextual post-filtering approach: recommendation list adjustment.

As with many recommendation techniques, the contextual post-filtering approaches can be classified into heuristic and model-based techniques. *Heuristic* post-filtering approaches focus on finding common item characteristics (attributes) for a given user in a given context (e.g., preferred actors to watch in a given context), and then use these attributes to adjust the recommendations, including:

- *Filtering* out recommended items that do not have a significant number of these characteristics (e.g., to be recommended, the movies must have at least two of the preferred actors in a given context), or
- *Ranking* recommended items based on how many of these relevant characteristics they have (e.g., the movies that star more of the user’s preferred actors in a given context will be ranked higher).

In contrast, *model-based* post-filtering approaches can build predictive models that calculate the probability with which the user chooses a certain type of item in a given context, i.e., probability of relevance (e.g., likelihood of choosing movies of a certain genre in a given context), and then use this probability to adjust the recommendations, including:

- *Filtering* out recommended items that have the probability of relevance smaller than a pre-defined minimal threshold (e.g., remove movies of genres that have a low likelihood of being picked), or
- *Ranking* recommended items by weighting the predicted rating with the probability of relevance.

Panniello et al. [55] provide an experimental comparison of the exact pre-filtering method (discussed in Section 7.3.1) versus two different post-filtering methods – *Weight* and *Filter* – using several real-world e-commerce datasets. The *Weight* post-filtering method reorders the recommended items by weighting the predicted rating with the probability of relevance in that specific context, and the *Filter* post-filtering method filters out recommended items that have small probability of relevance in the specific context. Interestingly, the empirical results show that the *Weight* post-filtering method dominates the exact pre-filtering, which in turn dominates the *Filter* post-filtering method, thus, indicating that the best approach to use (pre- or post-filtering) really depends on a given application.

As was the case with the contextual pre-filtering approach, a major advantage of the contextual post-filtering approach is that it allows using any of the numerous traditional recommendation techniques previously proposed in the literature [5]. Also, similarly to the contextual pre-filtering approaches, incorporating context generalization techniques into post-filtering techniques constitutes an interesting issue for future research.

7.3.3 Contextual Modeling

As shown in Figure 7.4c, the contextual modeling approach uses contextual information *directly* in the recommendation function as an explicit predictor of a user's rating for an item. While contextual pre-filtering and post-filtering approaches can use traditional 2D recommendation functions, the contextual modeling approach gives rise to truly multidimensional recommendation functions, which essentially represent predictive models (built using decision tree, regression, probabilistic model, or other technique) or heuristic calculations that incorporate contextual information in addition to the user and item data, i.e., $Rating = R(User, Item, Context)$. A significant number of recommendation algorithms – based on a variety of heuristics as well as predictive modeling techniques – have been developed over the last 10-15 years, and some of these techniques can be extended from the 2D to the multidimensional recommendation settings. We present a few examples of multidimensional

mensional heuristic-based and model-based approaches for contextual modeling [4] in the rest of this section.

7.3.3.1 Heuristic-Based Approaches

The traditional two-dimensional (2D) neighborhood-based approach [21, 62] can be extended to the multidimensional case, which includes the contextual information, in a straightforward manner by using an n -dimensional distance metric instead of the user-user or item-item similarity metrics traditionally used in such techniques. To see how this is done, consider an example of the $User \times Item \times Time$ recommendation space. Following the traditional nearest neighbor heuristic that is based on the weighted sum of relevant ratings, the prediction of a specific rating $r_{u,i,t}$ in this example can be expressed as (see [4] for additional details):

$$r_{u,i,t} = k \sum_{(u',i',t') \neq (u,i,t)} W((u,i,t), (u',i',t')) \times r_{u',i',t'},$$

where $W((u,i,t), (u',i',t'))$ describes the “weight” rating $r_{u',i',t'}$ carries in the prediction of $r_{u,i,t}$, and k is a normalizing factor. Weight $W((u,i,t), (u',i',t'))$ is typically inversely related to the distance between points (u,i,t) and (u',i',t') in multidimensional space, i.e., $dist[(u,i,t), (u',i',t')]$. In other words, the closer the two points are (i.e., the smaller the distance between them), the more weight $r_{u',i',t'}$ carries in the weighted sum. One example of such relationship would be $W((u,i,t), (u',i',t')) = 1/dist[(u,i,t), (u',i',t')]$, but many alternative specifications are also possible. As before, the choice of the distance metric $dist$ is likely to depend on a specific application. One of the simplest ways to define a multidimensional $dist$ function is by using the reduction-like approach (somewhat similar to the one described in Section 7.3.1), by taking into account only the points with the same contextual information, i.e.,

$$dist[(u,i,t), (u',i',t')] = \begin{cases} dist[(u,i), (u',i')], & \text{if } t = t' \\ +\infty, & \text{otherwise} \end{cases}$$

This distance function makes $r_{u,i,t}$ depend only on the ratings from the segment of points having the same values of time t . Therefore, this case is reduced to the standard 2-dimensional rating estimation on the segment of ratings having the same context t as point (u,i,t) . Furthermore, if we further refine function $dist[(u,i), (u',i')]$ in so that it depends only on the distance between users when $i = i'$, then we would obtain a method that is similar to the pre-filtering approach described earlier. Moreover this approach easily extends to an arbitrary n -dimensional case by setting the distance d between two rating points to $dist[(u,i), (u',i')]$ if and only if the contexts of these two points are the same.

Other ways to define the distance function would be to use the weighted Manhattan distance metric, i.e.,

$$\text{dist}[(u, i, t), (u', i', t')] = w_1 d_1(u, u') + w_2 d_2(i, i') + w_3 d_3(t, t'),$$

or the weighted Euclidean distance metric, i.e.,

$$\text{dist}[(u, i, t), (u', i', t')] = \sqrt{w_1 d_1^2(u, u') + w_2 d_2^2(i, i') + w_3 d_3^2(t, t')}$$

where d_1, d_2 , and d_3 are distance functions defined for dimensions User, Item, and Time respectively, and w_1, w_2 , and w_3 are the weights assigned for each of these dimensions (e.g., according to their importance). In summary, distance function $\text{dist}[(u, i, t), (u', i', t')]$ can be defined in many different ways and, while in many systems it is typically computed between ratings of the same user or of the same item, it constitutes an interesting research problem to identify various more general ways to define this distance and compare these different ways in terms of predictive performance.

7.3.3.2 Model-Based Approaches

There have been several model-based recommendation techniques proposed in recommender systems literature for the traditional two-dimensional recommendation model [5]. Some of these methods can be directly extended to the multidimensional case, such as the method proposed in [12], who show that their 2D technique outperforms some of the previously known collaborative filtering methods.

The method proposed by Ansari et al. [12] combines the information about users and items into a single hierarchical regression-based Bayesian preference model that uses Markov Chain Monte Carlo (MCMC) techniques to estimate its parameters. Specifically, let's assume that there are N_U users, where each user u is defined by vector z_u of observed attributes of user u , such as gender, age, and income. Also assume that there are N_I items, where each item i is defined by vector w_i of attributes of the item, such as price, weight, and size. Let r_{ui} be the rating assigned to item i by user u , where r_{ui} is a real-valued number. Moreover, ratings r_{ui} are only known for some subset of all possible (user, item) pairs. Then the unknown rating estimation problem is defined as

$$r_{ui} = x'_{ui}\mu + z'_u\gamma_i + w'_i\lambda_u + e_{ui}, \quad e_{ui} \sim N(0, \sigma^2), \quad \lambda_u \sim N(0, \Lambda), \quad \gamma_i \sim N(0, \Gamma)$$

where observed (known) values of the model are ratings r_{ui} assigned by user u for item i , user attributes z_u , item attributes w_i , and vector $x_{ui} = z_u \otimes w_i$, where \otimes is the Kronecker product, i.e., a long vector containing all possible cross-product combinations between individual elements of z_u and w_i . Intuitively, this equation presents a regression model specifying unknown ratings r_{ui} in terms of the characteristics z_u of user u , the characteristics w_i of item i , and the interaction effects x_{ui} between them. Interaction effects arise from the hierarchical structure of the model and are intended to capture effects such as how the age of a user changes his or her preferences for certain genres of movies. Vector μ in the above equation represents

unobserved (unknown) slope of the regression line, i.e., unknown coefficients in the regression model that need to be estimated, as discussed below. Vector γ_i represents weight coefficients specific to item i that determine idiosyncrasy of item i , i.e., the unobserved heterogeneity of item i that are not explicitly recorded in the item profiles, such as direction, music and acting for the movies. Similarly, vector λ_u represents weight coefficients specific to user u that determine idiosyncrasy of that user, i.e., the unobserved user effects (heterogeneity) of user u . Finally, the error term e_{ui} is normally distributed with mean zero and standard deviation σ . Further, we consider a hierarchical model and assume that regression parameters γ_i and λ_u are normally distributed as $\gamma_i \sim N(0, \Gamma)$ and $\lambda_u \sim N(0, \Lambda)$, where Γ and Λ are unknown covariance matrices. The parameters of the model are μ, σ^2, Λ , and Γ . They are estimated from the data containing the already known ratings using the MCMC methods described in [12].

While the approach presented in [12] is described in the context of the traditional two-dimensional recommender systems, it can be directly extended to include the contextual information. For example, assume that we have a third dimension *Time* that is defined by the following two attributes (variables): (a) Boolean variable *week-end* specifying whether a movie was seen on a weekend or not, and (b) a positive integer variable *numdays* indicating the number of days after the release when the movie was seen.

In such a case, the Ansari et al. [12] model can be extended to the third (*Time*) dimension as in Adomavicius and Tuzhilin [4]:

$$r_{uit} = x'_{uit}\mu + p'_{ui}\theta_t + q'_{it}\lambda_u + r'_{iu}\gamma_i + z'_{it}\delta_{it} + w'_i\pi_{iu} + y'_t\sigma_{ui} + e_{uit}$$

where $e_{uit} \sim N(0, \sigma^2)$, $\gamma_i \sim N(0, \Gamma)$, $\lambda_u \sim N(0, \Lambda)$, $\theta_t \sim N(0, \Theta)$, $\delta_{it} \sim N(0, \Delta)$, $\pi_{iu} \sim N(0, \Pi)$, and $\sigma_{ui} \sim N(0, \Sigma)$.

This model encompasses the effects of observed and unobserved user-, item- and temporal-variables and their interactions on rating r_{uit} of user u for movie i seen at time t . The variables z_u , w_i and y_t stand for the observed attributes of users (e.g., demographics), movies (e.g., genre) and time dimension (e.g., weekend, numdays). The vector x_{uit} represents the interaction effects of the user, movies, and time variables, and its coefficient μ represents the unobserved (unknown) slope of the regression line, i.e., unknown coefficients in the above regression model that need to be estimated, as discussed below. The vectors λ_u , γ_i and θ_t are random effects that stand for the unobserved sources of heterogeneity of users (e.g., their ethnic background), movies (e.g., the story, screenplay, etc.) and temporal effects (e.g., was the movie seen on a holiday or not, the season when it was released, etc). The vector p_{ui} represents the interaction of the observed user and item variables, and likewise q_{it} and r_{iu} . The vector σ_{ui} represents the interaction of the unobserved user and item attributes, and vectors π_{iu} and δ_{it} have similar types of interactions. Finally, the parameters $\mu, \sigma^2, \Lambda, \Gamma, \Theta, \Delta, \Pi$, and Σ of the model can be estimated from the data of the already known ratings using Markov Chain Monte Carlo (MCMC) methods, as was done in [12].

Finally, note that the number of parameters in the above model that would need to be estimated rises with the number of dimensions and, therefore, the sparsity of known ratings may become a problem. If this is a serious problem for a particular application, then some of the terms in the above model can be dropped, leading to a simplified model. For example, we may decide to drop the term $q_{ii}\lambda_u$, or perhaps some other terms, which would lead to a simpler model with fewer parameters. Note that this simpler model would still take into account the contextual information, e.g., time in this case. Furthermore, parameter estimation of this model can be very time consuming and not scalable. Therefore, one of the research challenges is to make such models more scalable and more robust in terms of the more accurate estimations of unknown ratings. Some of the initial ideas of how to make these approaches more scalable are presented in [67].

In addition to possible extensions of existing 2D recommendation techniques to multiple dimensions, there have also been some new techniques developed specifically for context-aware recommender systems based on the context modeling paradigm. For example, following the general contextual modeling paradigm, Oku et al. [53] propose to incorporate additional contextual dimensions (such as time, companion, and weather) directly into recommendation space and use machine learning technique to provide recommendations in a restaurant recommender system. In particular, they use support vector machine (SVM) classification method, which views the set of liked items and the set of disliked items of a user in various contexts as two sets of vectors in an n -dimensional space, and constructs a separating hyperplane in this space, which maximizes the separation between the two data sets. The resulting hyperplane represents a classifier for future recommendation decisions (i.e., a given item in a specific context will be recommended if it falls on the “like” side of the hyperplane, and will not be recommended if it falls on the “dislike” side). Furthermore, Oku et al. [53] empirically show that context-aware SVM significantly outperforms non-contextual SVM-based recommendation algorithm in terms of predictive accuracy and user’s satisfaction with recommendations. Similarly, Yu et al. [72] use contextual modeling approach to provide content recommendations for smart phone users by introducing context as additional model dimensions and using hybrid recommendation technique (synthesizing content-based, Bayesian-classifier, and rule-based methods) to generate recommendations.

Finally, another model-based approach is presented in [1] where a Personalized Access Model (PAM) is presented that provides a set of personalized context-based services, including context discovery, contextualization, binding and matching services. Then Abbar et al. [1] describe how these services can be combined to form Context-Aware Recommender Systems (CARS) and deployed in order to provide superior context-aware recommendations.

In this section we described various ways to incorporate contextual information into recommendation algorithms within the framework of pre-filtering, post-filtering, and contextual modeling methods. Since CARS is a new and an emerging area of recommender systems, the presented methods constitute only the initial approaches to providing recommendations, and better-performing methods need and should be developed across all these three approaches.

In the next section, we discuss how these different individual methods can be combined together into one common approach.

7.4 Combining Multiple Approaches

As has been well-documented in recommender systems literature, often a combination (a “blend” or an ensemble) of several solutions provides significant performance improvements over the individual approaches [24, 25, 43, 56]. The three paradigms for context-aware recommender systems offer several different opportunities for employing combined approaches.

One possibility is to develop and combine several models of the same type. For example, Adomavicius et al. [3] followed this approach to develop a technique that combines information from several different contextual pre-filters. The rationale for having a number of different pre-filters is based on the fact that, as mentioned earlier, typically there can be multiple different (and potentially relevant) generalizations of the same specific context. For example, context $c = (\text{Girlfriend}, \text{Theater}, \text{Saturday})$ can be generalized to $c_1 = (\text{Friend}, \text{AnyPlace}, \text{Saturday})$, $c_2 = (\text{NotAlone}, \text{Theater}, \text{AnyTime})$, and a number of other contexts. Following this idea, Adomavicius et al. [3] use pre-filters based on the number of possible contexts for each rating, and then combine recommendations resulting from each contextual pre-filter. The general overview of this approach is shown in Figure 7.6. Note that the combination of several pre-filters can be done in multiple ways. For example, for a given context, (a) one could choose the best-performing pre-filter, or (b) use an “ensemble” of pre-filters. In the remainder of Section 7.4, we will discuss the approach developed by Adomavicius et al. [3] in more detail as a case study.

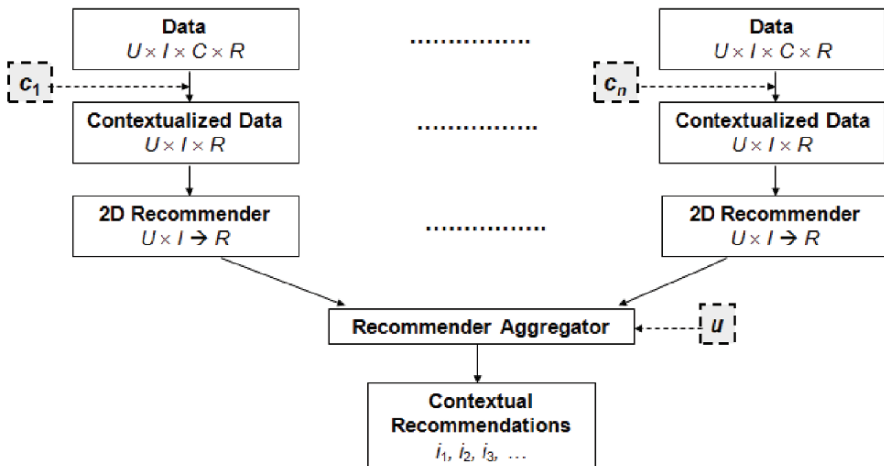


Fig. 7.6: Combining multiple pre-filters: an overview.

Another interesting possibility stems from an observation that complex contextual information can be split into several components, and the utility of each piece of contextual information may be different depending on whether it is used in the pre-filtering, post-filtering, or modeling stage. For example, time information (weekday vs. weekend) may be most useful to pre-filter relevant data, but weather information (sunny vs. rainy) may be the most appropriate to use as a post-filter. Determining the utility of different contextual information with respect to different paradigms of context-aware recommender systems constitutes an interesting and promising direction for future research.

7.4.1 Case Study of Combining Multiple Pre-Filters: Algorithms

The combined approach to rating estimation consists of two phases [3]: (i) using known user-specified ratings (i.e., training data), determine the use of which pre-filters outperforms the traditional CF method; (ii) in order to predict a specific rating in a given context, choose the best pre-filter for that particular context and use the two-dimensional recommendation algorithm on this segment.

The first phase is a pre-processing phase and is usually performed “offline.” It can work with any traditional 2D rating estimation method A , and consists of the following three steps [3]:

1. Among all possible generalized pre-filters, find the ones which result in contextual segments having a significantly large amount of data, i.e., segments with more than N ratings, where N is some predetermined threshold (e.g., $N = 250$ was used in that study). If the recommendation space is “small” (in the number of dimensions and the ranges of attributes in each dimension), the large segments can be obtained simply by doing an exhaustive search in the space of all possible segments. Alternatively, the help of a domain expert (e.g., a marketing manager) or some greedy heuristics could be used to determine the important large segments for the application.
2. For each generalized pre-filter c' determined in Step 1, algorithm A is run using this pre-filter and its predictive performance is determined using a chosen performance metric (this study used F-measure, which is defined as a harmonic mean of Precision and Recall – two standard decision support metrics widely used in information retrieval and, more recently, in recommender systems [37]). Only those pre-filters are kept, where the performance of algorithm A on contextually pre-filtered inputs exceeds the performance of the standard, i.e., non-filtered, version of algorithm A on that *same* data segment.
3. Among the remaining pre-filters, if there exist pre-filters c' and c'' such that $c' \rightarrow c''$ (i.e., c'' is strictly more general than c') and algorithm A demonstrates better performance using pre-filter c'' than pre-filter c' , then c' is deemed to be redundant (less general *and* less accurate) and is removed from the set of pre-filters. The set of remaining contextual segments, denoted $SEGM^*$, constitutes the result of the “offline” phase of the combined approach.

Once the set of high-performing pre-filters $SEGM^*$ is computed, we can perform the second phase of the combined approach and determine which pre-filter to use in “real-time” when an actual recommendation needs to be produced. Given the specific context c of recommendation, the best-performing pre-filter $c' \in SEGM^*$ such that $c \rightarrow c'$ or $c = c'$ is used in conjunction with algorithm A . If no such pre-filter c' exists, then the standard 2D non-filtered algorithm A (i.e., trained on the entire dataset) is used for rating prediction.

The main advantage of the combined pre-filtering approach described in this section is that it uses the contextual pre-filters *only* for those contextual situations where this method outperforms the standard 2D recommendation algorithm, and continues to use the latter where there is no improvement. Therefore, the combined approach is expected to perform equally well or better than the pure 2D approach in practice. The extent to which the combined approach can outperform the 2D approach depends on many different factors, such as the problem domain or quality of data.

7.4.2 Case Study of Combining Multiple Pre-Filters: Experimental Results

To illustrate how the combined approach presented in Section 7.4.1 performs in practice, it was evaluated on a real-world movie recommendation application and compared its performance with the traditional 2D CF method [3]. In this application, in addition to being asked to rate their movie-going experience, the users were asked to specify: (a) *time* when the movie was seen (choices: weekday, weekend, don't remember); furthermore, if seen on a weekend, was it the opening weekend for the movie (choices: yes, no, don't remember); (b) *place* where the movie was seen (choices: in a movie theater, at home, don't remember); and (c) *companion* with whom the movie was seen (choices: alone, with friends, with boyfriend/girlfriend, with family or others). Overall, 1755 ratings were entered by 117 students over a period of 12 months (May'01-Apr'02). Since some students rated very few movies, those students with fewer than 10 ratings were dropped in our analysis. Therefore, from an initial data, the final dataset had only 62 students, 202 movies and 1457 total ratings.

During the first step of the “offline”, pre-filter selection phase, 9 large contextual segments were discovered, as presented in Table 7.1. However, after comparing the performance of the standard CF technique and the contextual pre-filtering CF technique on each of these segments (i.e., second step of the pre-filter selection phase), only 4 “high-performing” segments were found; one of them was subsequently removed after the redundancy check (i.e., third step of the pre-filter selection phase, as described in Section 7.4.1). The remaining set of 3 high-performing pre-filters is shown in Table 7.2, i.e., $SEGM^* = \{Theater-Weekend, Theater, Weekend\}$.

Finally, the resulting high-performing segments $SEGM^*$ were used in the “online”, rating estimation phase. Overall, there were 1373 (of the 1457 collected) ratings that both 2D CF and the combined pre-filtering CF approaches were able to

Table 7.1: Large contextual segments generated in Step 1 of the pre-filter selection algorithm.

Name	Size	Description
Home	727	Movies watched at home
Friends	565	Movies watched with friends
NonRelease	551	Movies watched on other than the opening weekend
Weekend	538	Movies watched on weekends
Theater	526	Movies watched in the movie theater
Weekday	340	Movies watched on weekdays
GBFriend	319	Movies watched with girlfriend/boyfriend
Theater-Weekend	301	Movies watched in the movie theater on weekends
Theater-Friends	274	Movies watched in the movie theater with friends

predict (not all ratings were predicted because of the sparsity-related limitations of data). The results show that the combined pre-filtering CF approach substantially outperformed the traditional 2D CF (1st row in Table 7.3).

Table 7.2: High-performing large contextual segments.

Segment	CF: Segment-trained F-measure	CF: Whole-data-trained F-measure
Theater-Weekend	0.641	0.528
Theater	0.608	0.479
Weekend	0.542	0.484

Note that, as discussed earlier, the combined pre-filtering CF approach incorporates the standard CF approach, since it would use the standard 2D CF to predict the value of any rating that does not belong to any of the discovered high-performing pre-filters. Consequently, in this application, the predictions of the two approaches are identical for all ratings that do not belong to any segment in $\{\textit{Theater-Weekend}, \textit{Theater}, \textit{Weekend}\}$. Since such ratings do not contribute to the differentiation between the two approaches, it is important to determine how well the two approaches do on the ratings from *SEGM**. In this case, there were 743 such ratings in *SEGM** (out of 1373), and the difference in F-measure performance of the two approaches is 0.095 (2nd row in Table 7.3), which is even more substantial than for the previously described case.

In this section, we discussed combining multiple pre-filtering, post-filtering, and contextual modeling methods to generate better predictions, focusing primarily on combining multiple pre-filters, based on [3]. We outlined only some of the main ideas, while leaving most of the problems in this area as wide open and subject of future research. We believe that creative combinations of multiple methods using ensemble techniques from machine learning can significantly increase performance of CARS and constitute an important and interesting area of research.

Table 7.3: Overall comparison based on F-measure.

Comparison	Overall F-measure		Difference between F-measures
	Standard 2D CF	Combined reduction-based CF	
All predictions (1373 ratings)	0.463	0.526	0.063
Predictions on ratings from <i>SEGM</i> * (743 ratings)	0.450	0.545	0.095

7.5 Additional Issues in Context-Aware Recommender Systems

In addition to the three paradigms of incorporating context in recommender systems and the methods of combining these three paradigms, there are several other important topics in context-aware recommender systems, such as how to better utilize contextual information, how to develop richer interaction capabilities with CARS that make recommendations more flexible, and how to build high-performing CARS systems. We discuss these issues in the rest of this section.

Studying Tradeoffs Between Pre-, Post-Filtering, and Contextual Modeling Approaches and Developing Better Understanding of How to Combine Them. In Section 7.3, we only described three types of general CARS paradigms and did not consider tradeoffs between them. In order to achieve better understanding of pre-filtering, post-filtering, and contextual modeling approaches, it is important to conduct studies comparing all the three approaches and identify relative advantages and disadvantages of these methods. One such study is described in [55], where a pre-filtering method is compared to a particular type of post-filtering method in terms of the quality of recommendations. It was shown that neither method dominated the other in terms of providing better recommendations, as measured using the F-measure. Furthermore, Panniello et al. [55] proposed a procedure identifying a set of conditions under which the pre-filtering method should be used vis-à-vis the post-filtering methods.

The work reported in [55] constitutes only the first step towards a systematic program of comparing the pre-filtering, post-filtering, and contextual modeling methods, and much more work is required to develop comprehensive understanding of the relative merits of each of the three methods and to understand which methods perform better than others and under which conditions.

Similarly, more work is required to better understand the combined approach discussed in Section 7.4. This is a fruitful area of research that is open to numerous improvements and important advances including various types of deployments of ensemble methods combining the pre- and post-filtering as well contextual modeling approaches.

Developing Richer Interaction and More Flexible Recommendation Capabilities of CARS. Context-aware recommendations have the following two important properties:

- *Complexity.* Since CARS involve not only users and items in the recommendation process, but also various types of contextual information, the types of such recommendations can be significantly more complex in comparison to the traditional non-contextual cases. For example, in a movie recommendation application, a certain user (e.g., Tom) may seek recommendations for him and his girlfriend of top 3 movies and the best times to see them over the weekend.
- *Interactivity.* The contextual information usually needs to be elicited from the user in the CARS settings. For example, to utilize the available contextual information, a CARS system may need to elicit from the user (Tom) with whom he wants to see a movie (e.g., girlfriend) and when (e.g., over the weekend) before providing any context-specific recommendations.

The combination of these two features calls for the development of more flexible recommendation methods that allow the user to express the types of recommendations that are of interest to them rather than being “hard-wired” into the recommendation engines provided by most of the current vendors that, primarily, focus on recommending top- N items to the user and vice versa. The second requirement of interactivity also calls for the development of tools allowing users to provide inputs into the recommendation process in an interactive and iterative manner, preferably via some well-defined user interface (UI).

Such flexible context-aware recommendations can be supported in several ways. First, Adomavicius et al. [6] developed a *recommendation query language* REQUEST⁴ that allows its users to express in a flexible manner a broad range of recommendations that are tailored to their own individual needs and, therefore, more accurately reflect their interests. REQUEST is based on the multidimensional contextual recommendation model described in Section 7.2.2 and also in [3]. REQUEST supports a wide variety of features, and the interested reader can find the detailed account of these features as well as the formal syntax and various properties of the language in [6]. In addition, Adomavicius et al. [6] provide a discussion of the expressive power of REQUEST and present a multidimensional recommendation algebra that provides the theoretical basis for this language.

So far, we have briefly mentioned only the recommendation query language itself. Since a major argument for introducing such a language is its use by the end-users, it is also very important to develop simple, friendly, and expressive user interfaces (UIs) for supporting flexible but sometimes complex contextual recommendations. High-quality UIs should reduce the complexity and simplify interactions between the end-users and the recommender system and make them available to wider audiences. Developing such UIs constitutes a topic of future research.

Another proposal to provide flexible recommendations is presented in [44], where the FlexRecs system and framework are described. FlexRecs approach supports flexible recommendations over structured data by decoupling the definition of

⁴ REQUEST is an acronym for REcommendation QUery STatements.

a recommendation process from its execution. In particular, a recommendation can be expressed declaratively as a high-level parameterized workflow containing traditional relational operators and novel recommendation-specific operators that are combined together into a recommendation workflow.

In addition to developing languages for expressing context-aware recommendations, it is also important to provide appropriate user interfaces so that the users were able to express flexible recommendations in an interactive manner. For FlexRecs, this entails building a UI for defining and managing recommendation workflows, and for REQUEST this entails providing front-end UI allowing users to express REQUEST queries using visual and interactive methods.

Developing High-Performing CARS Systems and Testing Them on Practical Applications. Most of the work on context-aware recommender systems has been conceptual, where a certain method has been developed, tested on some (often limited) data, and shown to perform well in comparison to certain benchmarks. There has been little work done on developing novel data structures, efficient storage methods, and new systems architectures for CARS. One example of such work is the paper by Hussein et al. [38], where the authors introduce a service-oriented architecture enabling to define and implement a variety of different “building blocks” for context-aware recommender systems, such as recommendation algorithms, context sensors, various filters and converters, in a modular fashion. These building blocks can then be combined and reused in many different ways into systems that can generate contextual recommendations. Another example of such work is Abbar et al. [1], where the authors present a service-oriented approach that implements the Personalized Access Model (PAM) previously proposed by the authors. The implementation is done using global software architecture of CARS developed by the authors and described in [1]. These two papers constitute only the initial steps towards developing better understanding of how to build more user-friendly, scalable, and better performing CARS systems, and much more work is required to achieve this goal.

7.6 Conclusions

In this chapter we argued that relevant contextual information does matter in recommender systems and that it is important to take this contextual information into account when providing recommendations. We also explained that the contextual information can be utilized at various stages of the recommendation process, including at the pre-filtering and the post-filtering stages and also as an integral part of the contextual modeling. We have also showed that various techniques of using the contextual information, including these three methods, can be combined into a single recommendation approach, and we presented a case study describing one possible way of such combining.

Overall, the field of context-aware recommender systems (CARS) is a relatively new and underexplored area of research, and much more work is needed to inves-

tigate it comprehensively. We provided suggestions of several possible future research directions that were presented throughout the paper. In conclusion, CARS constitutes a newly developing and promising research area with many interesting and practically important research problems.

Acknowledgements

Research of G. Adomavicius was supported in part by the National Science Foundation grant IIS-0546443, USA. The authors thank YoungOk Kwon for editorial assistance.

References

1. Abbar, S., Bouzeghoub, M., and Lopez, S., Context-aware recommender systems: A service-oriented approach. *VLDB PersDB Workshop*, 2009.
2. Abowd, G.D., Atkeson, C.G., Hong, J., Long, S., Kooper, R., and Pinkerton, M., Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, 1997.
3. Adomavicius, G., Sankaranarayanan, R., Sen, S., and Tuzhilin, A., Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems (TOIS)*, 23(1):103–145, 2005.
4. Adomavicius, G., and Tuzhilin, A., Incorporating context into recommender systems using multidimensional rating estimation methods. In *Proceedings of the 1st International Workshop on Web Personalization, Recommender Systems and Intelligent User Interfaces (WPR-SIUI 2005)*, 2005.
5. Adomavicius, G., and Tuzhilin, A., Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
6. Adomavicius, G., Tuzhilin, A., and Zheng, R., REQUEST: A query language for customizing recommendations. *Information System Research*, 22(11)s, 2011.
7. Agrawal, R., Rantzaou, R., and Terzi, E., Context-sensitive ranking. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 383–394. ACM, 2006.
8. Ahn, H., Kim, K., and Han, I., Mobile advertisement recommender system using collaborative filtering: MAR-CF. In *Proceedings of the 2006 Conference of the Korea Society of Management Information Systems*, pages 709–715.
9. Akrivas, G., Wallace, M., Andreou, G., Stamou, G., and Kollias, S., Context-sensitive semantic query expansion. In *Proceedings of the IEEE international conference on artificial intelligence systems (ICAIS)*, pages 109–114. Divnomorskoe, Russia, 2002.
10. Alpaydin, E., *Introduction to machine learning*. The MIT Press, 2004.
11. Anand, S.S., and Mobasher, B., Contextual recommendation. *WebMine, LNAI*, 4737:142–160, 2007.
12. Ansari, A., Essegaier, S., and Kohli, R., Internet recommendation systems. *Journal of Marketing Research*, 37(3):363–375, 2000.
13. Arbel, R. and Rokach, L., Classifier evaluation under limited resources, *Pattern Recognition Letters*, 27(14): 1619–1631, 2006.

14. Ardissono, L., Goy, A., Petrone, G., Segnan, M., and Torasso, P., Intrigue: personalized recommendation of tourist attractions for desktop and hand held devices. *Applied Artificial Intelligence*, 17(8):687–714, 2003.
15. Baltrunas, L., and Amatriain, X., Towards time-dependant recommendation based on implicit feedback. In *Workshop on Context-Aware Recommender Systems (CARS 2009)*. New York, 2009.
16. Baltrunas, L., and Ricci, F., Context-dependent items generation in collaborative filtering. In *Workshop on Context-Aware Recommender Systems (CARS 2009)*. New York, 2009.
17. Bazire, M., and P. Brezillon. Understanding context before using it. In Dey, A., and et al., editors, *Proceedings of the 5th International Conference on Modeling and Using Context*. Springer-Verlag, 2005.
18. Berry, M.J., and Linoff, G., *Data mining techniques: for marketing, sales, and customer support*. John Wiley & Sons, Inc. New York, NY, USA, 1997.
19. Bettman, J.R., Luce, M.F., and Payne, J.W., Consumer decision making: A constructive perspective. In M. Tedeschi (editor), *Consumer Behavior and Decision Theory* pages 1–42, 1991.
20. Boutemedjet, S., and Ziou, D., A graphical model for context-aware visual content recommendation. *IEEE Transactions on Multimedia*, 10(1):52–62, 2008.
21. Breese, J.S., Heckerman, D., and Kadie, C., Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, volume 461, pages 43–52. San Francisco, CA, 1998.
22. Brown, B., Chalmers, M., Bell, M., Hall, M., I. MacColl, and Rudman, P., Sharing the square: collaborative leisure in the city streets. In Gellersen, H., Schmidt, K., M. Beaudouin-Lafon, and Mackay, W.E., editors, *Proceedings of the ninth conference on European Conference on Computer Supported Cooperative Work*, pages 427–447. Springer, 2005.
23. Brown, P.J., Bovey, J.D., and Chen, X., Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4:58–64, 1997.
24. Burke, R., Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.
25. Burke, R., Hybrid web recommender systems. *The Adaptive Web*, pages 377–408, 2007.
26. Cantador, I., and Castells, P., Semantic contextualisation in a news recommender system. In *Workshop on Context-Aware Recommender Systems (CARS 2009)*. New York, 2009.
27. Cena, F., Console, L., Gena, C., Goy, A., Levi, G., Modeo, S., and Torre, I., Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide. *AI Communications*, 19(4):369–384, 2006.
28. Chatterjee, S., Hadi, A.S., and Price, B., *Regression analysis by example*. John Wiley and Sons, 2000.
29. Chaudhuri, S., and Dayal, U., An overview of data warehousing and olap technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.
30. Chen, G., and Kotz, D., A survey of context-aware mobile computing research. *Technical Report TR2000-381, Dartmouth Computer Science*, 2000.
31. Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstathiou, C., Developing a context-aware electronic tourist guide: some issues and experiences. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24. ACM, 2000.
32. De Carolis, B., Mazzotta, I., Novielli, N., and Silvestri, V., Using common sense in providing personalized recommendations in the tourism domain. In *Workshop on Context-Aware Recommender Systems (CARS 2009)*. New York, 2009.
33. Dey, A.K., Abowd, G.D., and Salber, D., A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001.
34. Dourish, P., What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1):19–30, 2004.

35. Franklin, D., and Flachsbart, J., All gadget and no representation makes jack a dull environment. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, pages 155–160. AAAI Press, 1998.
36. Herlocker, J.L., and Konstan, J.A., Content-independent task-focused recommendation. *IEEE Internet Computing*, pages 40–47, 2001.
37. Herlocker, J.L., Konstan, J.A., Terveen, L.G., and Riedl, J.T., Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.
38. Hussein, T., Linder, T., Gaulke, W., and Ziegler, J., Context-aware recommendations on rails. In *Workshop on Context-Aware Recommender Systems (CARS 2009)*, New York, 2009.
39. Jiang, T., and Tuzhilin, A., Improving personalization solutions through optimal segmentation of customer bases. *IEEE Transactions on Knowledge and Data Engineering*, 21(3):305–320, 2009.
40. Jones, G., J.F., Glasnevin, D., and Gareth, I., Challenges and opportunities of context-aware information access. In *International Workshop on Ubiquitous Data Management*, pages 53–62, 2005.
41. Kimball, R., and Ross, M., *The data warehousing toolkit*. John Wiley & Sons, New York, 1996.
42. Koller, D., and Sahami, M., Toward optimal feature selection. In *Proceedings of the 13th International Conference on Machine Learning*, pages 284–292. Morgan Kaufmann, 1996.
43. Koren, Y., Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, New York, NY, 2008.
44. Koutrika, G., Bercovitz, B., and H. Garcia-Molina. Flexrecs: expressing and combining flexible recommendations. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 745–758. ACM, Providence, RI, 2009.
45. Lawrence, S., Context in web search. *IEEE Data Engin. Bulletin*, 23(3):25, 2000.
46. Leake, D.B., and Scherle, R., Towards context-based search engine selection. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 109–112. ACM New York, NY, USA, 2001.
47. Lilien, G.L., Kotler, P., and Moorthy, K.S., *Marketing models*. Prentice Hall, 1992.
48. Liu, H., and Motoda, H., *Feature selection for knowledge discovery and data mining*. Springer, 1998.
49. Lombardi, S., Anand, S.S., and Gorgoglione, M., Context and customer behavior in recommendation. In *Workshop on Context-Aware Recommender Systems (CARS 2009)*. New York.
50. Lussier, D.A., and Olshavsky, R.W., Task complexity and contingent processing in brand choice. *Journal of Consumer Research*, pages 154–165, 1979.
51. Maamar, Z., Benslimane, D., and Narendra, N.C., What can context do for web services? *Communications of the ACM*, 49(12):98–103, 2006.
52. Mokbel, M.F., and Levandoski, J.J., Toward context and preference-aware location-based services. In *Proceedings of the Eighth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 25–32. ACM, 2009.
53. Oku, K., Nakajima, S., Miyazaki, J., and Uemura, S., Context-aware SVM for context-dependent information recommendation. In *Proceedings of the 7th International Conference on Mobile Data Management*, page 109, 2006.
54. Palmisano, C., Tuzhilin, A., and Gorgoglione, M., Using context to improve predictive modeling of customers in personalization applications. *IEEE Transactions on Knowledge and Data Engineering*, 20(11):1535–1549, 2008.
55. Panniello, U., Tuzhilin, A., Gorgoglione, M., Palmisano, C., and Pedone, A., Experimental comparison of pre-vs. post-filtering approaches in context-aware recommender systems. In *Proceedings of the 3rd ACM conference on Recommender systems*, pages 265–268. ACM, 2009.

56. Pennock, D.M., and Horvitz, E., Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *IJCAI'99 Workshop: Machine Learning for Information Filtering*, 1999.
57. Prahalad, C.K., Beyond CRM: CK Prahalad predicts customer context is the next big thing. *American Management Association McWorld*, 2004.
58. Ramakrishnan, R., and Gehrke, J., *Database Management Systems*. USA: McGraw Hill Companies, 2000.
59. Ricci, F., and Nguyen, Q.N., Mobyrek: A conversational recommender system for on-the-move travelers. In Fesenmaier, D.R., Werthner, H. and Wober, K.W. (eds.) *Destination Recommendation Systems: Behavioural Foundations and Applications*, CABI Publishing, pages 281–294, 2006.
60. Rodden, T., Cheverst, K., Davies, K., and Dix, A., Exploiting context in HCI design for mobile systems. In *Workshop on Human Computer Interaction with Mobile Devices*, pages 21–22, 1998.
61. Ryan, N., Pascoe, J., and Morse, D., Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant. Gaffney, V., van Leusen, M., Exxon, S.(eds.) *Computer Applications in Archaeology*. British Archaeological Reports, Oxford, 1997.
62. Sarwar, B., Karypis, G., Konstan, J., and Reidl, J., Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.
63. Schilit, B.N., and Theimer, M.M., Disseminating active map information to mobile hosts. *IEEE network*, 8(5):22–32, 1994.
64. Schiller, J.H., and Voisard, A., *Location-based services*. Morgan Kaufmann, 2004.
65. Sieg, A., Mobasher, B., and Burke, R., Representing context in web search with ontological user profiles. In *Proceedings of the 6th International Conference on Modeling and Using Context*, 2007.
66. Stefanidis, K., Pitoura, E., and Vassiliadis, P., A context-aware preference database system. *International Journal of Pervasive Computing and Communications*, 3(4):439–600, 2007.
67. Umyarov, A., and Tuzhilin, A., Using external aggregate ratings for improving individual recommendations. *ACM Transactions on the Web*, 2010.
68. van Setten, M., Pokraev, S., and Koolwaaij, J., Context-aware recommendations in the mobile tourist application compass. In Nejdl, W., and De, P., Bra, editors, *Adaptive Hypermedia*, pages 235–244. Springer Verlag, 2004.
69. Ward, A., Jones, A., and Hopper, A., A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, 1997.
70. Webster, N., *Webster's new twentieth century dictionary of the English language*. Springfield, MA: Merriam-Webster, Inc., 1980.
71. Woerndl, W., Schueller, C., and Wojtech, R., A hybrid recommender system for context-aware recommendations of mobile applications. In *Proceedings of the 3rd International Workshop on Web Personalization, Recommender Systems and Intelligent User Interfaces*, pages 871–878, 2007.
72. Yu, Z., Zhou, X., Zhang, D., Chin, C.Y., Wang, X., and Men, J., Supporting context-aware media recommendations for smart phones. *IEEE Pervasive Computing*, 5(3):68–75, 2006.
73. Ziegler, C.N., S. M. McNee, Konstan, J.A., and Lausen, G., Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. Chiba, Japan, 2005.

Part II
Applications and Evaluation of RSs

Chapter 8

Evaluating Recommendation Systems

Guy Shani and Asela Gunawardana

Abstract *Recommender systems* are now popular both commercially and in the research community, where many approaches have been suggested for providing recommendations. In many cases a system designer that wishes to employ a recommendation system must choose between a set of candidate approaches. A first step towards selecting an appropriate algorithm is to decide which properties of the application to focus upon when making this choice. Indeed, recommendation systems have a variety of properties that may affect user experience, such as accuracy, robustness, scalability, and so forth. In this paper we discuss how to compare recommenders based on a set of properties that are relevant for the application. We focus on comparative studies, where a few algorithms are compared using some evaluation metric, rather than absolute benchmarking of algorithms. We describe experimental settings appropriate for making choices between algorithms. We review three types of experiments, starting with an offline setting, where recommendation approaches are compared without user interaction, then reviewing user studies, where a small group of subjects experiment with the system and report on the experience, and finally describe large scale online experiments, where real user populations interact with the system. In each of these cases we describe types of questions that can be answered, and suggest protocols for experimentation. We also discuss how to draw trustworthy conclusions from the conducted experiments. We then review a large set of properties, and explain how to evaluate systems given relevant properties. We also survey a large set of evaluation metrics in the context of the properties that they evaluate.

Guy Shani

Microsoft Research, One Microsoft Way, Redmond, WA, e-mail: guyshani@microsoft.com

Asela Gunawardana

Microsoft Research, One Microsoft Way, Redmond, WA, e-mail: aselag@microsoft.com

8.1 Introduction

Recommender systems can now be found in many modern applications that expose the user to a huge collections of items. Such systems typically provide the user with a list of recommended items they might prefer, or predict how much they might prefer each item. These systems help users to decide on appropriate items, and ease the task of finding preferred items in the collection.

For example, the DVD rental provider Netflix¹ displays predicted ratings for every displayed movie in order to help the user decide which movie to rent. The online book retailer Amazon² provides average user ratings for displayed books, and a list of other books that are bought by users who buy a specific book. Microsoft provides many free downloads for users, such as bug fixes, products and so forth. When a user downloads some software, the system presents a list of additional items that are downloaded together. All these systems are typically categorized as recommender systems, even though they provide diverse services.

In the past decade, there has been a vast amount of research in the field of recommender systems, mostly focusing on designing new algorithms for recommendations. An application designer who wishes to add a recommendation system to her application has a large variety of algorithms at her disposal, and must make a decision about the most appropriate algorithm for her goals. Typically, such decisions are based on experiments, comparing the performance of a number of candidate recommenders. The designer can then select the best performing algorithm, given structural constraints such as the type, timeliness and reliability of availability data, allowable memory and CPU footprints. Furthermore, most researchers who suggest new recommendation algorithms also compare the performance of their new algorithm to a set of existing approaches. Such evaluations are typically performed by applying some evaluation metric that provides a ranking of the candidate algorithms (usually using numeric scores).

Initially most recommenders have been evaluated and ranked on their prediction power — their ability to accurately predict the user's choices. However, it is now widely agreed that accurate predictions are crucial but insufficient to deploy a good recommendation engine. In many applications people use a recommendation system for more than an exact anticipation of their tastes. Users may also be interested in discovering new items, in rapidly exploring diverse items, in preserving their privacy, in the fast responses of the system, and many more properties of the interaction with the recommendation engine. We must hence identify the set of properties that may influence the success of a recommender system in the context of a specific application. Then, we can evaluate how the system preforms on these relevant properties.

In this paper we review the process of evaluating a recommendation system. We discuss three different types of experiments; offline, user studies and online experiments.

¹ www.Netflix.com

² www.amazon.com

Often it is easiest to perform offline experiments using existing data sets and a protocol that models user behavior to estimate recommender performance measures such as prediction accuracy. A more expensive option is a user study, where a small set of users is asked to perform a set of tasks using the system, typically answering questions afterwards about their experience. Finally, we can run large scale experiments on a deployed system, which we call online experiments. Such experiments evaluate the performance of the recommenders on real users who are oblivious to the conducted experiment. We discuss what can and cannot be evaluated for each of these types of experiments.

We can sometimes evaluate how well the recommender achieves its overall goals. For example, we can check an e-commerce website revenue with and without the recommender system and thereby estimate the value of the system to the website. In other cases, it can also be useful to evaluate how recommenders perform in terms of some specific properties, allowing us to focus on improving properties that fall short. First, one must show that a property is indeed relevant to users and affect their experience. Then, we can design algorithms that improve upon these properties. In improving one property we may reduce the quality of another property, creating a trade-off between a set of properties. In many cases it is also difficult to say how these trade-offs affect the overall performance of the system, and we have to either run additional experiments to understand this aspect, or use the opinions of domain experts.

This paper focuses on property-directed evaluation of recommender algorithms. We provide an overview of a large set of properties that can be relevant for system success, explaining how candidate recommenders can be ranked with respect to these properties. For each property we discuss the relevant experiment types—offline, user study, and online experiments—and explain how an evaluation can be conducted in each case. We explain the difficulties and outline the pitfalls in evaluating each property. For all these properties we focus on ranking recommenders on that property, assuming that better handling the property will improve user experience.

We also review a set of previous suggestions for evaluating recommendation systems, describing a large set of popular methods and placing them in the context of the properties that they measure. We especially focus on the widely researched accuracy and ranking measurements, describing a large set of evaluation metrics for these properties. For other, less studied properties, we suggest guidelines from which specific measures can be derived. We provide examples of such specific implementations where appropriate.

The rest of the paper is structured as follows. In Section 8.2 we discuss the different experimental settings in which recommender systems can be evaluated, discussing the appropriate use of offline experiments, user studies, and online trials. We also outline considerations that go into making reliable decisions based on these experiments, including generalization and statistical significance of results. In Section 8.3 we describe a large variety of properties of recommendation systems that may impact their performance, as well as metrics for measuring these properties. Finally, we conclude in Section 8.4.

8.2 Experimental Settings

In this section we describe three levels of experiments that can be used in order to compare several recommenders. The discussion below is motivated by evaluation protocols in related areas such as machine learning and information retrieval, highlighting practices relevant to evaluating recommendation systems. The reader is referred to publications in these fields for more detailed discussions [49, 13, 60].

We begin with offline experiments, which are typically the easiest to conduct, as they require no interaction with real users. We then describe user studies, where we ask a small group of subjects to use the system in a controlled environment, and then report on their experience. In such experiments we can collect both quantitative and qualitative information about the systems, but care must be taken to consider various biases in the experimental design. Finally, perhaps the most trustworthy experiment is when the system is used by a pool of real users, typically unaware of the experiment. While in such an experiment we are able to collect only certain types of data, this experimental design is closest to reality.

In all experimental scenarios, it is important to follow a few basic guidelines in general experimental studies:

- **Hypothesis:** before running the experiment we must form an hypothesis. It is important to be concise and restrictive about this hypothesis, and design an experiment that tests the hypothesis. For example, an hypothesis can be that algorithm *A* better predicts user ratings than algorithm *B*. In that case, the experiment should test the prediction accuracy, and not other factors.
- **Controlling variables:** when comparing a few candidate algorithms on a certain hypothesis, it is important that all variables that are not tested will stay fixed. For example, suppose that we wish to compare the prediction accuracy of movie ratings of algorithm *A* and algorithm *B*, that both use different collaborative filtering models. If we train *A* on the MovieLens data set, and *B* on the Netflix data set, and algorithm *A* presents superior performance, we can not tell whether the performance was due to the superior CF model, or due to the better input data, or both. We therefore must train the algorithms on the same data set (or over unbiased samples from the same data set), or train the same algorithms over the two different data sets, in order to understand the cause of the superior performance.
- **Generalization power:** when drawing conclusions from experiments, we may desire that our conclusions generalize beyond the immediate context of the experiments. When choosing an algorithm for a real application, we may want our conclusions to hold on the deployed system, and generalize beyond our experimental data set. Similarly, when developing new algorithms, we want our conclusions to hold beyond the scope of the specific application or data set that we experimented with. To increase the probability of generalization of the results we must typically experiment with several data sets or applications. It is important to understand the properties of the various data sets that are used.

Generally speaking, the more diverse the data used, the more we can generalize the results.

8.2.1 Offline Experiments

An offline experiment is performed by using a pre-collected data set of users choosing or rating items. Using this data set we can try to simulate the behavior of users that interact with a recommendation system. In doing so, we assume that the user behavior when the data was collected will be similar enough to the user behavior when the recommender system is deployed, so that we can make reliable decisions based on the simulation. Offline experiments are attractive because they require no interaction with real users, and thus allow us to compare a wide range of candidate algorithms at a low cost. The downside of offline experiments is that they can answer a very narrow set of questions, typically questions about the prediction power of an algorithm. In particular, we must assume that users' behavior when interacting with a system including the recommender system chosen will be modeled well by the users' behavior prior to that system's deployment. Thus we cannot directly measure the recommender's influence on user behavior in this setting.

Therefore, the goal of the offline experiments is to filter out inappropriate approaches, leaving a relatively small set of candidate algorithms to be tested by the more costly user studies or online experiments. A typical example of this process is when the parameters of the algorithms are tuned in an offline experiment, and then the algorithm with the best tuned parameters continues to the next phase.

8.2.1.1 Data sets for offline experiments

As the goal of the offline evaluation is to filter algorithms, the data used for the offline evaluation should match as closely as possible the data the designer expects the recommender system to face when deployed online. Care must be exercised to ensure that there is no bias in the distributions of users, items and ratings selected. For example, in cases where data from an existing system (perhaps a system without a recommender) is available, the experimenter may be tempted to pre-filter the data by excluding items or users with low counts, in order to reduce the costs of experimentation. In doing so, the experimenter should be mindful that this involves a trade-off, since this introduces a systematic bias in the data. If necessary, randomly sampling users and items may be a preferable method for reducing data, although this can also introduce other biases into the experiment (e.g. this could tend to favor algorithms that work better with more sparse data). Sometimes, known biases in the data can be corrected for by techniques such as reweighing data, but correcting biases in the data is often difficult.

Another source of bias may be the data collection itself. For example, users may be more likely to rate items that they have strong opinions on, and some users may

provide many more ratings than others. Thus, the set of items on which explicit ratings are available may be biased by the ratings themselves [38]. Once again, techniques such as resampling or reweighting the test data may be used to attempt to correct such biases.

8.2.1.2 Simulating user behavior

In order to evaluate algorithms offline, it is necessary to simulate the online process where the system makes predictions or recommendations, and the user corrects the predictions or uses the recommendations. This is usually done by recording historical user data, and then hiding some of these interactions in order to simulate the knowledge of how a user will rate an item, or which recommendations a user will act upon. There are a number of ways to choose the ratings/selected items to be hidden. Once again, it is preferable that this choice be done in a manner that simulates the target application as closely as possible. In many cases, though, we are restricted by the computational cost of an evaluation protocol, and must make compromises in order to execute the experiment over large data sets.

Ideally, if we have access to time-stamps for user selections, we can simulate what the systems predictions would have been, had it been running at the time the data set was collected. We can begin with no available prior data for computing predictions, and step through user selections in temporal order, attempting to predict each selection and then making that selection available for use in future predictions. For large data sets, a simpler approach is to randomly sample test users, randomly sample a time just prior to a user action, hide all selections (of all users) after that instant, and then attempt to recommend items to that user. This protocol requires changing the set of given information prior to each recommendation, which can still be computationally quite expensive.

An even cheaper alternative is to sample a set of test users, then sample a single test time, and hide all items after the sampled test time for each test user. This simulates a situation where the recommender system is built as of the test time, and then makes recommendations without taking into account any new data that arrives after the test time. Another alternative is to sample a test time for each test user, and hide the test user's items after that time, without maintaining time consistency across users. This effectively assumes that the sequence in which items are selected is important, not the absolute times when the selections are made. A final alternative is to ignore time. We would first sample a set of test users, then sample the number n_a of items to hide for each user a , and finally sample n_a items to hide. This assumes that the temporal aspects of user selections are unimportant. We may be forced to make this assumption if the timestamps of user actions are not known. All three of the latter alternatives partition the data into a single training set and single test set. It is important to select an alternative that is most appropriate for the domain and task of interest (see Chapter 11), given the constraints, rather than the most convenient one.

A common protocol used in many research papers is to use a fixed number of known items or a fixed number of hidden items per test user (so called “given n ” or “all but n ” protocols). This protocol is useful for diagnosing algorithms and identifying in which cases they work best. However, when we wish to make decisions on the algorithm that we will use in our application, we must ask ourselves whether we are truly interested in presenting recommendations only for users who have rated exactly n items, or are expected to rate exactly n items more. If that is not the case, then results computed using these protocols have biases that make them unreliable in predicting the performance of the algorithms online.

8.2.1.3 More complex user modeling

All the protocols that we discuss above make some assumptions concerning the behavior of users, which could be regarded as a user-model for the specific application. While we discuss only very simple user-models it is possible to suggest more complicated models for user behavior [37]. Using advanced user models we can execute simulations of users interactions with the system, thus reducing the need for expensive user studies and online testing. However, care must be made when designing user-models; First, user-modeling is a difficult task, and there is a vast amount of research on the subject (see, e.g. [15]). Second, when the user model is inaccurate, we may optimize a system whose performance in simulation has no correlation with its performance in practice. While it is reasonable to design an algorithm that uses complex user-models to provide recommendations, we should be careful in trusting experiments where algorithms are verified using such complex, difficult to verify user models.

8.2.2 User Studies

Many recommendation approaches rely on the interaction of users with the system (see, e.g., Chapters 23, 13, 6). It is very difficult to create a reliable simulation of users interactions with the system, and thus, offline testing are difficult to conduct. In order to properly evaluate such systems, real user interactions with the system must be collected. Even when offline testing is possible, interactions with real users can still provide additional information about the system performance. In these cases we typically conduct user studies.

A user study is conducted by recruiting a set of test subjects, and asking them to perform several tasks requiring an interaction with the recommendation system. While the subjects perform the tasks, we observe and record their behavior, collecting any number of quantitative measurements, such as what portion of the task was completed, the accuracy of the task results, or the time taken to perform the task. In many cases we can ask qualitative questions before, during, and after the task is completed. Such questions can collect data that is not directly observable, such

as whether the subject enjoyed the user interface, or whether the user perceived the task as easy to complete.

A typical example of such an experiment is to test the influence of a recommendation algorithm on the browsing behavior of news stories. In this example, the subjects are asked to read a set of stories that are interesting to them, in some cases including related story recommendations and in some cases without recommendations. We can then check whether the recommendations are used, and whether people read different stories with and without recommendations. We can collect data such as how many times a recommendation was clicked, and even, in certain cases, track eye movement to see whether a subject looked at a recommendation. Finally, we can ask qualitative questions such as whether the subject thought the recommendations were relevant.

Of course, in many other research areas user studies are a central tool, and thus there is much literature on the proper design of user studies. This section only overviews the basic considerations that should be taken when evaluating a recommender system through a user study, and the interested reader can find much deeper discussions elsewhere (see. e.g. [5]).

8.2.2.1 Advantages and Disadvantages

User studies can perhaps answer the widest set of questions of all three experimental settings that we survey here. Unlike offline experiments this setting allows us to test the behavior of users when interacting with the recommendation system, and the influence of the recommendations on user behavior. In the offline case we typically make assumptions such as “given a relevant recommendation the user is likely to use it” which are tested in the user study. Second, this is the only setting that allows us to collect qualitative data that is often crucial for interpreting the quantitative results. Also, we can typically collect in this setting a large set of quantitative measurements because the users can be closely monitored while performing the tasks.

User studies however have some disadvantages. Primarily, user studies are very expensive to conduct; collecting a large set of subjects and asking them to perform a large enough set of tasks is costly in terms of either user time, if the subjects are volunteers, or in terms of compensation if paid subjects are employed. Therefore, we must typically restrict ourselves to a small set of subjects and a relatively small set of tasks, and cannot test all possible scenarios. Furthermore, each scenario has to be repeated several time in order to make reliable conclusions, further limiting the range of distinct tasks that can be tested.

As these experiments are expensive to conduct we should collect as much data about the user interactions, in the lowest possible granularity. This will allow us later to study the results of the experiment in detail, analyzing considerations that were not obvious prior to the trial. This guideline can help us to reduce the need for successive trials to collect overlooked measurements.

Furthermore, in order to avoid failed experiments, such as applications that malfunction under certain user actions, researchers often execute pilot user studies.

These are small scale experiments, designed not to collect statistical data, but to test the systems for bugs and malfunctions. In some cases, the results of these pilot studies are then used to improve the recommender. If this is the case, then the results of the pilot become “tainted”, and should not be used when computing measurements in the final user study.

Another important consideration is that the test subjects must represent as closely as possible the population of users of the real system. For example, if the system is designed to recommend movies, the results of a user study over avid movie fans may not carry to the entire population. This problem is most persistent when the participants of the study are volunteers, as in this case people who are originally more interested in the application may tend to volunteer more readily.

However, even when the subjects represent properly the true population of users, the results can still be biased because they are aware that they are participating in an experiment. For example, it is well known that paid subjects tend to try and satisfy the person or company conducting the experiment. If the subjects are aware of the hypothesis that is tested they may unconsciously provide evidence that supports it. To accommodate that, it is typically better not to disclose the goal of the experiment prior to collecting data. Another, more subtle effect occurs when the payment to subjects takes the form of a complete or partial subsidy of items they select. This may bias the data in cases where final users of the system are not similarly subsidized, as users’ choices and preferences may be different when they pay full price.

8.2.2.2 Between vs. Within Subjects

As typically a user study compares a few candidate approaches, each candidate must be tested over the same tasks. To test all candidates we can either compare the candidates *between subjects*, where each subject is assigned to a candidate method and experiments with it, or *within subjects*, where each subject tests a set of candidates on different tasks [20].

Typically, within subjects experiments are more informative, as the superiority of one method cannot be explained by a biased split of users between candidate methods. It is also possible in this setting to ask comparative questions about the different candidates, such as which candidate the subject preferred. However, in these types of tests users are more conscious of the experiment, and hiding the distinctions between candidates is more difficult.

Between subjects experiments, also known as *A-B testing* (All Between), provide a setting that is closer to the real system, as each user experiments with a single system. Such experiments can also test long term effects of using the system, because the user is not required to switch systems. Thus we can test how the user becomes accustomed to the system, and estimate a learning curve of expertise.

8.2.2.3 Variable Counter Balance

As we have noted above, it is important to control all variables that are not specifically tested. However, when a subject is presented with the output of several candidates, as is done in within subject experiments, we must counter balance several variables.

When presenting several results to the subject, the results can be displayed either sequentially, or together. In both cases there are certain biases that we need to correct for. When presenting the results sequentially the previously observed results influence the user opinion of the current results. For example, if the results that were displayed first seem inappropriate, the results displayed afterwards may seem better than they actually are. When presenting two sets of results, there can be certain biases due to location. For example, users from many cultures tend to observe results left to right and top to bottom. Thus, the user may observe the results displayed on top as superior.

A common approach to correct for such untested variables is by using the *Latin square* [5] procedure. This procedure randomizes the order or location of the various results each time, thus canceling out biases due to these untested variables.

8.2.2.4 Questionnaires

User studies allow us to use the powerful questionnaire tool. Prior, during, and after subjects perform their tasks we can ask them questions about their experience. These questions can provide information about properties that are difficult to measure, such as the subject's state of mind, or whether the subject enjoyed the system.

While these questions can provide valuable information, they can also provide misleading information. It is important to ask neutral questions, that do not suggest a "correct" answer. People may also answer untruthfully, for example when they perceive the answer as private, or if they think the true answer may put them in an unflattering position.

Indeed, vast amount of research was conducted in other areas about the art of questionnaire writing, and we refer the readers to that literature (e.g. [46]) for more details.

8.2.3 Online Evaluation

In many realistic recommendation applications the designer of the system wishes to influence the behavior of users. We are therefore interested in measuring the change in user behavior when interacting with different recommendation systems. For example, if users of one system follow the recommendations more often, or if some utility gathered from users of one system exceeds utility gathered from users of the

other system, then we can conclude that one system is superior to the other, all else being equal.

The real effect of the recommendation system depends on a variety of factors such as the user's intent (e.g. how specific their information needs are, how much novelty vs. how much risk they are seeking), the user's context (e.g. what items they are already familiar with, how much they trust the system), and the interface through which the recommendations are presented.

Thus, the experiment that provides the strongest evidence as to the true value of the system is an online evaluation, where the system is used by real users that perform real tasks. It is most trustworthy to compare a few systems online, obtaining a ranking of alternatives, rather than absolute numbers that are more difficult to interpret.

For this reason, many real world systems employ an online testing system [32], where multiple algorithms can be compared. Typically, such systems redirect a small percentage of the traffic to different alternative recommendation engine, and record the users interactions with the different systems.

There are a few considerations that must be made when running such tests. For example, it is important to sample (redirect) users randomly, so that the comparisons between alternatives are fair. It is also important to single out the different aspects of the recommenders. For example, if we care about algorithmic accuracy, it is important to keep the user interface fixed. On the other hand, if we wish to focus on a better user interface, it is best to keep the underlying algorithm fixed.

In some cases, such experiments are risky. For example, a test system that provides irrelevant recommendations, may discourage the test users from using the real system ever again. Thus, the experiment can have a negative effect on the system, which may be unacceptable in commercial applications.

For these reasons, it is best to run an online evaluation last, after an extensive offline study provides evidence that the candidate approaches are reasonable, and perhaps after a user study that measures the user's attitude towards the system. This gradual process reduces the risk in causing significant user dissatisfaction.

Online evaluations are unique in that they allow direct measurement of overall system goals, such as long-term profit or user retention. As such, they can be used to understand how these overall goals are affected by system properties such as recommendation accuracy and diversity of recommendations, and to understand the trade-offs between these properties. However, since varying such properties independently is difficult, and comparing many algorithms through online trials is expensive, it can be difficult to gain a complete understanding of these relationships.

8.2.4 Drawing Reliable Conclusions

In any type of experiment it is important that we can be confident that the candidate recommender that we choose will also be a good choice for the yet unseen data the system will be faced with in the future. As we explain above, we should exercise

caution in choosing the data in an offline experiments, and the subjects in a user study, to best resemble the online application. Still, there is a possibility that the algorithm that performed best on this test set did so because the experiment was fortuitously suitable for that algorithm. To reduce the possibility of such statistical mishaps, we must perform significance testing on the results.

8.2.4.1 Confidence and p -values

A standard tool for significance testing is a significance level or p -value — the probability that the obtained results were due to luck. Generally, we will reject the null hypothesis that algorithm A is no better than algorithm B if the p -value is above some threshold. That is, if the probability that the observed ranking is achieved by chance exceeds the threshold, then the results of the experiment are not deemed significant. Traditionally, people choose $p = 0.05$ as their threshold, which indicates less than 95% confidence. More stringent significance levels (e.g. 0.01 or even lower) can be used in cases where the cost of making the wrong choice is higher.

8.2.4.2 Paired Results

In order to perform a significance test that algorithm A is indeed better than algorithm B , we often use the results of several independent experiments comparing A and B . Indeed, the protocol we have suggested for generating our test data (Section 8.2.1.2) allows us to obtain such a set of results. Assuming that test users are drawn independently from some population, the performance measures of the algorithms for each test user give us the independent comparisons we need. However, when recommendations or predictions of multiple items are made to the same user, it is unlikely that the resulting per-item performance metrics are independent. Therefore, it is better to compare algorithms on a per-user basis.

Given such paired per-user performance measures for algorithms A and B a simple test of significance is the **sign test** [13]. We count the number of users for whom algorithm A outperforms algorithm B (n_A) and the number of users for whom algorithm B outperforms algorithm A (n_B). The significance level is the probability that A is not truly better than B , and is estimated as the probability of at least n_A out of $n = n_A + n_B$ 0.5-probability Binomial trials succeeding (i.e. n_A out of $n_A + n_B$ fair coin-flips coming up “heads”), and is given by

$$p = (0.5)^n \sum_{i=n_A}^n \frac{n!}{i!(n-i)!} \quad (8.1)$$

The sign test is an attractive choice due to its simplicity, and lack of assumptions over the distribution of cases. While tied results are traditionally ignored, Demšar [13] recommends splitting them between A and B , since they provide evidence for the null hypothesis that the algorithms are no different. When $n_A + n_B$ is large, we

can take advantage of large sample theory to approximate (8.1) by a normal distribution. However, this is usually unnecessary with powerful modern computers. Some authors (e.g. [49]) use the term **McNemar's test** to refer to the use of a χ^2 approximation to the two-sided sign test.

Note that sometimes, the sign test may indicate that system A outperforms system B with high probability, even though the average performance of system B is higher than that of system A. This happens in cases where system B occasionally outperforms system A overwhelmingly. Thus, the reason for this seemingly inconsistent result is that the test only examines the probability of one system outperforming the other, without regard to the magnitude of the difference.

The sign test can be extended to cases where we want to know the probability that one system outperforms the other by some amount. For example, suppose that system A is much more resource intensive than system B, and is only worth deploying if it outperforms system B by some amount. We can define "success" in the sign test as A outperforming B by this amount, and find the probability of A not truly outperforming B by this amount as our p value in equation (8.1).

A commonly used test that takes the magnitude of the differences into account is the **paired Student's t-test**, which looks at the average difference between the performance scores of algorithms A and B , normalized by the standard deviation of the score difference. Using this test requires that the differences in scores for different users is comparable, so that averaging these differences is reasonable. For small numbers of users, the validity of the test also depends on the differences being Normally distributed. Demšar [13] points out that this assumption is hard to verify when the number of samples is small and that the t-test is susceptible to outliers. He recommends the use of **Wilcoxon signed rank test**, which like the t-test, uses the magnitude of the differences between algorithms A and B , but without making distributional assumptions on the differences. However, using the Wilcoxon signed rank test still requires that differences between the two systems are comparable between users.

Another way to improve the significance of our conclusions is to use a larger test set. In the offline case, this may require using a smaller training set, which may result in an experimental protocol that is not representative of the amount of training data available after deployment. In the case of user studies, this implies an additional expense. In the case of online testing, increasing the amount of data collected for each algorithm requires either the added expense of a longer trial or the comparison of fewer algorithms.

8.2.4.3 Unpaired Results

The tests described above are suitable for cases where observations are paired. That is, each algorithm is run on each test case, as is often done in offline tests. In online tests, however, it is often the case that users are assigned to one algorithm or the other, so that the two algorithms are not evaluated on the same test cases. The **Mann-**

Whitney test is an extension of the Wilcoxon test to this scenario. Suppose we have n_A results from algorithm A and n_B results from algorithm B.

The performance measures of the two algorithms are pooled and sorted so that the best result is ranked first and the worst last. The ranks of ties are averaged. For example if the second through fifth place tie, they are all assigned a rank of 3.5. The Mann-Whitney test computes the probability of the null hypothesis that n_A randomly chosen results from the total $n_A + n_B$ have at least as good an average rank as the n_A results that came from algorithm A.

This probability can be computed exactly by enumerating all $\frac{(n_A+n_B)!}{n_A!n_B!}$ choices and counting the choices that have at least the required average rank, or can be approximated by repeatedly resampling n_A of the results. When n_A and n_B are both large enough (typically over 5), the distribution of the average rank of n_A results randomly selected from a pool of $n_A + n_B$ under the null hypothesis is well approximated by a Gaussian with mean $\frac{1}{2}(n_A + n_B + 1)$ and standard deviation $\sqrt{\frac{1}{12} \frac{n_A}{n_B} (n_A + n_B + 1)}$. Thus, in this case we can compute the average rank of the n_A results from system A, subtract $\frac{1}{2}(n_A + n_B + 1)$, divide by $\sqrt{\frac{1}{12} \frac{n_A}{n_B} (n_A + n_B + 1)}$, and evaluate the standard Gaussian CDF at this value to get the p value for the test.

8.2.4.4 Multiple tests

Another important consideration, mostly in the offline scenario, is the effect of evaluating multiple versions of algorithms. For example, an experimenter might try out several variants of a novel recommender algorithm and compare them to a baseline algorithm until they find one that passes a sign test at the $p = 0.05$ level and therefore infer that their algorithm improves upon the baseline with 95% confidence. However, this is not a valid inference. Suppose the experimenter evaluated 10 different variants all of which are statistically the same as the baseline. If the probability that any one of these trials passes the sign test mistakenly is $p = 0.05$, the probability that at least one of the ten trials passes the sign test mistakenly is $1 - (1 - 0.05)^{10} = 0.40$. This risk is colloquially known as “tuning to the test set” and can be avoided by separating the test set users into two groups—a development (or tuning) set, and an evaluation set. The choice of algorithm is done based on the development test, and the validity of the choice is measured by running a significance test on the evaluation set.

A similar concern exists when ranking a number of algorithms, but is more difficult to circumvent. Suppose the best of $N + 1$ algorithms is chosen on the development test set. To achieve a confidence $1 - p$ that the chosen algorithm is indeed the best, it must outperform the N other algorithms on the evaluation set with significance $1 - (1 - p)^{1/N}$. This is known as the Bonferroni correction, and should be used when pair-wise significant tests are used multiple times. Alternatively, approaches such as ANOVA or the **Friedman test for ranking**, which are generalization of the Student’s t-test and Wilcoxon’s rank test. ANOVA makes strong assumptions about the Normality of the different algorithms’ performance measures, and about the re-

relationships of their variances. We refer the reader to [13] for further discussion of these and other tests for ranking multiple algorithms.

A more subtle version of this concern is when a pair of algorithms are compared in a number of ways. For example, two algorithms may be compared using a number of accuracy measures, a number of coverage measures, etc. Even if the two algorithms are identical in all measures, the probability of finding a measure by which one algorithm seems to outperform the other with some significance level increases with the number of measures examined. If the different measures are independent, the Bonferroni correction mentioned above can be used. However, since the measures are often correlated, the Bonferroni correction may be too stringent, and other approaches such as controlling for false discovery rate [2] may be used.

8.2.4.5 Confidence Intervals

Even though we focus here on comparative studies, where one has to choose the most appropriate algorithm out of a set of candidates, it is sometimes desirable to measure the value of some property. For example, an administrator may want to estimate the error in the system predictions, or the net profit that the system is earning. When measuring such quantities it is important to understand the reliability of your estimates. A popular approach for doing this is to compute **confidence intervals**.

For example, one may estimate that the RMSE of a system is expected to be 1.2, and that it will be between 1.1 and 1.35 with probability 0.95. The simplest method for computing confidence intervals is to assume that the quantity of interest is Gaussian distributed, and then estimate its mean and standard deviations from multiple independent observations. When we have many observations, we can dispense with this assumption by computing the distribution of the quantity of interest with a non-parametric method such as a histogram and finding upper and lower bounds such that include the quantity of interest with the desired probability.

8.3 Recommendation System Properties

In this section we survey a range of properties that are commonly considered when deciding which recommendation approach to select. As different applications have different needs, the designer of the system must decide on the important properties to measure for the concrete application at hand. Some of the properties can be traded-off, the most obvious example perhaps is the decline in accuracy when other properties (e.g. diversity) are improved. It is important to understand and evaluate these trade-offs and their effect on the overall performance. However, the proper way of gaining such understanding without intensive online testing or deferring to the opinions of domain experts is still an open question.

Furthermore, the effect of many of these properties on the user experience is unclear, and depends on the application. While we can certainly speculate that users

would like diverse recommendations or reported confidence bounds, it is essential to show that this is indeed important in practice. Therefore, when suggesting a method that improves one of these properties, one should also evaluate how changes in this property affect the user experience, either through a user study or through online experimentation.

Such an experiment typically uses a single recommendation method with a tunable parameter that affects the property being considered. For example, we can envision a parameter that controls the diversity of the list of recommendations. Then, subjects should be presented with recommendations based on a variety of values for this parameter, and we should measure the effect of the parameter on the user experience. We should measure here not whether the user noticed the change in the property, but whether the change in property has affected their interaction with the system. As is always the case in user studies, it is preferable that the subjects in a user study and users in an online experiment will not know the goal of the experiment. It is difficult to envision how this procedure could be performed in an offline setting because we need to understand the user response to this parameter.

Once the effects of the specific system properties in affecting the user experience of the application at hand are understood, we can use differences in these properties to select a recommender.

8.3.1 User Preference

As in this paper we are interested in the selection problem, where we need to choose one out of a set of candidate algorithms, an obvious option is to run a user study (within subjects) and ask the participants to choose one of the systems [25]. This evaluation does not restrict the subjects to specific properties, and it is generally easier for humans to make such judgments than to give scores for the experience. Then, we can select the system that had the largest number of votes.

However, aside from the biases in user studies discussed earlier, there are additional concerns that we must be aware of. First, the above scheme assumes that all users are equal, which may not always be true. For example, an e-commerce website may prefer the opinion of users who buy many items to the opinion of users who only buy a single item. We therefore need to further weight the vote by the importance of the user, when applicable. Assigning the right importance weights in a user study may not be easy.

It may also be the case that users who preferred system *A*, only slightly preferred it, while users who preferred *B*, had a very low opinion of *A*. In this case, even if more users preferred *A* we may still wish to choose *B*. To measure this we need non-binary answers for the preference question in the user study. Then, the problem of calibrating scores across users arises.

Finally, when we wish to improve a system, it is important to know why people favor one system over the other. Typically, it is easier to understand that when comparing specific properties. Therefore, while user satisfaction is important to mea-

sure, breaking satisfaction into smaller components is helpful to understand the system and improve it.

8.3.2 Prediction Accuracy

Prediction accuracy is by far the most discussed property in the recommendation system literature. At the base of the vast majority of recommender systems lie a prediction engine. This engine may predict user opinions over items (e.g. ratings of movies) or the probability of usage (e.g. purchase).

A basic assumption in a recommender system is that a system that provides more accurate predictions will be preferred by the user. Thus, many researchers set out to find algorithms that provide better predictions.

Prediction accuracy is typically independent of the user interface, and can thus be measured in an offline experiment. Measuring prediction accuracy in a user study measures the accuracy given a recommendation. This is a different concept from the prediction of user behavior without recommendations, and is closer to the true accuracy in the real system.

We discuss here three broad classes of prediction accuracy measures; measuring the accuracy of ratings predictions, measuring the accuracy of usage predictions, and measuring the accuracy of rankings of items.

8.3.2.1 Measuring Ratings Prediction Accuracy

In some applications, such as in the new releases page of the popular Netflix DVD rental service, we wish to predict the rating a user would give to an item (e.g. 1-star through 5-stars). In such cases, we wish to measure the accuracy of the system's predicted ratings.

Root Mean Squared Error (RMSE) is perhaps the most popular metric used in evaluating accuracy of predicted ratings. The system generates predicted ratings \hat{r}_{ui} for a test set \mathcal{T} of user-item pairs (u, i) for which the true ratings r_{ui} are known. Typically, r_{ui} are known because they are hidden in an offline experiment, or because they were obtained through a user study or online experiment. The RMSE between the predicted and actual ratings is given by:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} (\hat{r}_{ui} - r_{ui})^2} \quad (8.2)$$

Mean Absolute Error (MAE) is a popular alternative, given by

$$\text{MAE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i) \in \mathcal{T}} |\hat{r}_{ui} - r_{ui}|} \quad (8.3)$$

Compared to MAE, RMSE disproportionately penalizes large errors, so that, given a test set with four hidden items RMSE would prefer a system that makes an error of 2 on three ratings and 0 on the fourth to one that makes an error of 3 on one rating and 0 on all three others, while MAE would prefer the second system.

Normalized RMSE (NMRSE) and **Normalized MAE (NMAE)** are versions of RMSE and MAE that have been normalized by the range of the ratings (i.e. $r_{max} - r_{min}$). Since they are simply scaled versions of RMSE and MAE, the resulting ranking of algorithms is the same as the ranking given by the unnormalized measures.

Average RMSE and **Average MAE** adjust for unbalanced test sets. For example, if the test set has an unbalanced distribution of items, the RMSE or MAE obtained from it might be heavily influenced by the error on a few very frequent items. If we need a measure that is representative of the prediction error on any item, it is preferable to compute MAE or RMSE separately for each item and then take the average over all items. Similarly, one can compute a per-user average RMSE or MAE if the test set has an unbalanced user distribution and we wish to understand the prediction error a randomly drawn user might face.

RMSE and MAE depend only on the magnitude of the errors made. In some applications, the semantics of the ratings may be such that the impact of a prediction error does not depend only on its magnitude. In such domains it may be preferable to use a suitable distortion measure $d(\hat{r}, r)$ than squared difference or absolute difference. For example in an application with a 3-star rating system where 1 means “disliked,” 2 means “neutral” and 3 means “liked,” and where recommending an item the user dislikes is worse than not recommending an item a user likes, a distortion measure with $d(3, 1) = 5$, $d(2, 1) = 3$, $d(3, 2) = 3$, $d(1, 2) = 1$, $d(2, 3) = 1$, and $d(1, 3) = 2$ may be reasonable.

8.3.2.2 Measuring Usage Prediction

In many applications the recommendation system does not predict the user’s preferences of items, such as movie ratings, but tries to recommend to users items that they may use. For example, when movies are added to the queue, Netflix suggests a set of movies that may also be interesting, given the added movie. In this case we are interested not in whether the system properly predicts the ratings of these movies but rather whether the system properly predicts that the user will add these movies to the queue (use the items).

In an offline evaluation of usage prediction, we typically have a data set consisting of items each user has used. We then select a test user, hide some of her selections, and ask the recommender to predict a set of items that the user will use. We then have four possible outcomes for the recommended and hidden items, as shown in Table 8.1.

In the offline case, since the data isn’t typically collected using the recommender system under evaluation, we are forced to assume that unused items would have not been used even if they had they been recommended — i.e. that they are uninteresting

	Recommended	Not recommended
Used	True-Positive (tp)	False-Negative (fn)
Not used	False-Positive (fp)	True-Negative (tn)

Table 8.1: Classification of the possible result of a recommendation of an item to a user.

or useless to the user. This assumption may be false, such as when the set of unused items contains some interesting items that the user did not select. For example, a user may not have used an item because she was unaware of its existence, but after the recommendation exposed that item the user can decide to select it. In this case the number of false positives is over estimated.

We can count the number of examples that fall into each cell in the table and compute the following quantities:

$$\text{Precision} = \frac{\#tp}{\#tp + \#fp} \tag{8.4}$$

$$\text{Recall (True Positive Rate)} = \frac{\#tp}{\#tp + \#fn} \tag{8.5}$$

$$\text{False Positive Rate (1 - Specificity)} = \frac{\#fp}{\#fp + \#tn} \tag{8.6}$$

Typically we can expect a trade off between these quantities — while allowing longer recommendation lists typically improves recall, it is also likely to reduce the precision. In applications where the number of recommendations that can be presented to the user is preordained, the most useful measure of interest is **Precision at N**.

In other applications where the number of recommendations that are presented to the user is not preordained, it is preferable to evaluate algorithms over a range of recommendation list lengths, rather than using a fixed length. Thus, we can compute curves comparing precision to recall, or true positive rate to false positive rate. Curves of the former type are known simply as precision-recall curves, while those of the latter type are known as a Receiver Operating Characteristic³ or ROC curves. While both curves measure the proportion of preferred items that are actually recommended, precision-recall curves emphasize the proportion of recommended items that are preferred while ROC curves emphasize the proportion of items that are not preferred that end up being recommended.

We should select whether to use precision-recall or ROC based on the properties of the domain and the goal of the application; suppose, for example, that an online video rental service recommends DVDs to users. The precision measure describes the proportion of their recommendations were actually suitable for the user. Whether the unsuitable recommendations represent a small or large fraction of the unsuitable DVDs that could have been recommended (i.e. the false positive rate) may not be

³ A reference to their origins in signal detection theory.

as relevant as what proportion of the relevant items the system recommended to the user, so a precision-recall curve would be suitable for this application. On the other hand, consider a recommender system that is used for selecting items to be marketed to users, for example by mailing an item to the user who returns it at no cost to themselves if they do not purchase it. In this case, where we are interested in realizing as many potential sales as possible while minimizing marketing costs, ROC curves would be more relevant than precision-recall curves.

Given two algorithms, we can compute a pair of such curves, one for each algorithm. If one curve completely dominates the other curve, the decision about the superior algorithm is easy. However, when the curves intersect, the decision is less obvious, and will depend on the application in question. Knowledge of the application will dictate which region of the curve the decision will be based on.

Measures that summarize the precision recall of ROC curve such as **F-measure** [58] and the **Area Under the ROC Curve (AUC)** [1] are useful for comparing algorithms independently of application, but when selecting an algorithm for use in a particular task, it is preferable to make the choice based on a measure that reflects the specific needs at hand.

Precision-Recall and ROC for Multiple Users

When evaluating precision-recall or ROC curves for multiple test users, a number of strategies can be employed in aggregating the results, depending on the application at hand.

In applications where a fixed number of recommendations is made to each user (e.g. when a fixed number of headlines are shown to a user visiting a news portal), we can compute the precision and recall (or true positive rate and false positive rate) at each recommendation list length N for each user, and then compute the average precision and recall (or true positive rate and false positive rate) at each N [51]. The resulting curves are particularly valuable because they prescribe a value of N for each achievable precision and recall (or true positive rate and false positive rate), and conversely, can be used to estimate performance at a given N . An ROC curve obtained in this manner is termed a **Customer ROC (CROC) curve** [52].

When different numbers of recommendations can be shown to each user (e.g. when presenting the set of all recommended movies to each user), we can compute ROC or precision-recall curves by aggregating the hidden ratings from the test set into a set of reference user-item pairs, using the recommender system to generate a single ranked list of user-item pairs, picking the top recommendations from the list, and scoring them against the reference set. An ROC curve calculated in this way is termed a **Global ROC (GROC) curve** [52]. Picking an operating point on the resulting curve can result in a different number of recommendations being made to each user.

A final class of applications is where the recommendation process is more interactive, and the user is able to obtain more and more recommendations. This is typical of information retrieval tasks, where the user can keep asking the system for

more recommended documents. In such applications, we compute a precision-recall curve (or ROC curve) for each user and then average the resulting curves over users. This is the usual manner in which precision-recall curves are computed in the information retrieval community, and in particular in the influential TREC competitions [59]. Such a curve can be used to understand the trade-off between precision and recall (or false positives and false negatives) a typical user would face.

8.3.2.3 Ranking Measures

In many cases the application presents to the user a list of recommendations, typically as vertical or horizontal list, imposing a certain natural browsing order. For example, in Netflix, the “movies you’ll love” tab, shows a set of categories, and in each category, a list of movies that the system predicts the user to like. These lists may be long and the user may need to continue to additional “pages” until the entire list is browsed. In these applications, we are not interested in predicting an explicit rating, or selecting a set of recommended items, as in the previous sections, but rather in ordering items according to the user’s preferences. This task is typically known as ranking of items. There are two approaches for measuring the accuracy of such a ranking. We can try to determine the correct order of a set of items for each user and measure how close a system comes to this correct order, or we can attempt to measure the utility of the system’s ranking to a user. We first describe these approaches for offline tests, and then describe their applicability to user studies and online tests.

Using a Reference Ranking

In order to evaluate a ranking algorithm with respect to a reference ranking (a correct order), it is first necessary to obtain such a reference. In cases where explicit user ratings of items are available, we can rank the rated items in decreasing order of the ratings, with ties. For example, in the case of Netflix, movies ranked by a user can be ranked in order of decreasing order of rating, with 5-star movies tied, followed by 4-star movies which are themselves tied, etc. In cases where we only have usage data, it may be appropriate to construct a reference ranking where items used by the user are ranked above unused items. However, this is only valid if we know that the user was aware of the unused items, so that we can infer that the user actually preferred the used items to the unused items. Thus, for example, logs from an online music application such as Pandora⁴ can be used to construct a reference ranking where tracks that a user listened to are ranked above ones they skipped. Once again, used items should be tied with each other and unused items should be tied with each other.

⁴ www.pandora.com

In both kinds of reference rankings above, two items are tied when we have no information about the user's relative ranking of the two. However, a recommender system used in an application such as Netflix's "movies you'll love" needs to strictly rank items with no ties. In such cases, a system under evaluation should not be penalized for ranking one item over another when they are tied in the reference ranking. The **Normalized Distance-based Performance Measure (NDPM)** measure [61] is suitable for such cases. If we have reference rankings r_{ui} and system rankings \hat{r}_{ui} of n_u items i for user u , we can define

$$C^+ = \sum_{ij} \text{sgn}(r_{ui} - r_{uj}) \text{sgn}(\hat{r}_{ui} - \hat{r}_{uj}) \quad (8.7)$$

$$C^- = \sum_{ij} \text{sgn}(r_{ui} - r_{uj}) \text{sgn}(\hat{r}_{uj} - \hat{r}_{ui}) \quad (8.8)$$

$$C^u = \sum_{ij} \text{sgn}^2(r_{ui} - r_{uj}) \quad (8.9)$$

$$C^s = \sum_{ij} \text{sgn}^2(\hat{r}_{ui} - \hat{r}_{uj}) \quad (8.10)$$

$$C^{u0} = C^u - (C^+ + C^-) \quad (8.11)$$

where the sums range over the $\frac{1}{2}n_u(n_u - 1)$ pairs of items. Thus, C^u is the number of pairs of items for which the reference ranking asserts an ordering (i.e. not tied), while C^+ and C^- are the number of these pairs that the system ranking asserts the correct order and the incorrect order respectively. C^{u0} is the number of pairs where the reference ranking does not tie but the system ranking ties. The NDPM is then given by

$$\text{NDPM} = \frac{C^- + 0.5C^{u0}}{C^u} \quad (8.12)$$

Thus, the NDPM measure gives a perfect score of 0 to systems that correctly predicts every preference relation asserted by the reference. The worst score of 1 is assigned to systems that contradict every reference preference relation. Not predicting a reference preference relation is penalized only half as much as contradicting it. Predicting preferences that the reference does not order (i.e. when we do not know the user's true preference) is not penalized.

In some cases, we may completely know the user's true preferences for some set of items. For example, we may elicit the user's true ordering by presenting the user with binary choices. In this case, when a pair of items are tied in the reference ranking it means that the user is actually indifferent between the items. Thus, a perfect system should not rank one item higher than the other. In such cases, rank correlation measures such as **Spearman's ρ** or **Kendall's τ** [30, 31] can be used. These measures tend to be highly correlated in practice [18]. Kendall's τ is given by

$$\tau = \frac{C^+ - C^-}{\sqrt{C^u} \sqrt{C^s}} \quad (8.13)$$

while Spearman’s ρ is given by

$$\rho = \frac{1}{n_u} \frac{\sum_i (r_{i,u} - \bar{r})(\hat{r}_{i,u} - \bar{\hat{r}})}{\sigma(r)\sigma(\hat{r})} \quad (8.14)$$

where $\bar{\cdot}$ and $\sigma(\cdot)$ denote the mean and standard deviation. Note that in the case of ties, each tied item should be assigned the average ranking, so that, e.g., if two items are tied for second and third place, they are assigned a rank of 2.5 [31].

Utility-based ranking

While using a reference ranking scores a ranking on its correlation with some “true” ranking, there are other criteria for deciding on ordering a list of items. One popular alternative is to assume that the utility of a list of recommendations is additive, given by the sum of the utilities of the individual recommendations. The utility of each recommendation is the utility of the recommended item discounted by a factor that depends on its position in the list of recommendations. One example of such a utility is the likelihood that a user will observe a recommendation at position i in the list. It is usually assumed that users scan recommendation lists from the beginning to the end, with the utility of recommendations being discounted more heavily towards the end of the list. The discount can also be interpreted as the probability that a user would observe a recommendation in a particular position in the list, with the utility of the recommendation given that it was observed depending only on the item recommend. Under this interpretation, the probability that a particular position in the recommendation list is observed is assumed to depend only on the position and not on the items that are recommended.

In many applications, the user can use only a single, or a very small set of items, or the recommendation engine is not used as the main browsing tool. In such cases, we can expect the users to observe only a few items of the top of the recommendations list. We can model such applications using a very rapid decay of the positional discount down the list. The **R-Score** metric [8] assumes that the value of recommendations decline exponentially down the ranked list to yield the following score for each user u :

$$R_u = \sum_u \sum_j \frac{\max(r_{ui_j} - d, 0)}{2^{\frac{j-1}{\alpha-1}}} \quad (8.15)$$

where i_j is the item in the j th position, r_{ui} is user u ’s rating of item i , d is a task dependent neutral (“don’t care”) rating, and α is a half-life parameter, which controls the exponential decline of the value of positions in the ranked list. In the case of ratings prediction tasks, r_{ui} is the rating given by the user to each item (e.g. 4 stars), and d is the don’t care vote (e.g. 3 stars), and the algorithm only gets credit for ranking items with rating above the “don’t care” vote higher than d (e.g. 4 or 5 stars). In usage prediction tasks, r_{ui} is typically 1 if u selects i and 0 otherwise, while

d is 0. Using $r_{ui} = -\log(\text{popularity}(i))$ if i is used and 0 otherwise [54] can capture the amount of information in the recommendation. The resulting per-user scores are aggregated using:

$$R = 100 \frac{\sum_u R_u}{\sum_u R_u^*} \quad (8.16)$$

where R_u^* is the score of the best possible ranking for user u .

In other applications the user is expected to read a relatively large portion of the list. In certain types of search, such as the search for legal documents [24], users may look for all relevant items, and would be willing to read large portions of the recommendations list. In such cases, we need a much slower decay of the positional discount. **Normalized Cumulative Discounted Gain (NDCG)** [27] is a measure from information retrieval, where positions are discounted logarithmically. Assuming each user u has a “gain” g_{ui} from being recommended an item i , the average Discounted Cumulative Gain (DCG) for a list of J items is defined as

$$\text{DCG} = \frac{1}{N} \sum_{u=1}^N \sum_{j=1}^J \frac{g_{uj}}{\max(1, \log_b j)} \quad (8.17)$$

where the logarithm base is a free parameter, typically between 2 and 10. A logarithm with base 2 is commonly used to ensure all positions are discounted. NDCG is the normalized version of DCG given by

$$\text{NDCG} = \frac{\text{DCG}}{\text{DCG}^*} \quad (8.18)$$

where DCG^* is the ideal DCG.

We show the two methods here as they were originally presented, but note that the numerator in the two cases contains a utility function that assigns a value for each item. One can replace the original utility functions with a function that is more appropriate to the designed application. A measure closely related to R-score and NDCG is **Average Reciprocal Hit Rank (ARHR)** [14] which is an un-normalized measure that assigns a utility $1/k$ to a successful recommendation at position k . Thus, ARHR decays more slowly than R score but faster than NDCG.

Online evaluation of ranking

In an online experiment designed to evaluate the ranking of the recommendation list, we can look at the interactions of users with the system. When a recommendation list is presented to a user, the user may select a number of items from the list. We can now assume that the user has scanned the list at least as deep as the last selection. That is, if the user has selected items 1, 3, and 10, we can assume that the user has observed items 1 through 10. We can now make another assumption, that the user has found items 1,3, and 10 to be interesting, and items 2,4,5,6,7,8, and 9 to be

uninteresting. In some cases we can have additional information whether the user has observed more items. For example, if the list is spread across several pages, and only 20 results are presented per page, then, in the example above, if the user moved to the second page we can also assume that she has observed results 11 through 20 and had found them to be irrelevant.

In the scenario above, the results of this interaction is a division of the list into 3 parts — the interesting items (1,3,10 in the example above), the uninteresting items (the rest of the items from 1 through 20), and the unknown items (21 till the end of the list). We can now use an appropriate reference ranking metric to score the original list. This can be done in two different ways. First, the reference list can contain the interesting items at the top, then the unknown items, and the uninteresting items at the bottom. This reference list captures the case where the user may only select a small subset of the interesting items, and therefore the unknown items may contain more interesting items. Second, the reference list can contain the interesting items at the top, followed by the uninteresting items, with the unknown items completely ignored. This is useful when making unreasonable preference assumptions, such as that some unknown items are preferred to the uninteresting items, may have negative consequences. In either case, it should be borne in mind that the semantics of the reference ranking are different from the case of offline evaluations. In offline evaluations, we have a single reference ranking which is assumed to be correct, and we measure how much each recommender deviates from this “correct” ranking. In the online case, the reference ranking is assumed to be the ranking that the user would have preferred given that were presented with the recommender’s ranking. In the offline case, we assume that there is one correct ranking, while in the online case we allow for the possibility of multiple correct rankings.

In the case of utility ranking, we can evaluate a list based on the sum of the utilities of the selected items. Lists that place interesting items with high utility close to the beginning of the list, will hence be preferred to lists that place these interesting items down the list, because we expect that in the latter case, the user will often not observe these interesting items at all, generating no utility for the recommender.

8.3.3 Coverage

As the prediction accuracy of a recommendation system, especially in collaborative filtering systems, in many cases grows with the amount of data, some algorithms may provide recommendations with high quality, but only for a small portion of the items where they have huge amounts of data. The term coverage can refer to several distinct properties of the system that we discuss below.

8.3.3.1 Item Space Coverage

Most commonly, the term coverage refers to the proportion of items that the recommendation system can recommend. This is often referred to as **catalog coverage**. The simplest measure of catalog coverage is the percentage of all items that can ever be recommended. This measure can be computed in many cases directly given the algorithm and the input data set.

A more useful measure is the percentage of all items that are recommended to users during an experiment, either offline, online, or a user study. In some cases it may be desirable to weight the items, for example, by their popularity or utility. Then, we may agree not to be able to recommend some items which are very rarely used anyhow, but ignoring high profile items may be less tolerable.

Another measure of catalog coverage is the **sales diversity** [16], which measures how unequally different items are chosen by users when a particular recommender system is used. If each item i accounts for a proportion $p(i)$ of user choices, the **Gini Index** is given by:

$$G = \frac{1}{n-1} \sum_{j=1}^n (2j-n-1)p(i_j) \quad (8.19)$$

where i_1, \dots, i_n is the list of items ordered according to increasing $p(i)$. The index is 0 when all items are chosen equally often, and 1 when a single item is always chosen. The Gini index of the number of times each item is recommended could also be used. Another measure of distributional inequality is the **Shannon Entropy**:

$$H = - \sum_{i=1}^n p(i) \log p(i) \quad (8.20)$$

The entropy is 0 when a single item is always chosen or recommended, and $\log n$ when n items are chosen or recommended equally often.

8.3.3.2 User Space Coverage

Coverage can also be the proportion of users or user interactions for which the system can recommend items. In many applications the recommender may not provide recommendations for some users due to, e.g. low confidence in the accuracy of predictions for that user. In such cases we may prefer recommenders that can provide recommendations for a wider range of users. Clearly, such recommenders should be evaluated on the trade-off between coverage and accuracy.

Coverage here can be measured by the richness of the user profile required to make a recommendation. For example, in the collaborative filtering case this could be measured as the number of items that a user must rate before receiving recommendations. This measurement can be typically evaluated in offline experiments.

8.3.3.3 Cold Start

Another, related set of issues are the well known cold start problems — the performance of the system on new items and on new users. Cold start can be considered as a sub problem of coverage because it measures the system coverage over a specific set of items and users. In addition to measuring how large the pool of cold start items or users are, it may also be important to measure system accuracy for these users and items.

Focusing on cold start items, we can use a threshold to decide on the set of cold items. For example, we can decide that cold items are only items with no ratings or usage evidence [52], or items that exist in the system for less than a certain amount of time (e.g., a day), or items that have less than a predefined evidence amount (e.g., less than 10 ratings). Perhaps a more generic way is to consider the “coldness” of an item using either the amount of time it exists in the system or the amount of data gathered for it. Then, we can credit the system more for properly predicting colder items, and less for the hot items that are predicted.

It may be possible that a system better recommends cold items at the price of a reduced accuracy for hotter items. This may be desirable due to other considerations such as novelty and serendipity that are discussed later. Still, when computing the system accuracy on cold items it may be wise to evaluate whether there is a trade-off with the entire system accuracy.

8.3.4 Confidence

Confidence in the recommendation can be defined as the system’s trust in its recommendations or predictions [128, 22]. As we have noted above, collaborative filtering recommenders tend to improve their accuracy as the amount of data over items grows. Similarly, the confidence in the predicted property typically also grows with the amount of data.

In many cases the user can benefit from observing these confidence scores [22]. When the system reports a low confidence in a recommended item, the user may tend to further research the item before making a decision. For example, if a system recommends a movie with very high confidence, and another movie with the same rating but a lower confidence, the user may add the first movie immediately to the watching queue, but may further read the plot synopsis for the second movie, and perhaps a few movie reviews before deciding to watch it.

Perhaps the most common measurement of confidence is the probability that the predicted value is indeed true, or the interval around the predicted value where a predefined portion, e.g. 95% of the true values lie. For example, a recommender may accurately rate a movie as a 4 star movie with probability 0.85, or have 95% of the actual ratings lie within -1 and $+\frac{1}{2}$ of the predicted 4 stars. The most general method of confidence is to provide a complete distribution over possible outcomes [40].

Given two recommenders that perform similarly on other relevant properties, such as prediction accuracy, it can be desirable to choose the one that can provide valid confidence estimates. In this case, given two recommenders with, say, identical accuracy, that report confidence bounds in the same way, we will prefer the recommender that better estimates its confidence bounds.

Standard confidence bounds, such as the ones above, can be directly evaluated in regular offline trials, much the same way as we estimate prediction accuracy. We can design for each specific confidence type a score that measures how close the method confidence estimate is to the true error in prediction. This procedure cannot be applied when the algorithms do not agree on the confidence method, because some confidence methods are weaker and therefore easier to estimate. In such a case a more accurate estimate of a weaker confidence metric does not imply a better recommender.

Example 8.1. Recommenders A and B both report confidence intervals over possible movie ratings. We train A and B over a confidence threshold, ranging of 95%. For each trained model, we run A and B on offline data, hiding a part of the user ratings and requesting each algorithm to predict the missing ratings. Each algorithm produces, along with the predicted rating, a confidence interval. We compute A_+ and A_- , the number of times that the predicted rating of algorithm A was within and outside the confidence interval (respectively), and do the same for B . Then we compute the true confidence of each algorithm using $\frac{A_+}{A_-+A_+} = 0.97$ and $\frac{B_+}{A_-+A_+} = 0.94$. The result indicates that A is over conservative, and computes intervals that are too large, while B is liberal and computes intervals that are too small. As we do not require the intervals to be conservative, we prefer B because its estimated intervals are closer to the requested 95% confidence.

Another application of confidence bounds is in filtering recommended items where the confidence in the predicted value is below some threshold. In this scenario we assume that the recommender is allowed not to predict a score for all values, as is typically the case when presenting top n recommendations. We can hence design an experiment around this filtering procedure by comparing the accuracy of two recommenders after their results were filtered by removing low confidence items. In such experiments we can compute a curve, estimating the prediction accuracy for each portion of filtered items, or for different filtering thresholds. This evaluation procedure does not require both algorithms to agree on the confidence method.

While user studies and online experiments can study the effect of reporting confidence on the user experience, it is difficult to see how these types of tests can be used to provide further evidence as to the accuracy of the confidence estimate.

8.3.5 *Trust*

While confidence is the system trust in its ratings, in trust we refer here to the user's trust in the system recommendation⁵. For example, it may be beneficial for the system to recommend a few items that the user already knows and likes. This way, even though the user gains no value from this recommendation, she observes that the system provides reasonable recommendations, which may increase her trust in the system recommendations for unknown items. Another common way of enhancing trust in the system is to explain the recommendations that the system provides (see Chapter 15). Trust in the systems is also called the credibility of the system.

If we do not restrict ourselves to a single method of gaining trust, such as the one suggested above, the obvious method for evaluating user trust is by asking users whether the system recommendations are reasonable in a user study [22, 12, 11, 47]. In an online test one could associate the number of recommendations that were followed with the trust in the recommender, assuming that higher trust in the recommender would lead to more recommendations being used. Alternatively, we could also assume that trust in the system is correlated with repeated users, as users who trust the system will return to it when performing future tasks. However, such measurements may not separate well other factors of user satisfaction, and may not be accurate. It is unclear how to measure trust in an offline experiment, because trust is built through an interaction between the system and a user.

8.3.6 *Novelty*

Novel recommendations are recommendations for items that the user did not know about [33]. In applications that require novel recommendation, an obvious and easy to implement approach is to filter out items that the user already rated or used. However, in many cases users will not report all the items they have used in the past. Thus, this simple method is insufficient to filter out all items that the user already knows.

While we can obviously measure novelty in a user study, by asking users whether they were already familiar with a recommended item [9, 28], we can also gain some understanding of a system's novelty through offline experiments. For such an experiment we could split the data set on time, i.e. hide all the user ratings that occurred after a specific point in time. In addition, we can hide some ratings that occurred prior to that time, simulating the items that the user is familiar with, but did not report ratings for. When recommending, the system is rewarded for each item that was recommended and rated after the split time, but would be punished for each item that was recommended but rated prior to the split time.

⁵ Not to be confused with trust in the social network research, used to measure how much a user believes another user. Some literature on recommender systems uses such trust measurements to filter similar users [39] and Chapter 20.

To implement the above procedure we must carefully model the hiding process such that it would resemble the true preference discovery process that occurs in the real system. In some cases the set of rated items is not a uniform sample of the set of all items the user is familiar with, and such bias should be acknowledged and handled if possible. For example, if we believe that the user will provide more ratings about special items, but less ratings for popular items, then the hiding process should tend to hide more popular items.

In using this measure of novelty, it is important to control for accuracy, as irrelevant recommendations may be new to the user, but still worthless. One approach would be to consider novelty only among the relevant items [63].

Example 8.2. We wish to evaluate the novelty of a set of movie recommenders in an offline test. As we believe that users of our system rate movies after they watch them, we split the user ratings in a sequential manner. For each test user profile we choose a cutoff point randomly along the time-based sequence of movie ratings, hiding all movies after a certain point in the sequence.

User studies on our system showed that people tend not to report ratings of movies that they did not feel strongly about, but occasionally also do not report a rating of a movie that they liked or disliked strongly. Therefore, we hide a rating of a movie prior to the cutoff point with probability $1 - \frac{|r-3|}{2}$ where $r \in \{1, 2, 3, 4, 5\}$ is the rating of the movie, and 3 is the neutral rating. We would like to avoid predicting these movies with hidden ratings because the user already knows about them.

Then, for each user, each recommender produces a list of 5 recommendations, and we compute precision only over items after the cutoff point. That is, the recommenders get no credit for recommending movies with hidden ratings that occurred prior to the cutoff point. In this experiment the algorithm with the highest precision score is preferred.

Another method for evaluating novel recommendations uses the above assumption that popular items are less likely to be novel. Thus, novelty can be taken into account by using an accuracy metric where the system does not get the same credit for correctly predicting popular items as it does when it correctly predicts non-popular items [53]. Ziegler *et al.* [64] and Celma and Herrera [9] also give accuracy measures that take popularity into account.

Finally, we can evaluate the amount of new information in a recommendation together with the relevance of the recommended item. For example, when item ratings are available, we can multiply the hidden rating by some information measurement of the recommended item (such as the conditional entropy given the user profile) to produce a novelty score.

8.3.7 Serendipity

Serendipity is a measure of how surprising the successful recommendations are. For example, if the user has rated positively many movies where a certain star actor

appears, recommending the new movie of that actor may be novel, because the user may not know of it, but is hardly surprising. Of course, random recommendations may be very surprising, and we therefore need to balance serendipity with accuracy.

One can think of serendipity as the amount of relevant information that is new to the user in a recommendation. For example, if following a successful movie recommendation the user learns of a new actor that she likes, this can be considered as serendipitous. In information retrieval, where novelty typically refers to the new information contained in the document (and is thus close to our definition of serendipity), Zhang *et al.* [63] suggested to manually label pairs of documents as redundant. Then, they compared algorithms on avoiding recommending redundant documents. Such methods are applicable to recommender systems when some meta-data over items, such as content information, is available.

To avoid human labeling, we could design a distance measurement between items based on content (see Chapter 3). Then, we can score a successful recommendation by its distance from a set of previously rated items in a collaborative filtering system, or from the user profile in a content-based recommender [62]. Thus, we are rewarding the system for successful recommendations that are far from the user profile.

Example 8.3. In a book recommendation application, we would like to recommend books from authors that the reader is less familiar with. We therefore design a distance metric between a book b and a set of books B (the books that the user has previously read); Let $c_{B,w}$ be the number of books by writer w in B . Let $c_B = \max_w c_{B,w}$ the maximal number of books from a single writer in B . Let $d(b, B) = \frac{1+c_B-c_{B,w(b)}}{1+c_B}$, where $w(b)$ is the writer of book b .

We now run an offline experiment to evaluate which of the candidate algorithms generates more serendipitous recommendations. We split each test user profile — set of books that the user has read — into sets of observed books B_i^o and hidden books B_i^h . We use B_i^o as the input for each recommender, and request a list of 5 recommendations. For each hidden book $b \in B_i^h$ that appeared in the recommendation list for user i , the recommender receives a score of $d(b, B_i^o)$. Thus the recommender is getting more credit for recommending books from writers that the reader has read less often. In this experiment the recommender that received a higher score is selected for the application.

One can also think of serendipity as deviation from the “natural” prediction [44]. That is, given a prediction engine that has a high accuracy, the recommendations that it issues are “obvious”. Therefore, we will give higher serendipity scores to successful recommendations that the prediction engine would deem unlikely.

We can evaluate the serendipity of a recommender in a user study by asking the users to mark the recommendations that they find unexpected. Then, we can also see whether the user followed these recommendations, which would make them unexpected and successful and therefore serendipitous. In an online study, we can assume that our distance metric is correct and evaluate only how distance from the user profile affected the probability that a user will follow the recommendation. It is

important to check the effect of serendipity over time, because users might at first be intrigued by the unexpected recommendations and try them out. If after following the suggestion they discover that the recommendations are inappropriate, they may stop following them in the future, or stop using the recommendation engine at all.

8.3.8 Diversity

Diversity is generally defined as the opposite of similarity. In some cases suggesting a set of similar items may not be as useful for the user, because it may take longer to explore the range of items. Consider for example a recommendation for a vacation [55], where the system should recommend vacation packages. Presenting a list with 5 recommendations, all for the same location, varying only on the choice of hotel, or the selection of attraction, may not be as useful as suggesting 5 different locations. The user can view the various recommended locations and request more details on a subset of the locations that are appropriate to her.

The most explored method for measuring diversity uses item-item similarity, typically based on item content, as in Section 8.3.7. Then, we could measure the diversity of a list based on the sum, average, min, or max distance between item pairs, or measure the value of adding each item to the recommendation list as the new item's diversity from the items already in the list [64, 6]. The item-item similarity measurement used in evaluation can be different from the similarity measurement used by the algorithm that computes the recommendation lists. For example, we can use for evaluation a costly metric that produces more accurate results than fast approximate methods that are more suitable for online computations.

As diversity may come at the expense of other properties, such as accuracy [62], we can compute curves to evaluate the decrease in accuracy vs. the increase in diversity.

Example 8.4. In a book recommendation application, we are interested in presenting the user with a diverse set of recommendations, with minimal impact to accuracy. We use $d(b, B)$ from Example 8.3 as the distance metric. Given candidate recommenders, each with a tunable parameter that controls the diversity of the recommendations, we train each algorithm over a range of values for the diversity parameters. For each trained model, we now compute a precision score, and a diversity score as follows; we take each recommendation list that an algorithm produces, and compute the distance of each item from the rest of the list, averaging the result to obtain a diversity score. We now plot the precision-diversity curves of the recommenders in a graph, and select the algorithm with the dominating curve.

In recommenders that assist in information search (see Chapter 18), we can assume that more diverse recommendations will result in shorter search interactions [55]. We could use this in an online experiment measuring interaction sequence length as a proxy for diversification. As is always the case in online testing, shorter sessions may be due to other factors of the system, and to validate this claim it is

useful to experiment with different diversity thresholds using the same prediction engine before comparing different recommenders.

8.3.9 *Utility*

Many e-commerce websites employ a recommendation system in order to improve their revenue by, e.g., enhancing cross-sell. In such cases the recommendation engine can be judged by the revenue that it generates for the website [54]. In general, we can define various types of utility functions that the recommender tries to optimize. For such recommenders, measuring the utility, or the expected utility of the recommendations may be more significant than measuring the accuracy of recommendations. It is also possible to view many of the other properties, such as diversity or serendipity, as different types of utility functions, over single items or over lists. In this paper, however, we define utility as the value that either the system or the user gains from a recommendation.

Utility can be measured cleanly from the perspective of the recommendation engine or the recommender system owner. Care must be taken, though, when measuring the utility that the user receives from the recommendations. First, user utilities or preferences are difficult to capture and model, and considerable research has focused on this problem [7, 21, 48]. Second, it is unclear how to aggregate user utilities across users for computing a score for a recommender. For example, it is tempting to use money as a utility thus selecting a recommender that minimizes user cost. However, under the diminishing returns assumption [56], the same amount of money does not have the same utility for people with different income levels. Therefore, the average cost per purchase, for example, is not a reasonable aggregation across users.

In an application where users rate items, it is also possible to use the ratings as a utility measurement [8]. For example, in movie ratings, where a 5 star movie is considered an excellent movie, we can assume that a recommending a 5 star movie has a higher utility for the user than recommending a movie that the user will rate with 4 stars. As users may interpret ratings differently, user ratings should be normalized before aggregating across users.

While we typically only assign positive utilities to successful recommendations, we can also assign negative utilities to unsuccessful recommendations. For example, if some recommended item offends the user, then we should punish the system for recommending it by assigning a negative utility. We can also add a cost to each recommendation, perhaps based on the position of the recommended item in the list, and subtract it from the utility of the item.

For any utility function, the standard evaluation of the recommender is to compute the expected utility of a recommendation. In the case where the recommender is trying to predict only a single item, such as when we evaluate the system on time-based splits and try to predict only the next item in the sequence, the value of a correct recommendation should simply be the utility of the item. In the task where

the recommender predicts n items we can use the sum of the utilities of the correct recommendations in the list. When negative utilities for failed recommendations are used, then the sum is over all recommendations, successful or failed. We can also integrate utilities into ranking measurements, as discussed in Section 8.3.2.3. Finally, we can normalize the resulting score using the maximal possible utility given the optimal recommendation list.

Evaluating utility in user studies and online is easy in the case of recommender utility. If the utility we optimize for is the revenue of the website, measuring the change in revenue between users of various recommenders is simple. When we try to optimize user utilities the online evaluation becomes harder, because users typically find it challenging to assign utilities to outcomes. In many cases, however, users can say whether they prefer one outcome to another. Therefore, we can try to elicit the user preferences [26] in order to rank the candidate methods.

8.3.10 Risk

In some cases a recommendation may be associated with a potential risk. For example, when recommending stocks for purchase, users may wish to be risk-averse, preferring stocks that have a lower expected growth, but also a lower risk of collapsing. On the other hand, users may be risk-seeking, preferring stocks that have a potentially high, even if less likely, profit. In such cases we may wish to evaluate not only the (expected) value generated from a recommendation, but also to minimize the risk.

The standard way to evaluate risk sensitive systems is by considering not just the expected utility, but also the utility variance. For example, we may use a parameter q and compare two systems on $E[X] + q \cdot \text{Var}(X)$. When q is positive, this approach prefers risk-seeking (also called bold [41]) recommenders, and when q is negative, the system prefers risk-averse recommenders.

8.3.11 Robustness

Robustness (see Chapter 25) is the stability of the recommendation in the presence of fake information [45], typically inserted on purpose in order to influence the recommendations. As more people rely on recommender systems to guide them through the item space, influencing the system to change the rating of an item may be profitable to an interested party. For example, an owner of an hotel may wish to boost the rating for their hotel. This can be done by injecting fake user profiles that rate the hotel positively, or by injecting fake users that rate the competitors negatively.

Such attempts to influence the recommendation are typically called attacks [43, 35]. Coordinated attacks occur when a malicious user intentionally queries the data

set or injects fake information in order to learn some private information of some users. In evaluating such systems, it is important to provide a complete description of the attack protocol, as the sensitivity of the system typically varies from one protocol to another.

In general, creating a system that is immune to any type of attack is unrealistic. An attacker with an ability to inject an infinite amount of information can, in most cases, manipulate a recommendation in an arbitrary way. It is therefore more useful to estimate the cost of influencing a recommendation, which is typically measured by the amount of injected information. While it is desirable to theoretically analyze the cost of modifying a rating, it is not always possible. In these cases, we can simulate a set of attacks by introducing fake information into the system data set, empirically measuring average cost of a successful attack [36, 10].

As opposed to other evaluation criteria discussed here, it is hard to envision executing an attack on a real system as an online experiment. It may be fruitful, however, to analyze the real data collected in the online system to identify actual attacks that are executed against the system.

Another type of robustness is the stability of the system under extreme conditions, such as a large number of requests. While less discussed, such robustness is very important to system administrators, who must avoid system malfunction. In many cases system robustness is related to the infrastructure, such as the database software, or to the hardware specifications, and is related to scalability (Section 8.3.14).

8.3.12 Privacy

In a collaborative filtering system, a user willingly discloses his preferences over items to the system in the hope of getting useful recommendations. However, it is important for most users that their preferences stay private, that is, that no third party can use the recommendation system to learn something about the preferences of a specific user.

For example, consider the case where a user who is interested in the wonderful, yet rare art of growing Bahamian orchids has bought a book titled “The Divorce Organizer and Planner”. The spouse of that user, looking for a present, upon browsing the book “The Bahamian and Caribbean Species (Cattleyas and Their Relatives)” may get a recommendation of the type “people who bought this book also bought” for the divorce organizer, thus revealing sensitive private information.

It is generally considered inappropriate for a recommendation system to disclose private information even for a single user. For this reason analysis of privacy tends to focus on a worst case scenario, illustrating theoretical cases under which users private information may be revealed. Other researchers [17] compare algorithms by evaluating the portion of users whose private information was compromised. The assumption in such studies is that complete privacy is not realistic and that therefore we must compromise on minimizing the privacy breaches.

Another alternative is to define different levels of privacy, such as k -identity [17], and compare algorithms sensitivity to privacy breaches under varying levels of privacy.

Privacy may also come at the expense of the accuracy of the recommendations. Therefore, it is important to analyze this trade-off carefully. Perhaps the most informative experiment is when a privacy modification has been added to an algorithm, and the accuracy (or any other trade-off property) can be evaluated with or without the modification [42].

8.3.13 *Adaptivity*

Real recommendation systems may operate in a setting where the item collection changes rapidly, or where trends in interest over items may shift. Perhaps the most obvious example of such systems is the recommendation of news items or related stories in online newspapers [19]. In this scenario stories may be interesting only over a short period of time, afterwards becoming outdated. When an unexpected news event occurs, such as the tsunami disaster, people become interested in articles that may not have been interesting otherwise, such as a relatively old article explaining the tsunami phenomenon. While this problem is similar to the cold-start problem, it is different because it may be that old items that were not regarded as interesting in the past suddenly become interesting.

This type of adaptation can be evaluated offline by analyzing the amount of information needed before an item is recommended. If we model the recommendation process in a sequential manner, we can record, even in an offline test, the amount of evidence that is needed before the algorithm recommends a story. It is likely that an algorithm can be adjusted to recommend items faster once they become interesting, by sacrificing some prediction accuracy. We can compare two algorithms by evaluating a possible trade-off between accuracy and the speed of the shift in trends.

Another type of adaptivity is the rate by which the system adapts to a user's personal preferences [37], or to changes in user profile [34]. For example, when users rate an item, they expect the set of recommendations to change. If the recommendations stay fixed, users may assume that their rating effort is wasted, and may not agree to provide more ratings. As with the shift in trends evaluation, we can again evaluate in an offline experiment the changes in the recommendation list after adding more information to the user profile such as new ratings. We can evaluate an algorithm by measuring the difference between the recommendation lists before and after the new information was added. The Gini index and Shannon entropy measures discussed in Section 8.3.3 can be used to measure the variability of recommendations made to a user as the user profile changes.

8.3.14 Scalability

As recommender systems are designed to help users navigate in large collections of items, one of the goals of the designers of such systems is to scale up to real data sets. As such, it is often the case that algorithms trade other properties, such as accuracy or coverage, for providing rapid results even for huge data sets consisting of millions of items (e.g. [12]).

With the growth of the data set, many algorithms are either slowed down or require additional resources such as computation power or memory. One standard approach in computer science research is to evaluate the computational complexity of an algorithm in terms of time or space requirements (as done, e.g., in [29, 4]). In many cases, however, the complexity of two algorithms is either identical, or could be reduced by changing some parameters, such as the complexity of the model, or the sample size. Therefore, to understand the scalability of the system it is also useful to report the consumption of system resources over large data sets.

Scalability is typically measured by experimenting with growing data sets, showing how the speed and resource consumption behave as the task scales up (see, e.g. [19]). It is important to measure the compromises that scalability dictates. For example, if the accuracy of the algorithm is lower than other candidates that only operate on relatively small data sets, one must show over small data sets the difference in accuracy. Such measurements can provide valuable information both on the potential performance of recommender systems in general for the specific task, and on future directions to explore.

As recommender systems are expected in many cases to provide rapid recommendations online, it is also important to measure how fast does the system provides recommendations [23, 50]. One such measurement is the throughput of the system, i.e., the number of recommendations that the system can provide per second. We could also measure the latency (also called response time) — the required time for making a recommendation online.

8.4 Conclusion

In this paper we discussed how recommendation algorithms could be evaluated in order to select the best algorithm from a set of candidates. This is an important step in the research attempt to find better algorithms, as well as in application design where a designer chooses an existing algorithm for their application. As such, many evaluation metrics have been used for algorithm selection in the past.

We describe the concerns that need to be addressed when designing offline and online experiments and user studies. We outline a few important measurements that one must take in addition to the score that the metric provides, as well as other considerations that should be taken into account when designing experiments for recommendation algorithms.

We specify a set of properties that are sometimes discussed as important for the recommendation system. For each such property we suggest an experiment that can be used to rank recommenders with regards to that property. For less explored properties, we restrict ourselves to generic descriptions that could be applied to various manifestations of that property. Specific procedures that can be practically implemented can then be developed for the specific property manifestation based on our generic guidelines.

References

1. Bamber, D.: The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of Mathematical Psychology* **12**, 387–415 (1975)
2. benjamini: Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. R. Statist. Soc. B* **57**(1), 289–300 (1995)
3. Bonhard, P., Harries, C., McCarthy, J., Sasse, M.A.: Accounting for taste: using profile similarity to improve recommender systems. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 1057–1066. ACM, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1124772.1124930>
4. Boutilier, C., Zemel, R.S.: Online queries for collaborative filtering. In: In Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (2002)
5. Box, G.E.P., Hunter, W.G., Hunter, J.S.: *Statistics for Experimenters*. Wiley, New York (1978)
6. Bradley, K., Smyth, B.: Improving recommendation diversity. In: Twelfth Irish Conference on Artificial Intelligence and Cognitive Science, pp. 85–94 (2001)
7. Brazuinas, D., Boutilier, C.: Local utility elicitation in GAI models. In: Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence, pp. 42–49. Edinburgh (2005)
8. Breese, J.S., Heckerman, D., Kadie, C.M.: Empirical analysis of predictive algorithms for collaborative filtering. In: UAI, pp. 43–52 (1998)
9. Celma, O., Herrera, P.: A new approach to evaluating novel recommendations. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, pp. 179–186. ACM, New York, NY, USA (2008). DOI <http://doi.acm.org/10.1145/1454008.1454038>
10. Chirita, P.A., Nejdl, W., Zamfir, C.: Preventing shilling attacks in online recommender systems. In: WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management, pp. 67–74. ACM, New York, NY, USA (2005). DOI <http://doi.acm.org/10.1145/1097047.1097061>
11. Cramer, H., Evers, V., Ramlal, S., Someren, M., Rutledge, L., Stash, N., Aroyo, L., Wielinga, B.: The effects of transparency on trust in and acceptance of a content-based art recommender. *User Modeling and User-Adapted Interaction* **18**(5), 455–496 (2008). DOI <http://dx.doi.org/10.1007/s11257-008-9051-3>
12. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, pp. 271–280. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1242572.1242610>
13. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7**, 1–30 (2006)
14. Deshpande, M., Karypis, G.: Item-based top-N recommendation algorithms. *ACM Transactions on Information Systems* **22**(1), 143–177 (2004)
15. Fischer, G.: User modeling in human-computer interaction. *User Model. User-Adapt. Interact.* **11**(1-2), 65–86 (2001)

16. Fleder, D.M., Hosanagar, K.: Recommender systems and their impact on sales diversity. In: EC '07: Proceedings of the 8th ACM conference on Electronic commerce, pp. 192–199. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1250910.1250939>
17. Frankowski, D., Cosley, D., Sen, S., Terveen, L., Riedl, J.: You are what you say: privacy risks of public mentions. In: SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 565–572. ACM, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1148170.1148267>
18. Fredricks, G.A., Nelsen, R.B.: On the relationship between spearman's rho and kendall's tau for pairs of continuous random variables. *Journal of Statistical Planning and Inference* **137**(7), 2143–2150 (2007)
19. George, T.: A scalable collaborative filtering framework based on co-clustering. In: Fifth IEEE International Conference on Data Mining, pp. 625–628 (2005)
20. Greenwald, A.G.: Within-subjects designs: To use or not to use? *Psychological Bulletin* **83**, 216–229 (1976)
21. Haddawy, P., Ha, V., Restificar, A., Geisler, B., Miyamoto, J.: Preference elicitation via theory refinement. *Journal of Machine Learning Research* **4**, 2003 (2002)
22. Herlocker, J.L., Konstan, J.A., Riedl, J.T.: Explaining collaborative filtering recommendations. In: CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work, pp. 241–250. ACM, New York, NY, USA (2000). DOI <http://doi.acm.org/10.1145/358916.358995>
23. Herlocker, J.L., Konstan, J.A., Riedl, J.T.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.* **5**(4), 287–310 (2002). DOI <http://dx.doi.org/10.1023/A:1020443909834>
24. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **22**(1), 5–53 (2004). DOI <http://doi.acm.org/10.1145/963770.963772>
25. Hijikata, Y., Shimizu, T., Nishida, S.: Discovery-oriented collaborative filtering for improving user satisfaction. In: IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces, pp. 67–76. ACM, New York, NY, USA (2009). DOI <http://doi.acm.org/10.1145/1502650.1502663>
26. Hu, R., Pu, P.: A comparative user study on rating vs. personality quiz based preference elicitation methods. In: IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces, pp. 367–372. ACM, New York, NY, USA (2009). DOI <http://doi.acm.org/10.1145/1502650.1502702>
27. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.* **20**(4), 422–446 (2002). DOI <http://doi.acm.org/10.1145/582415.582418>
28. Jones, N., Pu, P.: User technology adoption issues in recommender systems. In: *Networking and Electronic Conference* (2007)
29. Karypis, G.: Evaluation of item-based top-n recommendation algorithms. In: CIKM '01: Proceedings of the tenth international conference on Information and knowledge management, pp. 247–254. ACM, New York, NY, USA (2001). DOI <http://doi.acm.org/10.1145/502585.502627>
30. Kendall, M.G.: A new measure of rank correlation. *Biometrika* **30**(1–2), 81–93 (1938)
31. Kendall, M.G.: The treatment of ties in ranking problems. *Biometrika* **33**(3), 239–251 (1945)
32. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. *Data Min. Knowl. Discov.* **18**(1), 140–181 (2009). DOI <http://dx.doi.org/10.1007/s10618-008-0114-1>
33. Konstan, J.A., McNee, S.M., Ziegler, C.N., Torres, R., Kapoor, N., Riedl, J.: Lessons on applying automated recommender systems to information-seeking tasks. In: *AAAI* (2006)
34. Koychev, I., Schwab, I.: Adaptation to drifting user's interests. In: *Proceedings of ECML2000 Workshop: Machine Learning in New Information Age*, pp. 39–46 (2000)
35. Lam, S.K., Frankowski, D., Riedl, J.: Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. In: *Proceedings of the 2006 Interna-*

- tional Conference on Emerging Trends in Information and Communication Security (ETRICS (2006)
36. Lam, S.K., Riedl, J.: Shilling recommender systems for fun and profit. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, pp. 393–402. ACM, New York, NY, USA (2004). DOI <http://doi.acm.org/10.1145/988672.988726>
 37. Mahmood, T., Ricci, F.: Learning and adaptivity in interactive recommender systems. In: ICEC '07: Proceedings of the ninth international conference on Electronic commerce, pp. 75–84. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1282100.1282114>
 38. Marlin, B.M., Zemel, R.S., Roweis, S., Slaney, M.: Collaborative filtering and the missing at random assumption. In: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (2007)
 39. Massa, P., Bhattacharjee, B.: Using trust in recommender systems: An experimental analysis. In: In Proceedings of iTrust2004 International Conference, pp. 221–235 (2004)
 40. McLaughlin, M.R., Herlocker, J.L.: A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In: SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 329–336. ACM, New York, NY, USA (2004). DOI <http://doi.acm.org/10.1145/1008992.1009050>
 41. McNee, S.M., Riedl, J., Konstan, J.A.: Making recommendations better: an analytic model for human-recommender interaction. In: CHI '06: CHI '06 extended abstracts on Human factors in computing systems, pp. 1103–1108. ACM, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1125451.1125660>
 42. McSherry, F., Mironov, I.: Differentially private recommender systems: building privacy into the netflix prize contenders. In: KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 627–636. ACM, New York, NY, USA (2009). DOI <http://doi.acm.org/10.1145/1557019.1557090>
 43. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Technol.* **7**(4), 23 (2007). DOI <http://doi.acm.org/10.1145/1278366.1278372>
 44. Murakami, T., Mori, K., Orihara, R.: Metrics for evaluating the serendipity of recommendation lists. *New Frontiers in Artificial Intelligence* **4914**, 40–46 (2008)
 45. O'Mahony, M., Hurley, N., Kushmerick, N., Silvestre, G.: Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Technol.* **4**(4), 344–377 (2004). DOI <http://doi.acm.org/10.1145/1031114.1031116>
 46. Pfleeger, S.L., Kitchenham, B.A.: Principles of survey research. *SIGSOFT Softw. Eng. Notes* **26**(6), 16–18 (2001). DOI <http://doi.acm.org/10.1145/505532.505535>
 47. Pu, P., Chen, L.: Trust building with explanation interfaces. In: IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces, pp. 93–100. ACM, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1111449.1111475>
 48. Queiroz, S.: Adaptive preference elicitation for top-k recommendation tasks using gain-networks. In: AIAP'07: Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference, pp. 579–584. ACTA Press, Anaheim, CA, USA (2007)
 49. Salzberg, S.L.: On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Min. Knowl. Discov.* **1**(3), 317–328 (1997). DOI <http://dx.doi.org/10.1023/A:1009752403260>
 50. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. In: WWW '01: Proceedings of the 10th international conference on World Wide Web, pp. 285–295. ACM, New York, NY, USA (2001). DOI <http://doi.acm.org/10.1145/371920.372071>
 51. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of recommendation algorithms for e-commerce. In: EC '00: Proceedings of the 2nd ACM conference on Electronic commerce, pp. 158–167. ACM, New York, NY, USA (2000). DOI <http://doi.acm.org/10.1145/352871.352887>

52. Schein, A.I., Popescul, A., Ungar, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 253–260. ACM, New York, NY, USA (2002). DOI <http://doi.acm.org/10.1145/564376.564421>
53. Shani, G., Chickering, D.M., Meek, C.: Mining recommendations from the web. In: RecSys '08: Proceedings of the 2008 ACM Conference on Recommender Systems, pp. 35–42 (2008)
54. Shani, G., Heckerman, D., Brafman, R.I.: An mdp-based recommender system. *Journal of Machine Learning Research* **6**, 1265–1295 (2005)
55. Smyth, B., McClave, P.: Similarity vs. diversity. In: ICCBR, pp. 347–361 (2001)
56. Spillman, W., Lang, E.: *The Law of Diminishing Returns*. World Book Company (1924)
57. Swearingen, K., Sinha, R.: Beyond algorithms: An hci perspective on recommender systems. In: ACM SIGIR 2001 Workshop on Recommender Systems (2001)
58. Van Rijsbergen, C.J.: *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA (1979). URL <http://portal.acm.org/citation.cfm?id=539927>
59. Voorhees, E.M.: Overview of trec 2002. In: In Proceedings of the 11th Text Retrieval Conference (TREC 2002), NIST Special Publication 500-251, pp. 1–15 (2002)
60. Voorhees, E.M.: The philosophy of information retrieval evaluation. In: CLEF '01: Revised Papers from the Second Workshop of the Cross-Language Evaluation Forum on Evaluation of Cross-Language Information Retrieval Systems, pp. 355–370. Springer-Verlag, London, UK (2002)
61. Yao, Y.Y.: Measuring retrieval effectiveness based on user preference of documents. *J. Amer. Soc. Inf. Sys* **46**(2), 133–145 (1995)
62. Zhang, M., Hurley, N.: Avoiding monotony: improving the diversity of recommendation lists. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, pp. 123–130. ACM, New York, NY, USA (2008). DOI <http://doi.acm.org/10.1145/1454008.1454030>
63. Zhang, Y., Callan, J., Minka, T.: Novelty and redundancy detection in adaptive filtering. In: SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 81–88. ACM, New York, NY, USA (2002). DOI <http://doi.acm.org/10.1145/564376.564393>
64. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: WWW '05: Proceedings of the 14th international conference on World Wide Web, pp. 22–32. ACM, New York, NY, USA (2005). DOI <http://doi.acm.org/10.1145/1060745.1060754>

Chapter 9

A Recommender System for an IPTV Service Provider: a Real Large-Scale Production Environment

Riccardo Bambini, Paolo Cremonesi and Roberto Turrin

Abstract In this chapter we describe the integration of a recommender system into the production environment of Fastweb, one of the largest European IP Television (IPTV) providers. The recommender system implements both collaborative and content-based techniques, suitable tailored to the specific requirements of an IPTV architecture, such as the limited screen definition, the reduced navigation capabilities, and the strict time constraints. The algorithms are extensively analyzed by means of off-line and on-line tests, showing the effectiveness of the recommender systems: up to 30% of the recommendations are followed by a purchase, with an estimated lift factor (increase in sales) of 15%.

9.1 Introduction

IP Television (IPTV) broadcasts multimedia content (e.g., movies, news programs, documentaries) in digital format via broadband Internet networks [23, 17]. IPTV services include scheduled television programs and video on demand (VoD) contents [30]. In the rest of the chapter we will generically refer to both scheduled television programs and video-on-demand contents as *items*.

In this chapter we present the integration of the Neptunus's ContentWise recommender system in Fastweb, the first company in the world to have launched fully IP-based broadband TV services, in 2001. Fastweb serves hundreds of thousands of

Riccardo Bambini
Fastweb, via Francesco Caracciolo 51, Milano, Italy
e-mail: riccardo.bambini@fastweb.it

Paolo Cremonesi · Roberto Turrin
Politecnico di Milano, p.zza Leonardo da Vinci 32, Milano, Italy
Neptunus, via Durando 10, Milano, Italy
e-mail: paolo.cremonesi@polimi.it
e-mail: roberto.turrin@polimi.it

IPTV customers, with a catalog of thousands of multimedia contents. Since 2007 Fastweb is part of the Swisscom group. ContentWise recommender algorithms have been developed with the cooperation of the Computer Science Department at the Politecnico di Milano.

Differently from conventional television, IPTV allows an interactive navigation of the available content [13] and, in particular, IPTV allows to collect implicit usage data and explicit user preferences for providing a personalized user navigation. The user interacts with the IPTV system by means of a special electronic appliance, referred to as *set-top-box* (STB). There are substantial peculiarities of the STB that limit the user interaction: (i) users control the STB by means of a remote control, which is rather limited in the set of actions it allows to perform, (ii) the user interface is shown on a TV screen typically designed to be looked at from a distance larger than that between a PC and the user, and (iii) the system deals with multimedia content, whose navigation is not particularly fast, mainly because of the channel switching time.

Differently from traditional e-commerce domains (e.g., Amazon, Netflix, iTunes, IMDB, Last.fm) where recommender systems have been exploited, IPTV recommender systems need to satisfy particular requirements:

- the list of proposed items has to be small because of the limited screen definition and the reduced navigation capabilities;
- the generation of the recommended items must respect very strict time constraints (few milliseconds) because TV's customers are used to a very responsive system;
- the system needs to scale up in a successful manner with both the number of customers and items in the catalog;
- part of the catalog is highly dynamic because of live broadcast channels.

The recommender system deployed in Fastweb generates recommendations by means of two collaborative algorithms (based on item-to-item similarities and dimensionality reduction techniques) and one content-based algorithm (based on latent semantic analysis). The recommender system selects the proper algorithm depending on the context. For instance, if the user is reading a movie synopsis, looking for movies with his preferred actors, the algorithm used is the content-based one. In order to respect the strict real-time requirements, the recommender system and the underlying algorithms follow a model-based approach and have been logically divided into two asynchronous stages, the batch stage and the real-time stage.

The input data of the whole architecture is composed by: (i) the item-content matrix and (ii) the user-rating matrix. The item-content matrix (ICM) describes the main attributes (metadata) of each item, such as the title of a movie, the set of actors and its genre(s). The user-rating matrix (URM) collects the ratings (i.e., preferences) of users about items. Ratings are mainly implicit, e.g., the system can detect if a user watched a program, without knowing explicitly the user's opinion about that program.

Before deploying the recommender system in production, extensive performance analysis has been performed by means of k -fold cross validation. The results sug-

gests a 2.5% recall for the content-based algorithm, while the collaborative algorithms are able to reach a recall of more than 20%.

The recommender system has released to production environment in October 2008 and is now available for one of the Fastweb VOD catalogs. The system is actually providing, on average, 30000 recommendations per day, with peaks of almost 120 recommendations per minute during peak hours. On-line analysis shows that almost 20% of the recommendations are followed by a purchase from the users. The estimated lift factor (i.e., increase in VOD sales) is 15%.

The rest of the chapter is organized as follows. Section 9.2 shows the typical architecture of an IPTV provider. Section 9.3 presents the architecture of the recommender system. Section 9.4 describes the implemented recommender algorithms. Section 9.5 explains the recommender services implemented into the Fastweb IPTV architecture. Section 9.6 evaluates the quality of recommendations. Finally, Section 9.7 draws the conclusions.

9.2 IPTV Architecture

IPTV, also called Internet Protocol Television, is a video service that delivers high quality traditional TV channels and on-demand video and audio contents over a private IP-based broadband network. From the end users perspective, IPTV looks and operates just like a standard TV service. The providers involved in deploying IPTV services range from cable and satellite TV carriers to large telephone companies and private network operators. IPTV has a number of unique features [13]:

Support for interactive TV: differently from conventional TV, where the communication is unidirectional, the two-way capabilities of IPTV systems allow the user to interact with the system.

Time shifting: IPTV permits the temporal navigation of programs (e.g., fast forward, pause and rewind) thanks to the Personal Video Recording (PVR), a mechanism for recording and storing IPTV content for later viewing.

Personalization: IPTV allows end users to personalize their TV viewing experience by letting them decide what they want to watch and when they want to watch it.

Figure 9.1 shows a generic IPTV system architecture that supports live broadcast TV channels (also called *linear channels*) and video on-demand (VOD). Broadcast TV service consists in the simultaneous reception by the users of traditional TV channels either free-to-air or pay-per-view. Video on-demand service consists in viewing multimedia content made available by the service provider, upon request.

The IPTV data center (also known as the *head end*) receives linear channels from a variety of sources including terrestrial, satellite and cable carriers. Once received, a number of different hardware components, ranging from encoders and video servers, are used to prepare the video content for delivery over an IP based network. On-demand contents are stored in fast storage boxes (e.g., using solid-state disks).

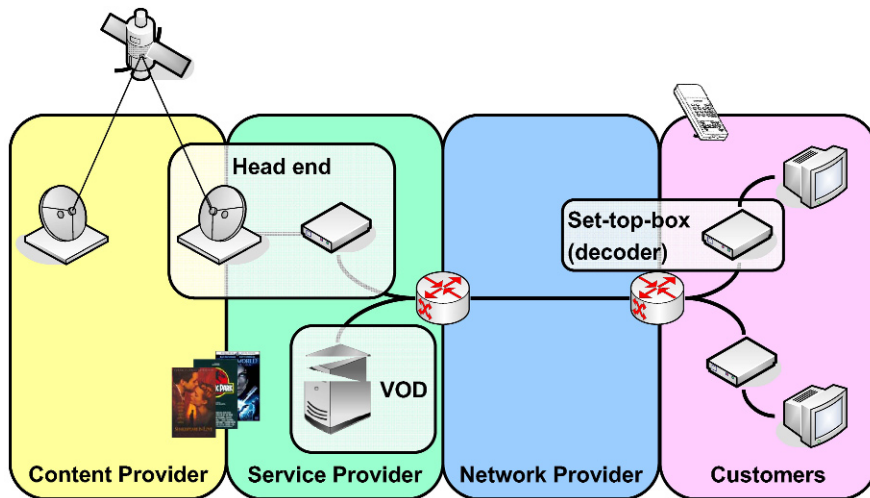


Fig. 9.1: Architecture of an IPTV system.

The set-top-box (STB) is an electronic appliance that connects to both the network and the home television: it is responsible for processing the incoming packet stream and displaying the resulting video on the television. The user interacts with the STB by means of a hand-held remote control. The remote control gives the user access to additional features of the STB, such as the Electronic Program Guide (EPG), a listing of available channels and program for an extended time period (typically 36 hours or more).

9.2.1 IPTV Search Problems

To benefit from the rich set of IPTV channels and contents, users need to be able to rapidly and easily find what they are actually interested in, and do so effortlessly while relaxing on the couch in their living room, a location where they typically do not have easy access to the keyboard, mouse, and close-up screen display typical of desktop web browsing. However, searching for a live channel or a VOD content is a challenging problem for IPTV users [11].

When watching live television, users browse through a set of available channels until they find something interesting. Channel selection (zapping) involves two steps: (a) sampling the content to decide whether to continue or stop watching the channel, and (b) switching across multiple channels for repeated sampling, until a desired channel is found. The problem of quickly finding the right channel becomes harder as the number of channel offerings grows in modern IPTV systems.

Moreover, IPTV channel switching time is not particularly responsive, compared to traditional TV, because of technological limitations [19].

When searching for VOD content, IPTV users generally have to either navigate a complex, pre-defined, and often deeply embedded menu structure or type in titles or other key phrases using an on-screen keyboard or triple tap input on the remote control keypad. These interfaces are cumbersome and do not scale well as the range of content available increases. Moreover, the television screens usually offer a limited resolution with respect to traditional personal computer screens, making traditional graphical user interfaces difficult to use.

This differs from traditional web-based domains (e.g., e-commerce web sites), where the content is textual, suited for information categorization and keyword-based seek and retrieval, and the input devices (keyboard and mouse) allow to point an arbitrary object on the screen and to easily enter text.

The integration of a recommender system into the IPTV infrastructure improves the user experience by providing a new and more effective way of browsing for interesting programs and movies. However, such integration has to deal with the following issues:

User identification. The STB is used indistinctly by all the components of a family, and the IPTV recommender system can not identify who is actually watching a certain program.

Real-time requirements. The IPTV recommender systems must generate recommendations within very strict real-time constraints (few milliseconds) in order to avoid a slow down of the user navigation, already affected by the long channel switching time.

Quality of content metadata. Differently from web-based domains, content-based IPTV recommender algorithms makes use of low-quality metadata. This aspect is particularly evident with live channels, where new content is added every day at a very high rate, and the only available metadata that can be used to describe programs can be found in the EPG (electronic program guide).

9.3 Recommender System Architecture

The architecture of the Fastweb recommender system is shown in Figure 9.2. These components are discussed in the following section. Section 9.3.1 describes the information available to the recommender system. Section 9.3.2 describes the two-stage architecture of the recommender algorithms, separating between batch and real-time stage. Section 9.4 details the three algorithms implemented in ContentWise. Finally, Section 9.5 shows the integration of the recommender system into the existing Fastweb architecture.

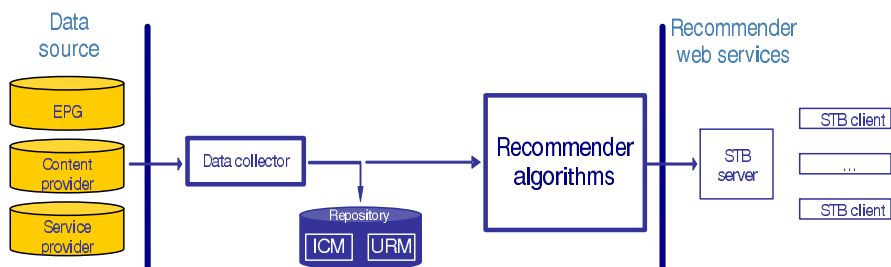


Fig. 9.2: Architecture of the recommender system ContentWise.

9.3.1 Data Collection

The logical component in charge of pre-processing the data and generating the input of the recommender algorithm is referred to as *data collector*. The data collector gathers data from different sources, such as the EPG for information about the live programs, the content provider for information about the VOD catalog and the service provider for information about the users.

The Fastweb recommender system does not rely on personal information from the users (e.g., age, gender, occupation). Recommendations are based on the past users' behavior (what they watched) and on any explicit preference they have expressed (e.g., preferred genres). If the users did not specify any explicit preferences, the system is able to infer them by analyzing the users' past activities.

An important question has been raised in Section 9.2: users interact with the IPTV system by means of the STB, but typically we can not identify who is actually in front of the TV. Consequently, the STB collects the behavior and the preferences of a set of users (e.g., the component of a family). This represents a considerable problem since we are limited to generate per-STB recommendations. In order to simplify the notation, in the rest of the paper we will refer to user and STB to identify the same entity. The user-disambiguation problem has been partially solved by separating the collected information according to the time slot they refer to. For instance, we can roughly assume the following pattern: housewives use to watch TV during the morning, children during the afternoon, the whole family at evening, while only adults watch TV during the night. By means of this simple time slot distinction we are able to distinguish among different potential users of the same STB.

Formally, the available information has been structured into two main matrices, practically stored into a relational database: the item-content matrix (ICM) and the user-rating matrix (URM).

The former describes the principal characteristics (metadata) of each item. In the following we will refer to the item-content matrix as \mathbf{W} , whose elements w_{ci} represent the relevance of characteristic (metadata) c for item i . The ICM is generated from the analysis of the set of information given by the content provider (i.e., the EPG). Such information concerns, for instance, the title of a movie, the

actors, the director(s), the genre(s) and the plot. Note that in a real environment we can face with inaccurate information especially because of the rate new content is added every day. The information provided by the ICM is used to generate a content-based recommendation, after being filtered by means of techniques for PoS (Part-of-Speech) tagging, stop words removal, and latent semantic analysis [32]. Moreover, the ICM can be used to perform some kind of processing on the items (e.g., parental control).

The URM represents the ratings (i.e., preferences) of users about items. In the following we will refer to such matrix as \mathbf{R} , whose elements r_{pi} represent the rating of user p about item i . Such preferences constitute the basic information for any collaborative algorithm. The user rating can be either explicit or implicit, according to the fact that the ratings are explicitly expressed by users or are implicitly collected by the system, respectively.

Explicit ratings confidently represent the user opinion, even though they can be affected by biases [4] due to: user subjectivity, item popularity or global rating tendency. The first bias depends on arbitrary interpretations of the rating scale. For instance, in a rating scale between 1 and 5, some user could use the value 3 to indicate an interesting item, someone else could use 3 for a not much interesting item. Similarly, popular items tend to be overrated, while unpopular items are usually underrated. Finally, explicit ratings can be affected by global attitudes (e.g., users are more willing to rate movies they like).

On the other hand, implicit ratings are inferred by the system on the basis of the user-system interaction, which might not match the user opinion. For instance, the system is able to monitor whether a user has watched a live program on a certain channel or whether the user has uninterruptedly watched a movie. Despite explicit ratings are more reliable than implicit ratings in representing the actual user interest towards an item, their collection can be annoying from the user's perspective.

The current deployment of the Fastweb recommender system collects only implicit ratings, but the system is thought to work when implicit and explicit ratings coexist. The rating scale is between 1 and 5, where values less than 3 express negative ratings, values greater or equal to 3 express positive ratings. In absence of explicit information, the rating implicitly inferred by monitoring the user behavior is assumed to be positive (i.e., greater or equal than 3). In fact, whether a user starts watching a certain program, there must be some characteristic of this program appealing for the user (e.g., actors or genre). The fact that in well-know, explicit datasets, such as Netflix and Movielens, the average rating is higher than 3, motivates this assumption. The system treats differently live IPTV programs and VOD content:

IPTV programs. The rating is proportional to the percentage user play time (e.g., [18, 35]), i.e., the percentage of program the user has watched. Let us assume L is the program play time and t is the user play time. Play times less than 5 minutes are discarded. If a user watched the entire program the rating is 5, if the user watched 5 minutes the rating is 3, otherwise the rating is a value between 3 and 5 given by:

$$\hat{r} = 3 + 2 \frac{t-5}{L-5}, \quad 5 \leq t \leq L \quad (9.1)$$

where t and L are expressed in minutes.

At this early stage of the project, the main goal is not to define an accurate implicit rating mechanism, but rather, to filter out noisy information (e.g., TV channel zapping).

VOD movies. When watching a VOD movie, users explicitly request to buy and to pay for that movie. For that reason, independently from the user play time, when a user requests a VOD movie, the system assign an implicit ratings equals to 4.

As aforementioned, should Fastweb start collecting explicit ratings too, they will naturally coexist with implicit ratings in the URM.

The ratings stored in the URM, before being used by the recommender algorithms, are normalized by subtracting the constant value 2.5. This allows the algorithms to distinguish between positive and negative ratings, because values greater or equals to 2.5 (i.e., 3, 4, and 5) remain positive, while values less than 2.5 (i.e., 1 and 2) become negative. In the rest of the chapter we will assume that the recommender algorithms receive as input a normalized URM.

Finally, users can express *explicit preferences* about the content they would like to watch. For instance, by means of the graphical interface, a user can set his preferred actors. The content-based algorithm explained in Section 9.4.2 takes into consideration such information and biases the recommended movies toward the expressed preferences.

9.3.2 Batch and Real-Time Stages

The recommender algorithms process the ICM and the URM described in Section 9.3.1 and they interface with the STB server by means of web services, as shown in Figure 9.3.

In order to respect the strict real-time requirements, the recommender system and the underlying algorithms follow a model-based approach [33, 9], i.e., they first develop a model of the user ratings and/or of the items, then they compute the recommendations. Consequently, the algorithms have been logically divided into two stages, the *batch* stage and the *real-time* stage:

- the batch stage creates a low dimensional representation (i.e., a model) of the input data. It is usually executed during the service off-peak hours, with a frequency which depends on the rate new items/users are added into the system (e.g., once a day);
- the real-time part uses the model in order to serve calls coming from the web services interface and satisfying the real-time constraints. The system output can be further constrained by post-processing, marketing rules (e.g., pushing up some movies, or filtering some channels).

The model repository makes the two stages asynchronous, e.g., while the real-time stage is recommending users by using a certain model, the batch stage can compute a new, updated model.

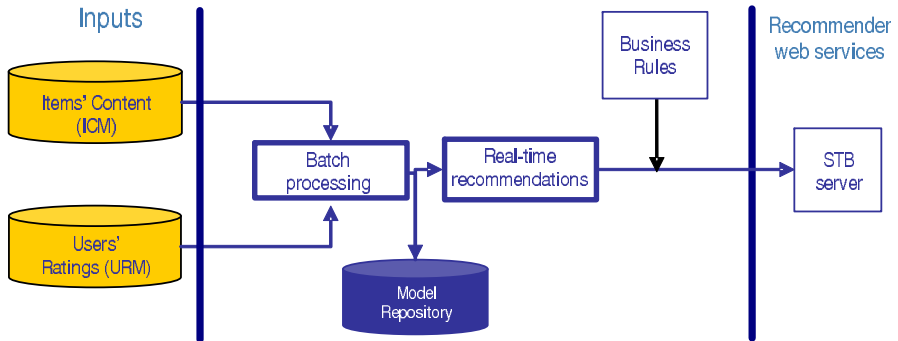


Fig. 9.3: Recommender system: batch and real-time stage.

Despite such logical division of the recommending process, the model construction in real domains can still be challenging because of input data size and the related time and memory requirements. For this reason, we have implemented high-performing, parallel versions of the most demanding matrix operations, optimized for sparse and big matrices, such as: matrix-matrix and matrix-vector multiplication, matrix transposition, column/row normalization, and singular value decomposition (svd). In particular, svd has been used with two of the three recommender algorithms (one content-based and one collaborative), allowing to greatly reduce the space dimensionality, with benefits both in terms of memory and time complexity. As we will show in the following sections, by its own, svd defines a model of the data, cleaning up the data noise and strengthening the correlation among similar information.

Realistic datasets with millions of users and items can have in principle prohibitive memory requirements. Fortunately, matrices such as URM and ICM are typically very sparse. In fact, most of the users interact (e.g., rate or watch) with very few items compared with the size of the catalog (e.g., the average users have watched few dozens of movies in a catalog of thousands). Sparse matrices can be treated using very efficient representations. Note that, even though such matrices are sparse, we could have difficulties in maintaining the data in memory. For such reasons, we opted for a solution based on a sort of *memory virtualization*, similar to the swap capability of any operating systems. Differently from the operating system virtual memory, our virtualization policy is tailored ad-hoc for each matrix operation, in order to limit the data exchange between memory and storage.

9.4 Recommender Algorithms

The recommender system implements one content-based algorithm (CB) and two collaborative algorithms (CF):

- a latent-semantic analysis content-based algorithm, referred to as *LSA-CB*;
- an item-based collaborative algorithm, referred to as *item-based-CF*;
- a dimensionality-reduction-based collaborative algorithm, referred to as *SVD-CF*.

In the following section we present a brief overview of the recommender algorithms. Section 9.4.2, 9.4.3, and 9.4.4 present the details of the three algorithms, i.e., respectively, the LSA, the item-based, and the dimensionality-reduction algorithms.

9.4.1 Overview of Recommender Algorithms

Recommender algorithms can be classified into content-based and collaborative algorithms.

The content-based (see Chapter 3) approach to recommendation has its roots in information retrieval, which provides a set of tools for searching for textual information, such as in documents, web sites, usenet news and mail messages.

A content-based system is based on the analysis of the content of the items. The model of an item is so composed by a set of features representing its content. The assumption underlying content-based techniques is that the meaning and the relevance of items can be captured by such features:

- each feature is assigned to a weight indicating how representative it is for an item;
- similar items contain similar features;
- the more items contain a feature, the less representative the feature is (i.e., it is less important in order to distinguish an item from an other).

The feature extraction is probably the most critical phase of such systems and it can be particularly challenging in IPTV, where resources are non-textual, such as audio/video streams. For instance, the textual features of a movie can be the genre (e.g., comedy), the list of actors, etc. While more interesting information could be obtained by analyzing the audio/video tracks, this technology [11] is fairly recent and it is necessary to examine whether it can really bring some improvement in this specific domain.

The classical way of representing items in content-based recommender is by means of the *bag-of-words* (BOW) approach [6], where we consider textual features and we only retain frequencies of words, discarding any grammar/semantic connection. Usually the words are pre-processed by means of tokenisation, stop-words removal and stemming [32]. The former simply splits text into tokens (e.g., words).

Tokens not useful for representing an item in a certain domain are discarded (stop-words). Finally, stemming is used to normalize some kind of grammar variability by converting tokens to their morphological root. For example, the words 'play', 'player', 'playing', and 'played' would all be converted to their root form, 'play'. After the pre-processing, each token has assigned a weight which is proportional to its frequency normalized using various schemes, the most known being the TF-IDF scheme [26, 32].

The BOW representation can be summarized in the matrix \mathbf{W} , where column i represents item i and the element w_{ci} represents the weight (relevance) of metadata c for item i . The metadata c can be the movie genre, an actor or the director, as well as a token extracted from the movie synopsis. We will present in Section 9.4.2 how the different kind of metadata have been dealt with. Analogously, also users are represented as vectors in the space of tokens. In fact, the profile of a user is derived by means of a linear combination of the vectors corresponding to the items he has rated, weighted with the related user rating. Recommendations are then obtained by comparing the similarity between the vector representing the user profile and the vectors representing the items. The most similar items are then proposed to the user. Similarity between two vectors can be expressed by several metrics, such as the euclidean distance and the cosine distance [26].

Content-based systems [1, 3, 21] recommend items similar to those that a user liked in the past, by considering their features. For example, the system analyzes the movies that a user liked in the past and it constructs a model of the user, whose features are the actors, the producers, the genres, the directors, etc., that such user prefers. Then, those movies that have a high degree of similarity to the user's preferences would be recommended. For instance, whether a user is used to watch many action movies, he will be recommended other action movies. This characteristic of content-based recommender systems has two direct effects: it assures that the recommended items are coherent with the user's interests, but, at the same time, the set of recommended items could be obvious and too homogeneous. This issue is usually referred to as *over-specialization* problem [3].

The main advantage of content-based techniques is that, since they are based on evident resource features, they can provide an understandable and immediate explanation of the recommended items. Furthermore, content-based filtering is based on a well-know and mature technology.

In contrast to content-based, collaborative systems (see Chapter 5) try to suggest items to a particular user on the basis of the other-users' preferences [29, 1]. In fact, in everyday life, we often rely on recommendations from other people such as by word of mouth or movie reviews. Such systems use the opinions of a community of users to help individuals more effectively identify content of interest. Collaborative systems assist and augment this process. They are based on the following two assumptions:

- there are groups of users with similar tastes, which rate the items similarly;
- correlated items are rated by a group of users similarly.

The concept of correlation is strongly different to the content-based similarity among items. For instance, here we are saying that whatever the content of a movie is, such movie is considered somehow “similar” to another one because the community expressed the same evaluation for both the movies. For instance, if a user has watched the movie “Forrest Gump”, from a collaborative point of view the system could suggest him to watch the movie “The Sixth Sense”. The relation among these movies has apparently no sense because, looking at the content, they are not similar movies, but they are actually strongly-correlated because most of the people who have watched “Forrest Gump”, also watched “The Sixth Sense”.

Starting from the previous two assumption, we can define two classes of collaborative recommenders, respectively, the user-based and the item-based [34]. Both of them are based on social interactions. In practice, user-based recommenders are seldom used because of their poor quality and their memory and time requirements.

Note that collaborative recommendation does not need to extract any feature from the items. Thus, such systems do not have the same shortcomings that content-based systems have. In particular, since collaborative systems use other-users’ preferences, they can deal with any kind of content. Furthermore they can recommend any items, even the ones with a content which does not correspond to any item previously liked.

However, also collaborative systems have their own limitations. The main drawback is that collaborative recommenders are affected by the (or first-rater) problem. Since such systems recommend the items most correlated to those preferred by the current user, a new item can not be recommended because nobody has rated it so far and the system can not define a model for such item. Therefore, until the new item is rated by a substantial number of users, the recommender system will not be able to recommend it. For such reasons, collaborative algorithms are not practicable in live TV domains, where new programs enter the system at a very high rate and appear and receive ratings for a very limited time window (e.g., few hours). Note that content-based recommenders do not suffer for such a problem because when new items enter into the collection their model is given by their own features.

A second issue is called the *sparsity* problem. In fact, the effectiveness of collaborative systems depend on the availability of sets of users with similar preferences. Unfortunately, in any recommender system, the number of ratings already obtained is usually very small compared to the number of ratings to estimate. As a consequence, it might not be possible to recommend someone with unique tastes, because there will not be anyone enough similar to him.

As a consequence of the above two points, at the beginning of its activity, a brand new system will not be able to provide any accurate recommendation; it is called the *cold start* problem. The problem is common to all kinds of recommender systems, both content-based and collaborative recommenders, but in the latter the issue is particularly evident since their model is based only on user ratings.

In addition, since popular items are the most rated, collaborative recommenders are likely to be biased toward the most popular items. For instance, if a movie has been rated by only few people, this movie would be recommended very rarely, because the predicted rating might be not reliable.

9.4.2 LSA Content-Based Algorithm

The content-based algorithm implemented in Fastweb is based on the BOW approach, enhanced by means of latent semantic analysis.

Referring to Figure 9.2, the retrieving of features available for each item in the catalog is performed by the data collector. The features are filtered and weighted, forming the ICM. The features of an item are classified into several groups, referred to as *metadata*. Different kinds of items have different sets of metadata:

- VOD content: actors, directors, producers, title, series title, episode name, studio name, country, year, runtime, synopsis, available languages;
- Live IPTV program: actors, directors, producers, channel and time scheduling, country, runtime, year, synopsis.

The main difference between VOD and live IPTV content is that the former can be accessed by users at any time upon request, while the latter can only be accessed by users at the time it is broadcasted on a certain channel. This must be taken into consideration by the recommender algorithms.

For any item, each metadata is represented by either a string (e.g., the title) or a vector of strings (e.g., the list of actors). According to the kind of metadata, each string is differently pre-processed and weighted. Whether the metadata contains proper names (i.e., actors and directors) we do not apply any processing, but we simply keep the string as it is. On the other hand, metadata containing sentences (i.e., the title and the synopsis) are tokenized, filtered (stop-word removal) and stemmed. Furthermore, some metadata are more important than others, and so the assigned weights. By means of cross-validation we obtained the weights of each kind of metadata (i.e., title, synopsis, actors, and directors). In addition, the weights of each stem in the synopsis are further multiplied for the corresponding TF-IDF value (e.g., see [26, 32]).

For instance, let us consider a movie with the following metadata:

- title: ‘the title’;
- genre: ‘comedy’;
- actors: ‘FirstName1 LastName1’, ‘FirstName2 LastName2’;
- synopsis: ‘The movie’s plot’.

The related column in the ICM matrix \mathbf{W} will have non-null weights in correspondence of the following elements:

- titl;
- genre;
- FirstName1-LastName1;
- FirstName2-LastName2;
- movi;
- plot;

where actors and genre are taken as-is, while the synopsis and the title are tokenized, stemmed (e.g., ‘titl’ is the stem of ‘title’) and stop-words are removed (e.g., ‘the’).

In addition to this data pre-processing, the content-based algorithm is powered by LSA (latent semantic analysis), a method well-known in the settings of information retrieval for automatic indexing and searching of documents [16, 8]. The approach takes advantage of the implicit structure (*latent semantic*) in the association of terms with documents. The technique consists in decomposing \mathbf{W} into a set of orthogonal factors whose linear combination approximates the original matrix. The decomposition is performed by means of singular value decomposition (SVD)

Supposing the ICM is a $c \times n$ matrix (c metadata and n items), it can be factorized into three matrices, \mathbf{U} ($c \times l$), \mathbf{S} ($l \times l$), and \mathbf{V} ($n \times l$) so that:

$$\mathbf{W} \approx \mathbf{USV}^T \quad (9.2)$$

where l is the number of latent semantic characteristics of items. Generally l is unknown and it must be computed with cross-validation techniques. \mathbf{S} contains the first l singular value of \mathbf{W} that, roughly speaking, are related to the importance of each latent characteristic. The columns of \mathbf{U} and \mathbf{V} are orthonormal and represent, respectively, the left and right singular vectors. The product \mathbf{USV}^T is the best rank- l linear approximation of \mathbf{W} in terms of the Frobenius norm [25]. Note that SVD is unique except for some linear combinations of rows and columns of the three resulting matrices and, conventionally, the diagonal elements of \mathbf{S} are constructed so to be positive and sorted by decreasing magnitude.

The SVD defines a new vector space, whose dimensions are not the c metadata, but the $l \ll c$ latent semantic features. We can represent item i into the latent space by projecting (folding-in) the related column of \mathbf{W} ; being \mathbf{d}_i such column vector, its projection $\tilde{\mathbf{d}}_i$ is given by:

$$\tilde{\mathbf{d}}_i = \mathbf{U}^T \mathbf{d}_i \quad (9.3)$$

Similarly, metadata c can be represented into the latent space as the projection of the related row of \mathbf{W} , referred to as \mathbf{w}_c , into the latent space:

$$\tilde{\mathbf{w}} = \mathbf{w}_c \mathbf{V} \mathbf{S} \quad (9.4)$$

Figure 9.4 describes the folding-in. Let's observe that we can project back the vectors into the original space, obtaining an approximate representation of the original vector. Although LSA is an approximation of the original BOW space, it has two main advantages:

- it constitutes a great improvement in terms of memory and computation requirements. In fact, once the SVD has been computed by the batch stage, the system works at real-time on the low-dimensional space defined by the l latent semantic dimensions, much smaller than the BOW space;
- by keeping only the l most important characteristics, we filter out the data noise and we strengthen the relationships between items and metadata. For instance, if two metadata co-appear in many items, this means they are somehow correlated and they will be represented similarly in the latent space. The correlation might also be indirect, discovering hidden dependences [31, 16].

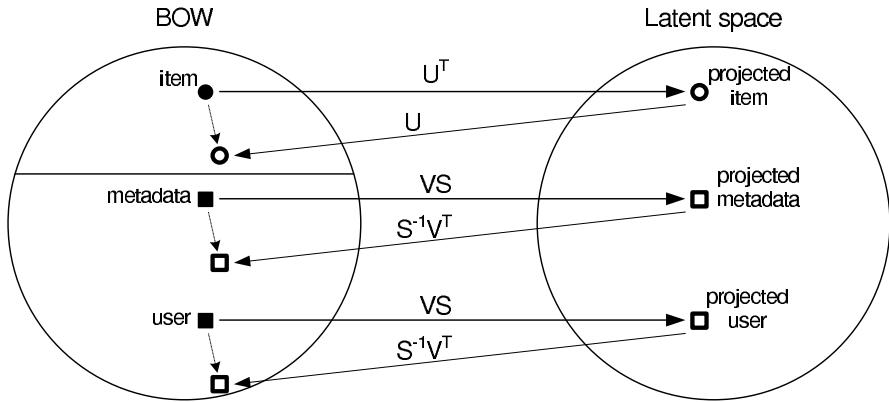


Fig. 9.4: LSA: folding-in of users, items and metadata into the common latent semantic space.

The major issue with SVD is its computational complexity: in fact, in the general case, the decomposition of a $m \times n$ matrix is $O(mn^2)$. Anyway, in the case of sparse matrices, there exist very efficient and scalable solutions. For instance, the SVD implementation by Lanczos [5] is optimized for sparse, large matrices: referring to z as the number of non-zero elements in the URM, the memory requirement is $O(z)$, and the computational complexity is $O(zl)$, i.e., directly proportional to z and to the number of singular values to be computed [36, 28]. In the Fastweb recommender system we have adopted the Lanczos implementation for the SVD, porting it to run on multi-processor architectures.

Recommending items to a user requires to estimate their relevance (rating). Thus, as well as we represented items in the latent space, we represent users in the same space, so that we can compute user-item correlations. A user is represented as a set of ratings and, as well as a row of the ICM (i.e., a metadata), the user ratings can be projected into the latent space by means of 9.4, where \mathbf{w}_c must be replaced with the user profile, i.e., a row vector of ratings. Once items and users have been represented in the same vector space, we can compute the relevance of item i for user p , referred to as \hat{r}_{pi} , by means of any correlation metric among vectors. The metric used in Fastweb is a shrank version of the *cosine*. Assuming that the l -dimensional vectors $\tilde{\mathbf{r}}_p$ and $\tilde{\mathbf{d}}_i$ represent, respectively, the projected user and the projected item, the estimated rating of user p about item i is given by:

$$\hat{r}_{pi} = \frac{\sum_{e=1}^l \tilde{r}_{pe} \cdot \tilde{d}_{ie}}{\sqrt{\sum_{e=1}^l [\tilde{r}_{pe}]^2} \cdot \sqrt{\sum_{e=1}^l [\tilde{d}_{ie}]^2} + \gamma} \tag{9.5}$$

where, for instance, \tilde{r}_{pe} indicates the e -th element of vector $\tilde{\mathbf{r}}_p$. The constant γ is the shrinking factor which corrects the metric in the case of scarce information,

i.e., when user or item vectors are meaningless because very close to the origin (e.g., an item with few metadata).

Observe that this representation allows to integrate *explicit* user preferences, e.g., the actors a user has explicitly declared to like. In fact, a vector of explicit user preferences can be treated similarly to an item, i.e., a vector of metadata. Once the explicit preferences have been folded into the latent space, the projected user and the projected explicit preferences can be combined to form a new user profile biased toward the explicit preferences.

9.4.3 Item-based Collaborative Algorithm

Item-based collaborative algorithms [9, 27] capture the fundamental relationships among items. As explained in Section 9.4.1, two items are similar (from a ‘collaborative’ point of view) if the community agrees about their ratings. Such similarity can be represented in a $m \times m$ matrix, referred to as \mathbf{D} , where the element d_{ij} expresses the similarity between item i and item j . Note that, potentially, \mathbf{D} could be non-symmetric (e.g., the conditional probability-based similarity described in [9]), i.e., $d_{ij} \neq d_{ji}$. That means that, for instance, item i could be very similar to item j (thus if a user likes item i he would like item j), even if item j is not similar to item i .

Item-based algorithms represent items in the user-rating space, i.e., an item is a vector whose dimensions are the ratings given by the n users. The coordinate of each dimension is the user rating. As a consequence, item i corresponds to the i -th column of \mathbf{R} , and the relationships among items are expressed by means of the similarities among the related vectors. In the following sections we describe several techniques to calculate the similarities among these vectors.

According to the system architecture shown in Figure 9.3, matrix \mathbf{D} represents the model of the recommender system and its calculation, being computational intensive, is delegated to the batch part of the recommender system. The real-time part generates a recommendation list by using such model. Given the profile of the target user p to recommend (represented by a vector of ratings), we can predict the rating \hat{r}_{pi} by computing the weighted sum of the ratings given by user p on the items similar to i . Such ratings are weighted by the similarity with item i . Referring to Q_i as the set of items similar to i , the prediction of \hat{r}_{pi} can be formulated as:

$$\hat{r}_{pi} = \frac{\sum_{j \in Q_i} d_{ji} \cdot r_{pj}}{F} \quad (9.6)$$

where F is a normalization factor. Such factor could be simply set to 1 or, as discussed in [27], it can be computed as:

$$F = \sum_{j \in Q_i} |d_{ji}| \quad (9.7)$$

thus assuring that \hat{r}_{pi} is within the predefined rating scale. Note that, being an item-based, model-based approach, user p can be recommended even though it is not taken into account during the model construction (in fact the batch stage computes a model of the items). This allows, for example, (i) to build the model with a subsample of users (e.g., in order to respect time and memory constraints) and (ii) to recommend a user even if his profile is new or update with respect to the moment the model was calculated.

Once computed the predicted ratings for all the items in the dataset that have not been rated by the target user, such ratings are sorted and the N highest-rated items compose the top- N recommendation list.

The set Q_i can be reduced by considering, for instance, only the items with a similarity greater than a certain threshold, or the k most similar items. This latter approach is the classical k NN (k -nearest-neighbors) approach. Section 9.6 shows that, by varying k , the quality of recommendations varies accordingly.

When using implicit datasets, similarity metric is usually computed using a frequency-based approach, as the one discussed by Deshpande and Karypis in [9]. For instance, when we only dispose of binary values, a high similarity between item i and item j means that when someone buys item i , it is very likely that he will buy also item j .

We can treat implicit datasets by considering each item as a vector in the user-rating space, where now the coordinates are binary values. Again, the similarity between two items can be computed as the similarity between the correspondent vectors, for example by means of the cosine metric.

With regard to implicit ratings, the cosine similarity is a special case of a more general approach that we refer in the following as direct relations (DR). In its basis formulation, the item-to-item matrix \mathbf{D} used with the DR is given by:

$$\mathbf{D} = \mathbf{R}^T \cdot \mathbf{R} \quad (9.8)$$

The elements d_{ii} on the principal diagonal is the total number of ratings for item i , while the other elements d_{ij} represent the number of users that have seen both item i and item j . The model (i.e., \mathbf{D}) can be post-processed by means of a post-normalization, whose general expression is [9]:

$$d_{ij} \leftarrow \frac{d_{ij}}{d_{ii}^\beta d_{jj}^\gamma + c} \quad (9.9)$$

where γ , β , and c are constant parameters whose optimal values depend on the dataset. The constant c is a shrinking factor [14], correcting the item-to-item similarity measure where poor information is available.

The model has been further enhanced by considering a k NN (k -nearest-neighborhood) approach. For each item, we consider only the k most similar items (referred to as the item's neighborhood), where k is to be chosen, for instance, by means of cross-validation techniques. By keeping only these items, we discard the noise of the items poorly correlated to the target item, improving the quality of recommendations.

Note that the aforementioned approaches are based on counting the co-rated items and they can be efficiently performed by any DBMS (Database Management System) using simple SQL statements without the need of external programs.

9.4.4 Dimensionality-Reduction-Based Collaborative Algorithm

Collaborative algorithms based on dimensionality reduction techniques describe the dataset (i.e., users and items) by means of a limited set of *features*. These features are different in their meaning from the features typically extracted in the case of content-based algorithms. In fact, the latter are characteristics concerning the content of items (e.g., the genre of a movie, the singer of a song, ...), while the features used by collaborative algorithms are not based on the content, but on the implicit way the user community interacts with the items.

Let us assume that an item can be described by means of l features, i.e., it is represented as a vector in the l -dimensional feature space. Similarly, a user is represented by a vector in the same space. As a consequence, the correlation between user p and item i (i.e., how much the item matches the user interests) can be computed as the similarity between the correspondent vectors, for instance by means of their inner product:

$$\hat{r}_{pi} = \sum_{e=1}^l a_{pe} \cdot b_{ie} \quad (9.10)$$

where, a_{pe} and b_{ie} are the e -th (unknown) features for user p and item i , respectively.

The point is to compute the l features which minimize the prediction error between the estimated \hat{r}_{pi} and the actual value r_{pi} .

For instance, Paterek in [20] applies an optimization method, referred to as regularized singular value decomposition, already used in the domain of natural language processing [12]. The l features of users and items are estimated by minimizing the metric RMSE (Root Mean Square Error), one feature at a time, using an optimization technique based on gradient descent. The metric RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{p,i} (\hat{r}_{pi} - r_{pi})^2} \quad (9.11)$$

In this implementation we have used again SVD, that has applied directly to the URM, similarly to the LSA. In fact, the URM can be factorized as:

$$\hat{\mathbf{R}} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \quad (9.12)$$

where, again, \mathbf{U} is a $n \times l$ orthonormal matrix, \mathbf{V} is a $m \times l$ orthonormal matrix, and \mathbf{S} is a $l \times l$ diagonal matrix containing the first l singular values, sorted in decreasing order.

The rating of user p about item i can be predicted as:

$$\hat{r}_{pi} = \sum_{e=1}^l u_{pe} \cdot s_{ee} \cdot v_{ie} \quad (9.13)$$

where u_{pe} is the element in the p -th row and e -th column of \mathbf{U} , v_{ie} is the element in the i -th row and e -th column of \mathbf{V} , and s_{ee} is the singular value in the e -th row and e -th column of \mathbf{S} .

Assuming that \mathbf{u}_p represents the p -row of \mathbf{U} and \mathbf{v}_i the i -row of \mathbf{V} , (9.13) can be rewritten as:

$$\hat{r}_{pi} = \mathbf{u}_p \cdot \mathbf{S} \cdot \mathbf{v}_i^T \quad (9.14)$$

Reminding that \mathbf{U} and \mathbf{V} have orthonormal columns, by multiplying both terms of (9.12) by \mathbf{V} , we can state that:

$$\mathbf{u}_p \cdot \mathbf{S} = \mathbf{r}_p \cdot \mathbf{V} \quad (9.15)$$

where \mathbf{r}_p is the p -th row of \mathbf{R} (i.e., the profile vector of user p). Consequently, (9.14) can be reformulated as:

$$\hat{r}_{pi} = \mathbf{r}_p \cdot \mathbf{V} \cdot \mathbf{v}_i^T \quad (9.16)$$

By means of (9.16) we are able to recommend any user, even if his profile \mathbf{r}_p is new or it has been updated since our model was created (i.e., since the SVD was performed). This represents a great advantage when compared, for instance, with other dimensionality-reduction techniques (e.g., the regularized SVD), where the features for a certain user are pre-computed and fixed during the model construction.

In order to predict all the ratings for user p , (9.16) can be straightforwardly extended as:

$$\hat{\mathbf{r}}_p = \mathbf{r}_p \cdot \mathbf{V} \cdot \mathbf{V}^T \quad (9.17)$$

Note that the product between \mathbf{V} and \mathbf{V}^T results into a $m \times m$ item-to-item matrix, whose meaning is very similar to the item-to-item matrix \mathbf{D} discussed in Section 9.4.3 about item-based algorithms.

Similarly to LSA, there are several advantages in using such SVD-based approach:

- SVD represents items and users in a low-dimensional space. Once \mathbf{R} has been factorized, which can result particularly challenging, the system operates with vectors having only l dimensions, much less than the original space of n users and m items;
- SVD reduces the noise in the data. In fact, by neglecting the singular values with low magnitude we are discarding the least-informative data, which is typically noisy [10, 8];
- SVD strengthens the relationships among the data. Thus, if two vectors (either users or items) are similar (because somehow related), they are represented closer in the l -dimensional feature space than in the original space. Observe that the relationship might also be indirect, i.e., by means of the SVD we could discover hidden dependences among users or items.

With regard to the algorithm architecture described in Section 9.3.2, the matrix factorization (9.12) is delegated to the batch part, while the prediction of ratings (9.16) can be performed at real-time. The real-time process estimates the ratings for all the unrated items of the target user, then such ratings are sorted and the N highest-rated items are selected to form the top- N recommendation list. In our tests, the time spent for computing the top- N recommendation list of a certain user by means of (9.16) is few milliseconds, fitting any real-time requirements.

9.5 Recommender Services

This section presents the implemented recommender services and how they impact into the user interfaces and the IPTV architecture. The recommender system can generate both content-based and collaborative-based recommendations. As summarized in Figure 9.5, content-based algorithms are applied both to VOD and live TV domains, while collaborative algorithms are applied only to VOD. In fact, we have already observed in Section 9.4.1 that collaborative algorithms are not practicable in this domain since new programs continuously enter the system, and collaborative algorithms are not able to recommend new items till they are viewed/rated by a substantial number of people.

	VOD	Live TV
Item-based CF	✔	
SVD-CF	✔	
LSA-CB	✔	✔

Fig. 9.5: Application of recommender algorithms to VOD and live TV.

At the current stage of the integration, Fastweb is exposing the full set of recommender services to a selected set of beta test users before the effective release. The other customers have access to a reduced set of the recommender functionalities. An example of the user interface available by means of the STB is shown in Figure 9.6.

The services released to the full customer base concern one of the catalog of VOD domain. Recommendations are provided by the LSA-CB algorithm presented in Section 9.4.2. The content-based algorithm has been preferred for the first few months of activity because collaborative algorithms suffer from the cold-start problem, as explained in Section 9.4.1. Moreover, collaborative recommenders need to record the behavior of users. This faces Fastweb with delicate legal questions that

require, for instance, to obtain authorizations from customers for storing and managing their data, and to implement solutions to grant confidentiality and anonymity of such information.

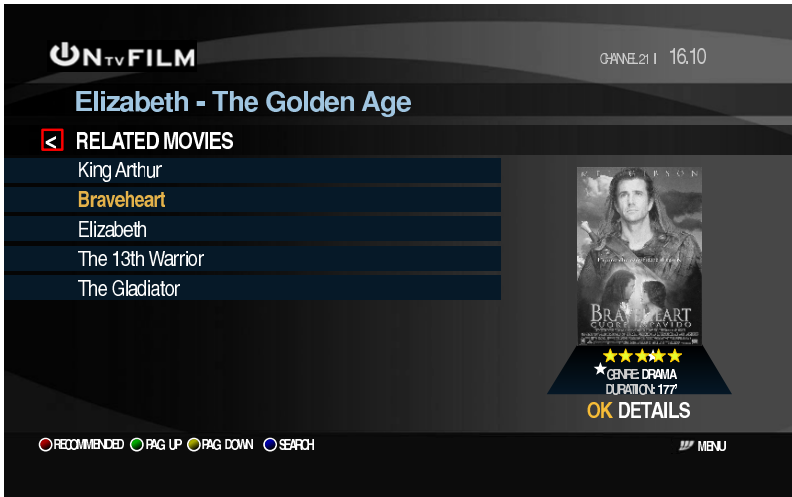


Fig. 9.6: Recommender system user interface

9.6 System Evaluation

In this section we first discuss the quality of the recommender system (see Chapter 8) by means of accuracy metrics computed adopting an off-line testing. We later present some feedbacks from the on-line analysis of the recommender system.

The off-line tests are based on the views collected during 7 months of users' activity from one of the VOD catalogs. Figure 9.7 shows the evolution of the number of views. The average number of views per days is about 1600, with up to 3300 views during week-end days. Figure 9.8, 9.9, and 9.10 complete the analysis by showing the time evolution of, respectively, the number of active users, the number of active items, and the dataset density. Active users are users that have rated at least one item. Similarly, active items are items that have been rated by at least one user. The dataset density is the ratio between the number of ratings and the product of the number of active users and the number of active items. We can notice from Figure 9.10 that the trend is not monotone. In fact, when a new user watches her/his first item, we have one new active user, and the dataset decrease its density.

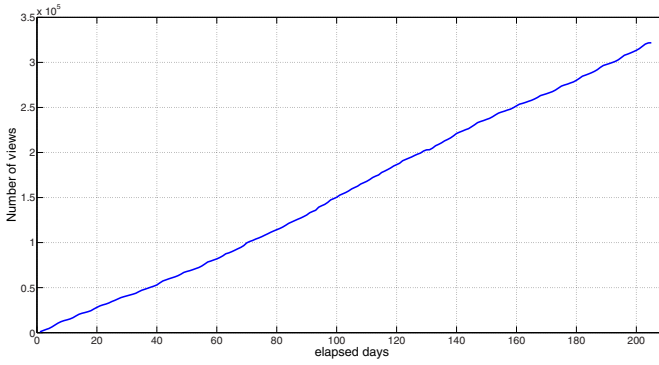


Fig. 9.7: Number of views collected during 7 months of users' activity from one of the VOD catalogs.

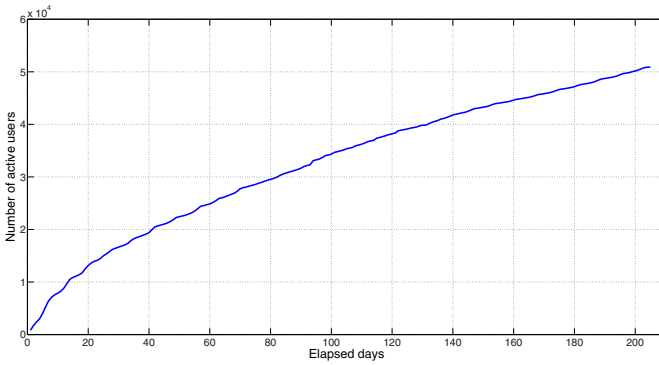


Fig. 9.8: Number of active users from the same VOD catalog.

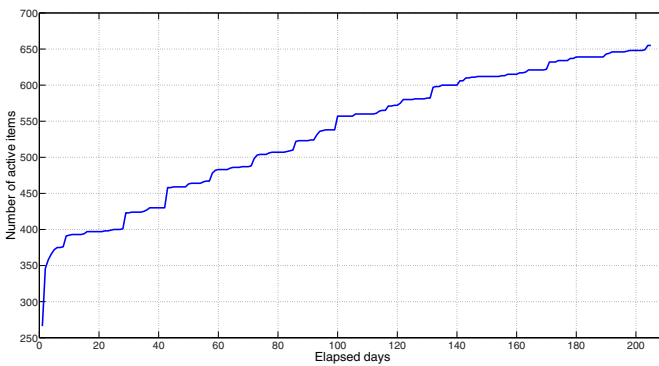


Fig. 9.9: Number of active items from the same VOD catalog.

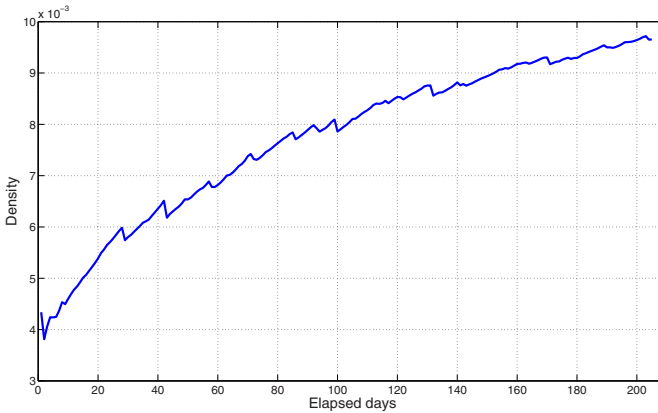


Fig. 9.10: Evolution of rating density in the same VOD catalog. Density is computed by considering the number of views (i.e., ratings) with respect to the number of active users and active items.

9.6.1 Off-Line Analysis

Typical approaches for recommender system evaluation are based either on error metrics (e.g., RMSE and MAE) [22] or classification accuracy metrics (e.g., recall, precision, and fall-out) [15, 7]. Since only implicit ratings are at disposal, expressing positive user interests, we are practically constrained in evaluating the quality of the system by means of accuracy metrics. To this end, Tables 9.1 and 9.2 present the recall of the three algorithms described in Sections 9.4.2, 9.4.3, and 9.4.4, respectively: the LSA-CB, the item-based-CF and the SVD-CF algorithms.

Recall is often used in information retrieval, where it specifies the percentage of relevant items that have been retrieved by, for instance, a search engine. In our domain, recall indicates how many movies that users have effectively watched are recommended by the recommender algorithm. To this purpose, we follow a leave-one-out approach:

- for each user in the test set, we select one rated item
- the selected item is removed from the user profile, and we generate a recommendation for this modified user profile; items already rated by the user are filtered out from the recommendation list.
- if the removed item is recommended within the first 5 positions we have a *hit*, i.e., a movie that has been watched by a user has been recommended by the algorithm (accordingly to the Fastweb user interface, the recommended list is limited to 5 items);
- the process is repeated for each item and for each user.

The recall is the percentage of hits with respect to the number of tests.

The test set is selected differently according to the kind of algorithm. In fact, content-based algorithms build their model by using the ICM, and the test set can be the whole URM. On the other hand, the model of collaborative algorithms is based on the URM itself, so they have been evaluated with a 10-fold cross validation approach, i.e., we have randomly split the users into 10 folds and, in turn, one fold has been used as test set for computing the recall, while the remaining nine folds have been used to generate the model. Each test fold is analyzed by means of the leave-one-out approach. The reported results are the average recall among the 10 folds.

The tables report the recall of the recommender algorithms both after 3 months of activity and after 6 months of activity, showing the time evolution of the system. Furthermore, the quality of recommendation of the three algorithms described in Section 9.4 are compared with a trivial algorithm, used only for comparison purposes: the *top-rated*. The top-rated algorithm is a basic collaborative algorithm that recommends to any user a fix list of items, ordered from the most popular to the less popular (discarding items already rated by the user).

Algorithm	Parameter	Recall	
		3 months	6 months
Item-based-CF	$k = 10$	16.8%	14.9%
	$k = 50$	18.7%	16.4%
	$k = 100$	19.0%	16.6%
	$k = 150$	18.8%	16.5%
SVD-CF	$l = 5$	15.1%	12.7%
	$l = 15$	12.6%	13.3%
	$l = 25$	10.9%	11.5%
	$l = 50$	9.3%	9.9%
	$l = 100$	6.3%	8.0%
LSA-CB	$l = 50$	1.9%	1.7%
	$l = 100$	2.3%	2.3%
	$l = 150$	2.4%	2.4%
	$l = 200$	2.5%	2.5%
Top-rated		12.2%	7.7%

Table 9.1: Recommendation quality concerning the considered VOD catalog.

For instance, Table 9.1 shows that during these 6 months of activity the best algorithm is the item-based collaborative algorithm, and the best configuration is with a neighborhood size k equals to 100. From Table 9.1 we can observe some particular aspects:

1. in some cases the quality of recommendations after 6 months is lower than after 3 months;
2. the quality of the top-rated algorithm is fairly good;
3. the quality of the content-based is poor, even less than the top-rated algorithm.

As for the first observation, we expect that as the system collects ratings, it acquires more precise user profiles and the quality of recommendations should im-

proves. However, this is not always true, as, for instance, [7] shows about a family of item-based collaborative algorithms based on naive Bayesian networks (NBN). Furthermore, the analysis we are conducting is not taking into consideration that time evolution concerns not only the ratings, but the items too. Indeed, after 3 months of activity there are 510 active items, while after 6 months we have 621 active items. In terms of probability, after 3 months an algorithm has to pick up 1 items among 510 candidates, while after 6 months the number of candidates is 621, as shown in Figure 9.9. As a partial counter-effect, while active items are increasing, users rate more items, and algorithms discard these items. Anyway, this minimally compensates the item-increase effect. In fact, while active items increase from 510 to 621, the average profile length increases from about 3 items per user to about 6 items per user, as shown in Figure 9.11.

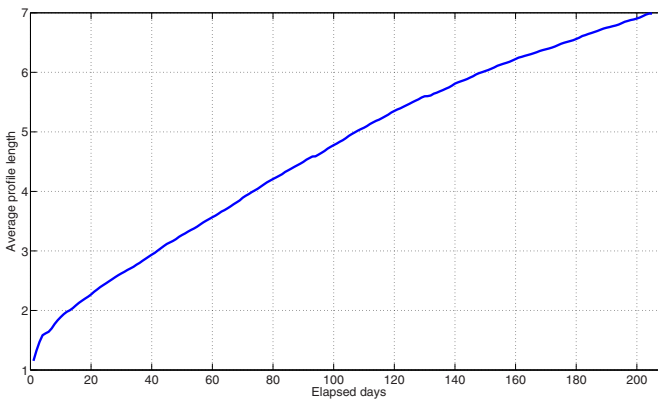


Fig. 9.11: Time evolution of the average user profile length. Profile lengths are computed on users active in one of the VOD catalogs.

As for the second and the third observations, they find a common explanation. The scarce quality of the content-based algorithm and the high quality of the top-rated algorithm partially depend on the testing methodology based on the leave-one-out approach. Indeed, the recall resulting from leave-one-out is biased toward the recall of the algorithm on popular items, since they are the most numerous, so the most tested. Content-based is extremely poor because it disregards item popularity. On the other hand, top-rated is particularly advantaged because, when the user profiles are short (e.g., during the cold start), most of the users have probably watched the most popular items, as shown in [7]. Furthermore, often users expect novel possibilities from a recommender system and recommending popular items does not address this concept known as *serendipity* [15].

For the above reasons, we present in the following a further evaluation of the quality of the recommender algorithms, where the most popular items have been excluded from the tests and the recall is computed only on the non-popular items, addressing the well-know concept referred to as long-tail [2]. Figure 9.12 illustrates

the distribution of ratings between popular and unpopular items. For instance, we can observe that about the 50% of ratings is concentrated in the 10% of the most-popular items (short-head), while the remaining 90% of items (long-tail) refers only to the 50% of ratings: one of the primary reason for integrating a recommender system is to push up the sells of long-tail items, since they represent potential incoming for a service provider. However, recommending long-tail items is harder than recommending short-head items.

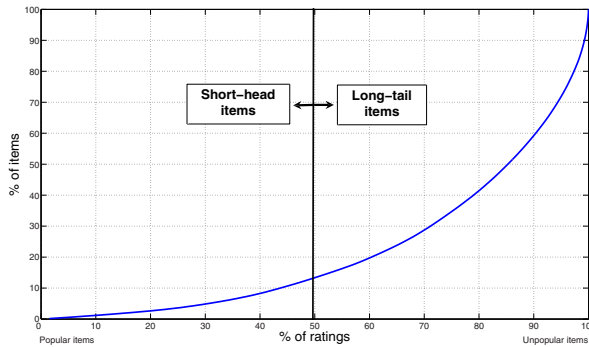


Fig. 9.12: Long-tail effect: 50% of ratings concerns 10-12% of popular items (short head).

Table 9.2 reports the recall when the 10 most-popular items have been discarded from testing (referred to as non-top-10), and the recall when the short-head (most-popular) items, representing the 50% of the total number of ratings, have been discarded from testing (referred to as non-top-50%).

From Table 9.2 we can note that:

1. the quality of the content-based algorithm is constant;
2. collaborative algorithms decrease their quality when recommending unpopular items, and top-rated fails;
3. unpopular items are better recommended by the dimensionality-reduction-based collaborative algorithm than by the item-based collaborative algorithm.

As for the first observation, the content-based algorithm is confirmed not to be affected by item popularity.

On the contrary, the recall of collaborative algorithms decreases. Among them, the top-rated algorithm quality drastically falls down and, in fact, top-rated is not able to recommend long-tail items.

Moreover, we can observe that for recommending non-top-10 items the best algorithm is again the item-based collaborative algorithm. However, when we focus on the long-tail (non-top-50%), the dimensionality-reduction-based collaborative algorithm overtakes the item-based. Again, we can observe that the dimensionality-

Algorithm	Parameter	Recall non-top-10		Recall non-top-50%	
		3 months	6 months	3 months	6 months
Item-based-CF	$k = 10$	14.0%	13.2%	7.7%	9.6%
	$k = 50$	14.0%	13.8%	6.8%	9.0%
	$k = 100$	13.8%	13.5%	6.2%	8.3%
	$k = 150$	13.5%	13.2%	6.1%	7.9%
SVD-CF	$l = 5$	6.6%	6.8%	0.7%	1.4%
	$l = 15$	11.5%	10.2%	1.2%	3.5%
	$l = 25$	12.6%	12.0%	2.2%	4.9%
	$l = 50$	11.4%	11.2%	4.8%	7.8%
	$l = 100$	7.6%	9.3%	9.8%	11.8%
LSA-CB	$l = 50$	2.1%	1.8%	1.8%	1.7%
	$l = 100$	2.3%	2.3%	2.0%	2.5%
	$l = 150$	2.5%	2.5%	2.1%	2.5%
	$l = 200$	2.6%	2.6%	2.2%	2.6%
Top-rated		0.4%	1.0%	0%	0%

Table 9.2: Recommendation quality in one of the VOD catalogs for long-tail items, i.e., items not in the top-10 and not in the top-50%, respectively.

reduction-based collaborative algorithm follows a positive trend as the system collects more ratings, increasing its capability in recommending unpopular items.

9.6.2 On-line Analysis

In this section we integrate the previous results, obtained from an off-line analysis of the recommender algorithms, with an on-line analysis, i.e., we directly study the feedback on the running recommender system. As explained in Section 9.5, the reported data refer to the content-based algorithm applied on one of the VOD catalogs.

In order to empirically evaluate the recall, we assume that whether a user watches a movie after it has been recommended by the system, such movie is relevant for the user and this represents a *success* for the recommender system.

Let us define the *recommendation success*, which measure the number of movies that have been viewed within a certain time period after being recommended. Indicating with $b(t)$ the recommendation success and with $w(t)$ the number of movies watched by the same users within a time period t from a recommendation, we can compute an *empirical recall* as the percentage ratio between the recommendation success and the number of views:

$$\text{empirical recall}(t) = \frac{b(t)}{w(t)} \tag{9.18}$$

The empirical recall represents the percentage of views that have been triggered by the recommender algorithm. The specified indexes depend on the time period t that

is taken into consideration after the recommendation has been provided to the user. Please note that a too long time period t could loose the dependency between the recommendation and the view. Table 9.3 shows the average quality of the system computed by monitoring the views within 2 hours, within 24 hours, and within 7 days from the recommendation. The reported results distinguish between popular and unpopular items.

From the table we can observe that the empiric recall is larger for unpopular movies with respect to popular movies. In fact, popular movies are already known by users, even without being suggested by the recommender system. For instance, either the user has already watched a popular movie (e.g., at cinema) or it is not interested in it.

As a further analysis, about 64% of the recommendation successes refers to unpopular movies (i.e., non-top 50%), while only 36% refers to popular movies (i.e., top 50%), i.e., the recommender system is stimulating users to watch unpopular movies, with a positive effect on the long-tail.

	2 hours	24 hours	7 days
All	17.0%	19.8%	24.7%
Top 10	5.1%	7.0%	10.6%
Non-top 10	24.2%	27.6%	32.1%
Top 50%	9.4%	11.5%	16.2%
Non-top 50%	28.4%	32.2%	36.1%

Table 9.3: Average empiric recall on the considered VOD catalog. Results refer to three time periods after the recommendation (2 hours, 24 hours, and 7 days) and are separated between popular and unpopular movies.

Moreover, we highlight the benefits of the recommender system by measuring the *lift factor* that it introduces in the number of views, i.e., the increase of views due to the recommender system. Generally speaking, the number of views in IPTV systems depends on the size of the customer base. Furthermore, we have to take into consideration that new users tend to view more movies than existing users. In addition to a constant incoming of new users, we have bursts of new users corresponding to marketing campaigns. For instance, Figure 9.13 shows the trend of the whole Fastweb customer base during more than two-year activity. The steep parts of new users are related to promotional campaigns. For privacy reasons, the real number of users is hidden and replaced with a proportional value.

In order to describe the correlation between users and views, we have defined an autoregressive moving average (ARMAX) model, whose inputs are the current size of the customer base and the number of new users. The parameters of the ARMAX model are estimated and validated by considering 50 weeks of users' activity before the integration of ContentWise. Figure 9.14 compares the actual number of views with the number of views estimated by the model. In order to smooth daily variability, views are aggregated by week. Splitting the data into training and validation sets, the RMSE on the validation set results below 2%.

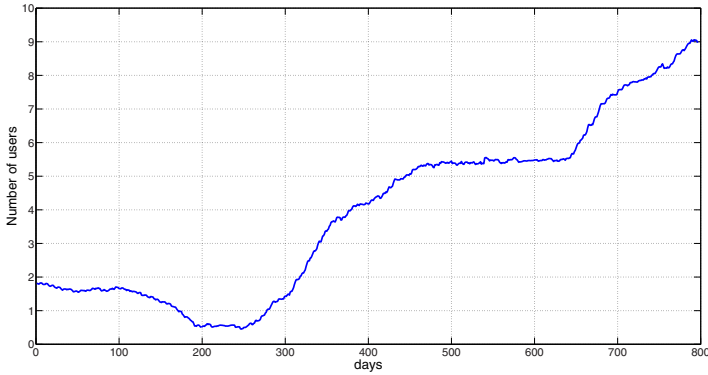


Fig. 9.13: Number of Fastweb users. The real number of users is proportional to the reported value.

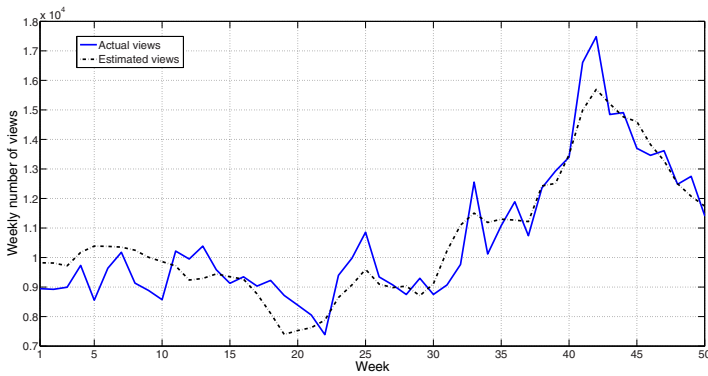


Fig. 9.14: Weekly number of views *before* the introduction of ContentWise.

The model is then used to estimate the number of views in the first 20 weeks after the integration of the recommender system. As shown in Figure 9.15, we have an increase of views with respect to the number of views estimated by the model, and this increase can be attributed to the impact of the recommender system, since the other potential factors (e.g., marketing campaigns) are included into the ARMAX model. On average, the lift factor within this period is equals to 15.5%.

Finally, we analyze how users look for interesting content in the considered VOD catalog. Figure 9.16 shows the daily number of search requests by means of the recommender system, the keyword-based search engine, and the alphabetic browsing, respectively. The gap between the requests to the recommender system and the requests to the other searching tools indicates that users effectively utilize the recommender algorithm to search for movies.

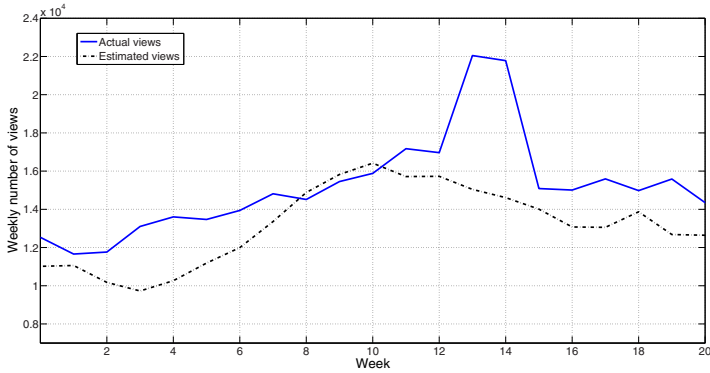


Fig. 9.15: Weekly number of views *after* the introduction of ContentWise.

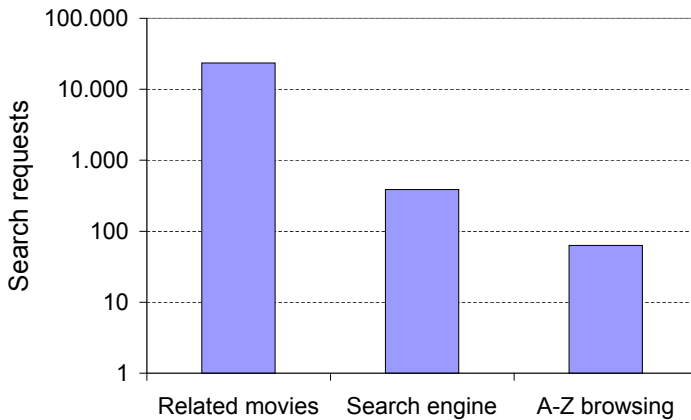


Fig. 9.16: Comparison among different ways of searching for interesting content: the recommender system (related movies), the keyword-based search engine, and the alphabetic browsing. Values are reported in a logarithmic scale.

9.7 Conclusions

The integration of the ContentWise recommender systems into the Fastweb architecture positively impacts both the customers and the service provider. Three major considerations derive from the on-line analysis, confirming the positive effects of the recommender system: (i) users prefer to browse the VOD catalog by means of the recommender interface, (ii) users tend to watch recommended movies within few hours, and (iii) users increase the number of watched movies.

Further experiments are currently running on the other catalogs of Fastweb, testing and tuning the quality of all the implemented recommender algorithms and monitoring the cold-start phase of the system in order to complete the release of recommender services. Other ongoing works are addressing the problem of accurately estimating implicit ratings from user behavior.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on* **17**(6), 734–749 (2005). DOI 10.1109/TKDE.2005.99
2. Anderson, C.: *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion (2006). URL <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{\&}path=ASIN/1401302378>
3. Balabanović, M., Shoham, Y.: Fab: content-based, collaborative recommendation. *Commun. ACM* **40**(3), 66–72 (1997). DOI <http://doi.acm.org/10.1145/245108.245124>
4. Bell, R.M., Koren, Y.: Scalable collaborative filtering with jointly derived neighborhood interpolation weights. *7th IEEE Int. Conf. on Data Mining* pp. 43–52 (2007)
5. Berry, M.W.: Large-scale sparse singular value computations. *The International Journal of Supercomputer Applications* **6**(1), 13–49 (1992). URL citeseer.ist.psu.edu/berry92large.html
6. Chai, K.M.A., Chieu, H.L., Ng, H.T.: Bayesian online classifiers for text classification and filtering pp. 97–104 (2002). DOI <http://doi.acm.org/10.1145/564376.564395>
7. Cremonesi, P., Lentini, E., Matteucci, M., Turrin, R.: An evaluation methodology for recommender systems. *4th Int. Conf. on Automated Solutions for Cross Media Content and Multi-channel Distribution* pp. 224–231 (2008)
8. Deerwester, S.C., Dumais, S.T., Landauer, T.K., Furnas, G.W., Harshman, R.A.: Indexing by latent semantic analysis. *Journal of the American Society of Information Science* **41**(6), 391–407 (1990). URL <http://citeseer.ist.psu.edu/deerwester90indexing.html>
9. Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)* **22**(1), 143–177 (2004). DOI <http://doi.acm.org/10.1145/963770.963776>
10. Furnas, G.W., Deerwester, S., Dumais, S.T., Landauer, T.K., Harshman, R.A., Streeter, L.A., Lochbaum, K.E.: Information retrieval using a singular value decomposition model of latent semantic structure. pp. 465–480. ACM Press, New York, NY, USA (1988). DOI <http://doi.acm.org/10.1145/62437.62487>
11. Geneve, U.D., Marchand-maillet, S.: Vision content-based video retrieval: An overview

12. Gorrell, G.: Generalized Hebbian Algorithm for Incremental Singular Value Decomposition in Natural Language Processing. 11th Conference of the European Chapter of the Association for Computational Linguistics (2006)
13. Hand, S., Varan, D.: Interactive narratives: Exploring the links between empathy, interactivity and structure pp. 11–19 (2008)
14. Herlocker, J., Konstan, J., Riedl, J.: An algorithmic framework for performing collaborative filtering. 22nd ACM SIGIR Conf. on R&D in Information Retrieval pp. 230–237 (1999)
15. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* **22**(1), 5–53 (2004)
16. Husbands, P., Simon, H., Ding, C.: On the use of singular value decomposition for text retrieval (2000). URL citeseer.ist.psu.edu/article/husbands00use.html
17. Jensen, J.F.: Interactive television - a brief media history **5066**, 1–10 (2008)
18. Kelly, D., Teevan, J.: Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum* **37**(2), 18–28 (2003). DOI <http://doi.acm.org/10.1145/959258.959260>
19. Lee, Y., Lee, J., Kim, I., Shin, H.: Reducing iptv channel switching time using h.264 scalable video coding. *Consumer Electronics, IEEE Transactions on* **54**(2), 912–919 (2008). DOI [10.1109/TCE.2008.4560178](http://doi.acm.org/10.1109/TCE.2008.4560178)
20. Paterek, A.: Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop* (2007)
21. Pazzani, M., Billsus, D.: Content-based recommendation systems. *The Adaptive Web: Methods and Strategies of Web Personalization, Lecture Notes in Computer Science* pp. 325–341 (2006)
22. Powers, D.M.W.: Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness and Correlation (2000)
23. Rafey, R.A., Gibbs, S., Hoch, M., Gong, H.L.V., Wang, S.: Enabling custom enhancements in digital sports broadcasts pp. 101–107 (2001). DOI <http://doi.acm.org/10.1145/363361.363384>
24. Rokach, L., Maimon, O., Averbuch, M., *Information Retrieval System for Medical Narrative Reports, Lecture Notes in Artificial intelligence 3055*, page 217-228 Springer-Verlag (2004)
25. Saad, Y.: Numerical methods for large eigenvalue problems. Halsted Press New York (1992)
26. Salton, G. (ed.): Automatic text processing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1988)
27. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Item-based collaborative filtering recommendation algorithms. 10th Int. Conf. on World Wide Web pp. 285–295 (2001)
28. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Application of Dimensionality Reduction in Recommender System-A Case Study. *Defense Technical Information Center* (2000)
29. Schafer, J., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. pp. 291–324 (2007)
30. Sun, J., Gao, S.: Iptv based on ip network and streaming media service station. *MIPPR 2007: Remote Sensing and GIS Data Processing and Applications; and Innovative Multi-spectral Technology and Applications* **6790**(1), 67904Q (2007). DOI [10.1117/12.749611](http://dx.doi.org/10.1117/12.749611). URL <http://link.aip.org/link/?PSI/6790/67904Q/1>
31. Valle-Lisboa, J.C., Mizraji, E.: The uncovering of hidden structures by latent semantic analysis. *Inf. Sci.* **177**(19), 4122–4147 (2007). DOI <http://dx.doi.org/10.1016/j.ins.2007.04.007>
32. Van Rijsbergen, C.J.: *Information Retrieval*, 2nd edition. Dept. of Computer Science, University of Glasgow (1979). URL citeseer.ist.psu.edu/vanrijsbergen79information.html
33. Vozalis, E., Margaritis, K.: Analysis of recommender systems algorithms. *Proc. of the 6th Hellenic European Conf. on Computer Mathematics and its Applications* (2003)
34. Wang, J., de Vries, A.P., Reinders, M.J.T.: Unifying user-based and item-based collaborative filtering approaches by similarity fusion. pp. 501–508. *ACM Press, New York, NY, USA* (2006). DOI <http://doi.acm.org/10.1145/1148170.1148257>

35. Zhang, H., Zheng, S., Yuan, J.: A personalized tv guide system compliant with mhp. *Consumer Electronics, IEEE Transactions on* **51**(2), 731–737 (2005). DOI 10.1109/TCE.2005.1468026
36. Zhang, X., Berry, M.W., Raghavan, P.: Level search schemes for information filtering and retrieval. *Information Processing and Management* **37**(2), 313–334 (2001). DOI [http://dx.doi.org/10.1016/S0306-4573\(00\)00032-7](http://dx.doi.org/10.1016/S0306-4573(00)00032-7)

Chapter 10

How to Get the Recommender Out of the Lab?

J erome Picault, Myriam Rib iere, David Bonnefoy and Kevin Mercer

Abstract A personalised system is a complex system made of many interacting parts, from data ingestion to presenting the results to the users. A plethora of methods, tools, algorithms and approaches exist for each piece of such a system: many data and metadata processing methods, many user models, many filtering techniques, many accuracy metrics, many personalisation levels. In addition, a real-world recommender is a piece of an even larger and more complex environment on which there is little control: often the recommender is part of a larger application introducing constraints for the design of the recommender, e.g. the data may not be in a suitable format, or the environment may impose some architectural or privacy constraints. This can make the task of building such a recommender system daunting, and it is easy to make errors. Based on the experience of the authors and the study of other works, this chapter intends to be a guide on the design, implementation and evaluation of personalised systems. It presents the different aspects that must be studied before the design is even started, and how to avoid pitfalls, in a hands-on approach. The chapter presents the main factors to take into account to design a recommender system, and illustrates them through case studies of existing systems to help navigate in the many and complex choices that have to be faced.

J erome Picault
Alcatel-Lucent Bell Labs, e-mail: jerome.picault@alcatel-lucent.com

Myriam Rib iere
Alcatel-Lucent Bell Labs, e-mail: myriam.ribiere@alcatel-lucent.com

David Bonnefoy
Pearltrees, e-mail: david.bonnefoy@pearltrees.com

Kevin Mercer
Loughborough University, e-mail: K.C.Mercer@lboro.ac.uk

10.1 Introduction

A personalised system is a complex piece of software made of many interacting parts, from data ingestion to presenting the results to the users. A plethora of methods, tools, algorithms and approaches exist for each piece of such a system: many data and metadata processing methods, many user models, many filtering techniques, many accuracy metrics, many personalisation levels. . . In addition, a real-world recommender is a piece of an even larger and more complex environment over which there is little control: often it is part of a larger application introducing constraints for the design of the recommender, e.g. the data may not be in a suitable format, or the environment may impose some architectural or privacy constraints. This can make the task of building such a recommender system daunting.

This chapter intends to be a guide to the design, implementation and evaluation of personalised systems. It will present the different aspects that must be studied before the design is even started, and how to avoid pitfalls, in a hands-on approach.

10.2 Designing Real-World Recommender Systems

Previous work in the literature provides guidelines on many aspects of building a recommender system. For example, [49] lists some characteristics and general principles that should drive a personalised system design, such as taking into account content specificity, importance of trust in the system and of involving users. [25] provides an extensive analysis of methods and metrics for evaluating collaborative filtering systems, including also a taxonomy of user tasks for recommender systems, and an interesting description of dataset properties. This work is extremely useful once initial technological choices have been made (user model, choice of algorithm, etc.). But *how can we make sensible choices when initially designing the system?* This is a major concern as any change later in the development is costly.

In order to tackle this problem in a systematic way, it is useful to step back and see from a wider perspective what are the main design decisions to make and the factors which influence them. Fig. 10.1 illustrates the approach suggested in this chapter.

Designing a recommender system means making choices that can be categorised into the following domains:

- Algorithms: which recommendation methods to use ;
- Architecture: how will the system be deployed, will it be centralised or distributed?
- User profile: what is the user model, is profile adaptation needed?

For a large part, these choices are constrained by the environment of the recommender. It is thus important to systematically study the environment the system will be situated in. We propose to describe it along three dimensions:

- Users: who are the users, what are their goals?

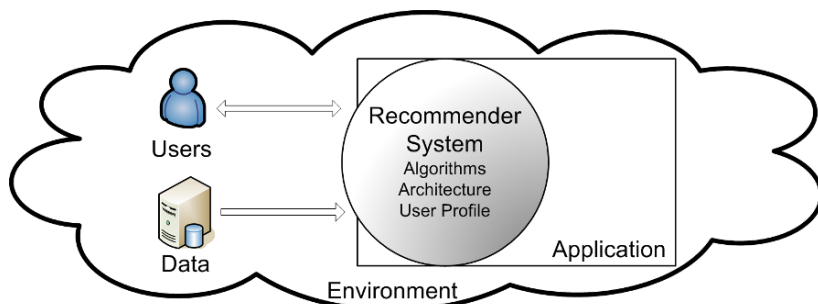


Fig. 10.1: The Recommender in its environment

- Data: what are the characteristics of the data on which recommendations are based?
- Application: what is the overall application the recommender is part of?

We propose to build a model of the environment based on these three dimensions, and base the recommender system design on these models.

The following sections of this chapter will describe this process. The next section first describes the three models that should be built prior to the system design and how they affect the design decisions. In section 10.4 we will then show how these models can help in evaluating the system. The section 10.5 will finally present a use-case of the methodology.

10.3 Understanding the Recommender Environment

As mentioned in the previous section, we propose to define three models (user, data, and application). These models will assist the recommender designer in decision making processes, helping them understand the key constraints of their future system, ask themselves the right questions and define constraints for making decisions about three main aspects: choice of the recommendation algorithm, choice about the recommender system architecture and choice in the possible adaptation of the user profile. Our methodology to define the environment models consists in defining key aspects of each model and the key questions to be asked throughout the process.

10.3.1 Application Model

Though a recommender system is itself a complex piece of software, it is by nature part of a larger system. A recommender is one of the features of an overall application. It may be a minor feature or a main selling point; the application may

be pre-existing or built together with the recommender, but in any case the design of a recommender system has to be integrated within the design of the application hosting it. This section studies the main factors regarding the host application that should influence the recommender design along two main lines: the role of the recommender and the influence of the application implementation (Table 10.1).

Table 10.1: Application model.

Model's features	Possible values
Recommender purpose	main service, long-tail focused, increase revenues, increase loyalty, increase system efficiency
Recommender type	single item, multiple items, sequence
Integration with navigation features	yes, no
Performance criteria (correctness, transparency, serendipity, risk-taking, response speed, robustness to attack)	performance target on each criterion
Device to support the application	fixed, mobile, multiple
Number of users	single, group
Application infrastructure	browser-based application, distributed application
Screen real-estate	limited, not limited

10.3.1.1 Understanding the recommender role in the application

The main question to solve before designing a recommender system is to be very clear on its goals within the overall application. This is often not as easy as it seems, and can have fundamental consequences on the type of system to be built. Two perspectives have to be studied: the application point of view and the user point of view. They overlap, but are still separate. The user point of view is studied in detail in section 10.3.2. This section focuses on the application point of view.

Purpose of the recommender: From the application side, a recommender system may have different purposes, for instance:

- It may be a *major service* provided by the application. Many such recommender systems have been developed in different fields such as music (Pandora¹, last.fm², MyStrands³ ...) or movies (MovieLens⁴, Netflix⁵ ...)
- To *take advantage of the 'Long Tail'*, as first described by Chris Anderson in [5]. The idea that recommendations can give easy access to previously hard

¹ Pandora Internet Radio : <http://www.pandora.com>

² Last FM : <http://www.last.fm>

³ MyStrands, Social Recommendation and Discovery : <http://www.mystrands.com/>

⁴ MovieLens, Movies Recommendations : <http://www.movielens.org/>

⁵ Netflix : <http://www.netflix.com/>

to find items is central to the business model of many e-commerce web sites. In that case, the recommendations have to be focused on lesser known items, accurately tailored to each user.

- To *increase loyalty* of users: customers return to the services that best match their needs. Loyalty can be increased by involving users in the recommendation process (ask for ratings or manual profile, highlight new recommendations, etc.)
- To *increase revenues* through the promotion of targeted products. In that case, the recommendations would be determined both by the user preferences and some marketing rules defined to follow a particular strategy. It is necessary to carefully balance the expectations of the users and the business strategy, to ensure users perceive value in the system.
- To *increase system efficiency*. By allowing the user to more directly get the content he is looking for, a recommender system can lower the amount of data to be exchanged, thus lowering the costs of running a system.

Recommendation type: a recommender can provide several sorts of recommendations, from a single item (or a simple list of items) to a sequence (e.g. in a travel recommender system). Single item or simple list recommenders do not take into account how the choice of an item by the user at a given point of time may influence the choice of next items. The choice of the recommendation type may be driven by the need or presence of a logical order in the recommendations. For example, in a travel recommender system, a trip can be seen as a sequence of travel steps (such as visiting a museum, going to the beach, etc.), which can be connected through various logical features, such as geography, culture, history, leisure, etc. in order to provide a good travel experience. Recommendations of sequences of items may be particularly useful when users are new to a domain and need a path in the selection of diverse items, helping them to go through their personal development goals: the logical order of the recommendations help them progressing in their learning curve by providing the next most appropriate step(s). Unfortunately, there is to date little work on recommenders of sequences. Some of our ongoing work is addressing this issue; other techniques coming from data mining domain, such as the Apriori algorithm [2], may be used.

Integration with content navigation features: Another key point to study is how the recommendations will integrate with other content navigation features. In most cases, users will be offered other means to browse content in addition to getting recommendations. A good integration of these different navigation methods can greatly enhance the user experience.

- Users may request recommendations completely separately from content browsing. This can be a good choice if recommendations are to be highlighted as a main feature of the application. Such recommendations may also appear on the home page of a web site or home screen of an application.
- It can also be beneficial to have recommendations dependant on the current interaction context. The typical case is to recommend items that are similar to those the user is currently browsing. In that case, the recommender system must be able to provide recommendations tailored to the context, e.g. the current genre when browsing music.

- It is also important to consider whether the use of the recommender system is optional or a mandatory part of the interaction model. This has strong implications on the expected reliability of the system: failure to complete a major task on a website because the only way of completing that task was using a recommender system which offered inaccurate recommendations could be a source of major user dissatisfaction. However within a system design where the recommender sat in parallel to more traditional navigation methods, the impact of the same recommender may be many times less severe.

Performance criteria: Once these goals are clarified, it is possible to define targets for the performance of the system along a number of criteria. Not only these criteria will allow evaluating the system once it is built, but they are also key to selecting the proper algorithms. Many criteria can be used, see [25] for a comprehensive reference of many possible criteria. Some key ones could include:

- *Correctness metrics*, such as accuracy, precision and recall: these are the technical criteria that can be used to evaluate recommendation algorithms, and have been the focus of many studies over the years. However, they are actually not sufficient to evaluate user satisfaction [33].
- *Transparency and explainability*: how important is it that users understand how the recommendations have been determined? A good level of transparency can be more difficult to achieve with some families of algorithms. For instance, collaborative filtering offers little transparency naturally, but [24] proposes an analysis of the problem and some solutions.
- *Serendipity*: should users be (hopefully pleasantly) surprised by some of the recommendations or is it desirable to allow obvious recommendations? Canonical collaborative filtering tends to recommend items that are very popular, and may be considered obvious and of little use to most users. Techniques exist to correct this tendency; some are described in [25].
- *Risk taking*: related to the previous criterion, should the recommendations be made only for items that the user has a high probability of liking? More risky items can be recommended if the goal is to allow the user to discover content they would not be aware of without the help of the system.
- *Response speed / performance*: in many cases, the reactivity of the application is a major concern and can be sometimes more important than the accuracy of the results. Knowing how many recommendations are needed per time unit allows to better choose algorithms or decide if recommendations should be pre-computed.
- *Reliability*: What is the criticality of the recommender output in the context of the given application? For instance the design of a recommender for an e-commerce website would not be approached in the same way as a solution for an organ donor matching system in a hospital.
- *Robustness to attacks*: in particular if the recommender system has a commercial role (for instance if it recommends products for purchase), it may be subject to attacks to skew the results. [34] presents a thorough analysis of possible attacks and some solutions for collaborative filtering algorithms.

10.3.1.2 Understanding the influence of the application implementation

In addition to the features seen by the users, some aspects of the application implementation also have a large influence on how the recommender can be designed.

Single or multiple devices: the same application may be accessed from a single or multiple devices (e.g. a news recommender system on mobile, PC, set-top box). It must be studied whether the recommendations should depend on the user context (see section 10.3.2). But in term of implementation, it also raises additional questions: should the collected preferences be merged or should they remain separate to enable contextual recommendations? Where should the preferences be stored? If the preferences are stored on a server, are they transmitted to the server in real-time (implying a constant connection) or in batches? Answering such questions is important even if access from multiple devices is not initially planned, as it is becoming the norm that web applications are derived into mobile versions.

Single or multiple users: conversely, the same device may be used by several users. The consequences of this user social environment are studied in section 10.3.2. In addition, users that interact with the personalised application may be either registered or anonymous and may interact frequently or occasionally. This impacts the architecture of the recommender (requires different identification means, e.g. login vs. cookies), algorithm choice (e.g. session profile in case of anonymous occasional users vs. persistent profile in case of registered users), and algorithm parameters (e.g. the degree of adaptability of the system to the user – i.e. the rapidity of profile adaptation: long-term vs. short-term – should depend on the frequency of use).

Application infrastructure: The infrastructure the application runs on puts strong constraints on the types of recommendation algorithms that can be used and on their specific implementation. In particular, the scalability of the solution has to be carefully studied. Two main cases can be identified, whether the application is accessed through a browser or if an application runs locally on the user device.

- *Browser-based application.* In the case of a browser-based application, the processing done on the client will be minimal. As all information is available at a single point, any kind of algorithm can be used. However, scalability will be a major focus in the design.
- *Distributed application.* When the user device runs a local application, different architectures may be used. To determine the most suitable distributed architecture, the following criteria must be studied:
 - Processing power of the relevant devices. Is the client device able to support intensive tasks? Can the server-side computing resources be extended as needed when the number of users grows?
 - Network connectivity. Is the network connection permanent? Does the data transfer have a cost for the users? For mobile devices, what is the impact of the connection to the battery life?
 - Data source. How are the data that the recommendations are drawn from accessed? Is it from a database (information retrieval) or a stream of data (information filtering)?

Content filtering algorithms may run entirely on the client device. This has several advantages: privacy is preserved as no personal information has to be transmitted; such an architecture is very scalable as the number of users has minimal impact on the server load. On the other hand, algorithms such as collaborative filtering require that information from all or large subsets of users be collated. In that case, a fully centralised architecture may be used with most of the same pros and cons as a browser-based application, but with the additional need for a mechanism to update the client application when needed. More sophisticated architectures can also be designed, such as in the case of the TV programme recommender TiVo [3]. In this system, a part of the computation is done server-side and another part is done on each set-top box. Other complex architectures include distributed profile management [13] or mechanisms to make different recommenders running on different devices communicate [28]. The choice of infrastructure may be influenced by other components the system should connect to such as external web services.

Screen real-estate: a point that is easily overlooked in the early stages of the design is how much screen space the recommendations will use. In many cases it is very limited and constrained by the application user interface design. This is not only a quantitative issue and can have influences on the very nature of the recommendations that are provided. For instance, if it is intended to provide more exploratory recommendations, it is necessary to have a sufficient number of recommendations and thus sufficient space. The same problem may arise for example in the display of a recommendation profile to a user; a solution to display the user profile on a small device has been proposed [40].

The study of the application the recommender system will be part of brings a first set of constraints on the design of the recommender, but on its own it is not enough to build an appropriate system. This additionally requires knowledge about both the user and the data.

10.3.2 User Model

Fully understanding the user is a fundamental component to the success of any recommender system. Insights into the end users which are to be built into the user model must come early enough in the development lifecycle to influence major design decisions surrounding the selection of technology. Applying a user-centred approach to any project within the initial phases can greatly reduce the need for extensive redesign, maintenance and customer support [9, 22].

In this section, we propose to characterize users by a number of properties that may have an impact on the recommender system design and choices the designer will have to face (Table 10.2).

At a fundamental level the aspects of the user which must be understood when developing a recommender revolve around best practices for understanding and specifying the context of use for an interactive system in a human-centred way [26]: Who

Table 10.2: User model.

Model's features	Possible values
Demographics information	yes, no
Goal existence	yes, no
Goal nature	implicit, explicit
Level of expectation	high, medium, low
Handling change of expectation over time	yes, no
Limited capabilities of user device	yes, no
Importance of user situation	high, medium, low
Social environment	alone, with others
Trust and privacy concerns	high, medium, low

are the end users? What expectations and goals lie behind the users motivations to use the system the recommender supports? What are the user centric contextual factors surrounding use of the system? In fully answering each *context of use* question, fundamental requirements for the system design and technology choices will be uncovered.

10.3.2.1 Understanding who the users are

Understanding who the users of the recommender system will be should revolve around three main concerns: understanding their key identifying characteristics, their skill levels and their prior experience with similar systems. We concentrate on the identification of user characteristics because it has special utility in terms of recommender design.

Identifying User Characteristics: Gathering a picture of the different user groups through both demographic information such as age, gender, job area, nationalities, spoken languages, and deep qualitative insights from user research are an important jumping off point in the development of recommender user models. Understanding these factors allows the development team to start to build a relationship with the users and get an appreciation of their needs.

Development of user group clusters may allow (1) the building of simple recommenders based on demographics. This is commonly used in targeted advertising solutions to cluster customers into segments [23]; (2) define stereotypes of users [42]: stereotyping techniques allow the definition of a set of differentiating characteristics for a group of users; when a new user is introduced into the system, they can be assigned to a predefined stereotype, based on their personal data, which allows the activation of a set of default preferences that may be further refined over time thanks to user profile adaptation methods [17]. Personalisation solutions exploiting user characteristics can be used in combination with more sophisticated techniques to provide a first simple step in a hybrid filtering process, or to bootstrap a content based filtering algorithm by using stereotype profiles.

In addition, the type of user (e.g. professional users vs. end users) is an essential criterion to help determine the user's level of expectations and so, to choose algorithms accordingly.

10.3.2.2 Understanding users' motivations, goals and expectations

Goals and motivations: The designer of the recommender system needs to identify the user tasks [25] and understand if the application can support completion. For example the Amazon web site's offering to the user revolves around the possibility to buy items and get recommendations for items to buy. From the user's viewpoint, their motivation for using the service is to complete one of the two goals of either buying an item for themselves, or buying an item for someone else. The Amazon recommender however, does not differentiate those two goals and therefore provides inaccurate recommendation results in one of the use cases. As another example, a search engine coupled with a content recommender can offer the opportunity for the user to browse the Internet and find information according to a request. The user in this context may be motivated by the need to complete a specific targeted goal, or their motivation may be simply to relax and spend some time browsing for fun. Identifying and understanding user motivation can result in fundamental recommender and user experience improvements. User insights captured within the user model (Table 10.2) allow designers to consider the possible range of tasks the future application needs to support.

In most cases there will be many motivations for the use of a system, and a designer must consider ways of finding the nature of the goal, either explicitly or implicitly. An *explicit* goal may be either defined by the application (Amazon could have added a button asking the user if they were buying the item for themselves or someone else) or expressed by the user (for example through a set of queries). An *implicit* goal may be either predefined by the application itself (such as in personalised news pushing systems where the system provides stories to the user in the belief that the user goal is to feel more informed about particular world events), or can be inferred from the user's interactions. In contrast it is quite possible that a clear task goal is not discernable (e.g. because the application is dedicated to enjoyment). In such cases it can be difficult to build a good recommender as user satisfaction may be more strongly related to external factors such as user mood outside of the system's control. In this case, spread, quantity or serendipity may be the most desirable qualities of the recommender.

The impact of the user's goal on filtering algorithms and diversity heuristics [55, 54] when displaying the results of a recommender is important and can generate user dissatisfaction if not addressed correctly at the design stage. For example a content-based method may be more adapted for focused people (because of the specialization of the results), whereas collaborative methods may be more adapted to people with less focused goals (because of the broader diversity of the results).

Users' expectations: The implicit or explicit goal of the user as described in the previous section is the key to evaluating the level of user expectation:

- *High expectation levels* would be seen in the context of goal-oriented users, who are focused on completing the current task. It means that the recommender must target an “all good items” [25] list of recommendations to achieve a high level of satisfaction for the user. This level of expectation is also linked to decisions made at the application level (see sec. 10.3.1), where the place of the recommender in the overall application and the constraints of the targeted device are central to the user expectations. Returning to the news pushing system as an example, that system needed to provide an “all good items” list. If the user cannot find the right personalised news in the first ten recommendations, they must at best start scrolling down to search through the content and at worst they reject the application never using it again because of their initial dissatisfaction at first use.
- *Medium expectation levels* can be seen as the recommender returning “some good items” [25]. If the user has choice and flexibility in the use of the recommender to complete their task, the user expectation is lowered. In this category we also find people that use the recommender purely to evaluate if the system corresponds to their expectation of what a recommender can bring to them. Some are looking for recommendations that are exactly aligned to their preferences; others would want to discover new suggestions that are different from their current habits.
- *Low expectation levels* are mainly a target for personalised applications used in an opportunistic context. As indicated by [46] on recommendation of web sites, “research has shown that if the task is intrinsic, i.e. just browsing for fun, it is very difficult to recommend sites”.

Each level of expectation leads to various attitudes from end users driven from the level of dissatisfaction, from deleting the application, to using it only for fun.

Handling changes to expectations over time: When using a recommender system, users’ expectations may change over time. This is important to consider with respect to the need to develop an adaptive recommender or not. The performance of most recommender systems evolves over time; with increases in user profile information comes better accuracy. Users too have expectations of a system’s capabilities at first use and these also change over time as both familiarity with the system increases and their own needs change. It is not only important that recommendation systems can adapt to the needs of users over time, but also that they can demonstrate system performance early enough in the use lifecycle to match the user’s expectations, building trust in the recommender output and ensuring continued user engagement.

10.3.2.3 Understanding users’ context

The final area to consider are contextual issues surrounding use of the recommender:

User’s device: The first consideration is what device will the user use to access the recommender? Recommenders are now a prevalent feature on applications and services accessible on devices as diverse as mobile handsets, desktop PCs, and

set top boxes. The implications of this on the system design are studied in section 10.3.1.

Situation of interaction: Situational considerations include firstly location, as the advent of ubiquitous mobile use has now made this a prime consideration. Location considerations may cover absolute geography such as if you wished to recommend attractions in the local area on a mobile device and also relate to understanding user activities beyond absolute positioning, e.g. contexts such as is the user at work, or are they shopping? Another important contextual consideration is temporal factors. It may be important for the recommender system to consider time information within user profile data collection when modelling the user. As an example, a user may watch news on mobile TV on the train into work, but prefers to watch comedy on the same train on the way home. Some solutions have been proposed to take into account these situations in the recommender model; for example with an explicit link between the interests and a given situation [12], or alternatively, the link between preferences and context information can be done implicitly [29]. The combination of location and temporal information in data collection can assist greatly in building a more robust recommender system [4].

Social environment: An important consideration here is whether the use of the system is usually carried out alone or with other people. This affects many design decisions including algorithm choice, data collection methods and recommendation presentation: e.g. this question helps decisions regarding strategies to provide group recommendations (see also Chapter 21), for example merging of individual preferences to obtain a unique group profile to be used in the content retrieval process [6, 10, 20], or the application of a consensus mechanism by users in order to cooperatively define a shared content retrieval policy [7, 32] or other methods described in [15]. However even in a group, people may still require individual recommendations, as explained for TV in [8]; for this example, some solutions have been proposed in [11, 30].

Consideration of all these questions requires considerable effort and raises the possibility for extensive user research. Though this may seem a very large activity, understanding which of these areas is relevant to the recommender under development can refine the amount of research needed.

10.3.3 Data Model

The last point the designer should study carefully are the characteristics of the items the system will exploit and manipulate. Indeed, item descriptions generally pre-exist before the personalised system, and the designer of the recommender system has little possibility to influence or change them. We propose a data model, helping to identify the main characteristics of data that may influence the design and the results of the future recommender. The designer shall implement our data model,

i.e. for each feature of the data model, they have to think about the possible values presented in Table 10.3.

Table 10.3: Data model.

Model's feature	Possible values
Data type	structured, semi-structured, unstructured
Metadata quality and quantity	high, medium, low
Metadata expressiveness	keyword-based, semantic-based
Description based on standards	yes, no
Volume of items	a lot, few
Diversity of items	homogeneous, heterogeneous
Distribution of items	long-tail, mainstream
Stability vs. persistence of items	stable, changing, changing a lot
User ratings	implicit, explicit, none
Type of rating	binary, multi-level...

10.3.3.1 Understanding the type of available data to describe items

There exist several ways to describe items:

- **unstructured data:** an item can be represented only by unstructured data, which refers to information that either does not have a data model or one that is not easily usable by a computer program (e.g. audio, video, unstructured text). In such cases, a pre-processing needs to be made to extract significant keywords, or concepts helping to distinguish each item from each other. The fact of having unstructured data is not blocking per se, because in many cases there are a number of techniques to obtain a set of metadata describing the items. For text analysis, tools such as GATE [19] or Lucene⁶ allows the extraction of keywords from unstructured text. Techniques for extraction of metadata from other types of data such as multimedia content (images, videos) are less reliable though, and often require to combine them together.
- **semi-structured data:** an item is often described by several generic metadata, corresponding to the main characteristics of the item and free text. A typical example of semi-structured data is an Amazon⁷ product page, or a TV programme, which is expressed with a number of properties such as programme genre, programme schedule, etc. Each property takes values in a finite set of vocabulary (that belong to a taxonomy or an ontology), or includes non-structured elements (e.g. the synopsis of the programme). In this case, the evaluation of the quantity and quality of data is required to know if the item metadata should be enhanced by performing some processing and analysis on the unstructured part.

⁶ Lucene, An Open Source Information Retrieval Library, <http://lucene.apache.org/>

⁷ Amazon, <http://www.amazon.com>

- **structured data:** items can also already be described by a well structured model that could belong to a standard, or a de facto standard, such as TVAnytime⁸ for the description of TV programs in an electronic program guide (EPG) for DVB-H. When having structured data, the designer must examine the quantity and quality of data to anticipate the potential lack of information for their use in the frame of a personalised system.

This first data feature has many impacts. First, on algorithms: if the data are unstructured and the cost to extract pertinent metadata is too high or too imprecise, a full set of recommender algorithm families are excluded: content-based approaches to recommendation [51] (cf. also Chapter 3), Bayesian model [21], rule-based system and distance metrics. Only collaborative filtering methods (cf. Chapter 4) could be used in such a configuration. Second, on user models: data representation should be reflected in the user model: for example, if a piece of text is represented as a vector of keywords, then content based method will lead to the use of a vector of keywords representation of user profile. In the case of unstructured data, the user model can be very simple, such as the history of the user's content consumption.

10.3.3.2 Understanding the quality / quantity of metadata

Quality and quantity of structured data are important performance factors of a recommender system, especially when using a content-based recommender. Several properties of metadata play a significant role in the choices that can be made with respect to the design of the personalised system.

Quality: the designer has to have some clues about the quality of the metadata available and has to understand the actions to take in the case where metadata are of medium or low quality. In general, an item metadata description is considered as of high quality if it enables one item to be distinguished from another. For example in news RSS feeds, two items can share common keywords corresponding to a category of news (e.g. sport news, football), but must have sufficient additional keywords for distinguishing news related to a certain match, a global event like a competition, a football player etc. In that example, the designer has to understand the right level of details in the description to capture the differences between news feeds. The quality feature of metadata in our model is a major point of decision for the recommender designer. He has to balance the accuracy of recommendation results and the recommender performance in terms of time to respond, storage capacity and processing cost. For example, if the designer prefers best performance of recommendation, he would have to introduce some constraints on the architecture and avoid implementing the recommender on a lightweight client. The designer can also choose to perform recommendations using a two step algorithm, distributed between the server and the client device [37]. This quality feature is also linked to the role of the recommender in the application and the expectation of users according to this role (see sec. 10.3.1 and 10.3.2).

⁸ TV Anytime, <http://www.tv-anytime.org/>

Expressiveness: the future users are also fundamental to the process of metadata evaluation. Metadata must reflect the users' points of view on items, and represent what users consider as differentiating characteristics of items. So the *expressiveness* of metadata is crucial to the performance of the recommender. Items can be expressed with a large variety of semantics: from no semantics using tags/keywords such as images annotated on Flickr⁹, to more advanced knowledge representations, such as taxonomies or ontologies (e.g. text annotations made by OpenCalais¹⁰). As a classic example, comparing a keyword-based approach and a semantic approach, an item referenced by specific keywords such as football, baseball, basketball, will not be recommended to a user interested in something more high level such as team sport. Through a semantic representation, the tacit knowledge that football, baseball and basketball are team sports is represented and can be used to extract pertinent user preferences and recommendations, and content about all type of sports teams would have been recommended because of the semantic link between the concepts. Metadata described with semantic concepts enable the use of more sophisticated recommender algorithms, such as [47] that takes into account the existence of "related" items according to their position in the ontology hierarchy; or spreading of semantic preferences (i.e. the extension of ontology-based user profiles through the semantic relations of the domain ontologies) as described in [48]. For collaborative filtering techniques, taking into account semantic similarities in the process has proven to increase accuracy and help reduce the sparsity problem [35]. However, there are some practical issues to be considered; for example, semantic reasoning induces some processing cost, that may not be feasible on the smallest devices. In addition, if the metadata are not sufficiently semantically richly described, some techniques enable the enrichment of the level of expressiveness of metadata, e.g. the matching of social tags with ontology concepts through Wikipedia¹¹[18], or the creation of folksonomies (cf. Chapter 19).

Quantity: The amount of metadata is an important factor to consider: too few metadata may lead to inaccurate recommendations, whereas too much metadata may lead to useless processing. In addition, metadata description may vary in terms of depth (degree of precision) and breadth (variety of description). The risk with superficial description is to propose items to users that do not correspond exactly to what they expect. For example, if a user is interested in news related to natural disasters: with an in-depth description of the content, the user will have recommendations about different types of disasters; with a breadth-wise description of the content, he will have different points of view about the disaster (political, sociological, economical, etc.). Very in-depth descriptions may reinforce some drawbacks of content-based filtering algorithms, in particular in terms of overspecialisation. The level of depth or breadth the user wants may be learned by the system.

⁹ Flickr, <http://www.flickr.com>

¹⁰ OpenCalais, <http://opencalais.com>

¹¹ Wikipedia, <http://www.wikipedia.org>

Description based on standard: Regardless of their level of expressiveness, the metadata can be described in different ways: using standards such as: Dublin Core¹², MPEG-7¹³, IPTC¹⁴; or using proprietary formats. If the metadata representation is not imposed on the designer of the personalisation application, the choice of metadata representation may be guided by several considerations such as the degree of integration between the recommender and other parts of the application (e.g. use in a retrieval engine).

10.3.3.3 Understanding the properties of the item set

Volume of items: in addition to the quantity of metadata per item, we should consider the volume of items in the data set. It represents an important factor in determining the choice of a recommender system family. Indeed, collaborative filtering algorithms require large datasets and /or large user sets in order to compute correlations efficiently. This is typically appropriate for books, films, etc, whereas content-based algorithms can cope with a smaller size of data set (e.g. TV programmes).

Distribution of items: It is also essential to consider how items are distributed among the data set. For example, if in a movie data set, a very high proportion of items is annotated with the concept “action”, this metadata may not be discriminative enough (too many content items selected if the user likes action films, too few if he does not). In such cases, and if possible, the level of depth of the annotation of data should be rethought in order to obtain better quality metadata.

Nature of items and stability of item set: News items, books, or TV programmes are intrinsically different, and therefore users do not behave the same way depending on the nature of the items. For example, it is relatively easy to conceive that interests related to TV programmes or books are more stable than the ones related to news, because news items change more frequently, and so there are more diverse subjects that can emerge. So, this criterion impacts the use or not of an adaptive recommender system (i.e. where the user profile evolves with time), and highlights the need to understand different evolution patterns, such as stable interests, progressive interests, fast changing interests, etc. [38, 39]. One should also consider the stability of the item set, i.e. how often new items are introduced or items disappear. In addition, a recommender can work with homogeneous data (e.g. movies only as done in MovieLens) or can manipulate heterogeneous content (as done in Amazon). In order to choose an appropriate strategy for item recommendations, it is important to study the correlation between the items, for example how an interest for a music type is reflected in book tastes. This analysis can help choose a strategy for recommendations, such as considering all items globally, or instead apply specific rules based on the specificity of each item family.

¹² Dublin Core Metadata Initiative, <http://dublincore.org/>

¹³ MPEG-7 overview, <http://www.chiariglione.org/mpeg/standards/mpeg-7>

¹⁴ IPTC (International Press Telecommunications Council), <http://iptc.org>

User items ratings: Another important consideration is that of taking into account user ratings or not. If there is no user rating related to items in the system, this excludes the whole family of collaborative filtering methods. User ratings about items can be either *explicit* (for example on Amazon, users can indicate how much they enjoyed a book or a CD), and may be expressed on different scales (e.g. binary, multi-level). If this is not directly available, but thought to be useful in the personalised application, it is possible to compute *implicit feedback indicators* [41] (for example, the fact of purchasing an item can be considered as an implicit indicator of user interest), which can be used either in some recommendation algorithms (such as collaborative filtering), or as an input to a user profile adaptation module that will update users' interests based on likes and dislikes of specific items.

10.3.4 A Method for Using Environment Models

In Tables 10.1,10.2,10.3, we introduced three models to help understand the environment of the future recommender system. For each model and each feature we proposed some guidelines to define requirements and constraints on the future recommender system. To fully understand the importance of those features we propose a method in two steps:

1. *identify the dependencies between features:* the designer must find which features are influencing others by building a dependency graph across the three models. This graph will help the designer understand how a change on one feature impacts the overall recommender environment. For example, changing the integration of recommendations with navigation features (application model) will change user expectations (user model).
2. *identify key features of the models:* key features are the ones that have the most important impact on the choice of the recommendation and adaptation algorithms, and recommender architecture. For example, the performance correctness (application model) has a significant impact on the recommender choice. The identification of these key features helps to prepare the evaluation framework and understand how to interpret evaluation results.

As an illustration of this method, an example is given in section 10.5.

Thus, in this section, we have identified all the constraints that should be studied when designing a recommender system. Based on this first step, the designer should be able to determine the appropriate algorithms for filtering the items, and for adapting if necessary the user profile, and can choose the right architecture. The next phase for the designer consists of implementing his recommender system (see our references) and then in evaluating the algorithms, as explained in the next section.

10.4 Understanding the Recommender Validation Steps in an Iterative Design Process

Even though the models presented above allow making informed decisions about the crucial elements of a recommender system, many parameters have to be adjusted before it can be deployed, making lab testing and iterative design necessary. Two aspects must be studied: validation of the algorithms themselves and validation of the recommendations, in which users must be kept in the loop.

10.4.1 Validation of the Algorithms

Many experimental papers have demonstrated the usefulness of testing a recommendation method against existing datasets such as MovieLens, Netflix¹⁵, etc. From experience, such datasets are useful to understand the behaviour of an algorithm; they must sometimes be transformed and enriched in order to be usable (e.g. enrich MovieLens with IMDB¹⁶ data). Experiments with datasets have been widely studied in the literature. For example, [25] gives a very exhaustive list of metrics that can be computed. In addition, they enable the support of objective comparison with other methods based on the characteristics and available information in the collections and may be particularly useful when large sets of users are needed. Such an evaluation also allows tweaking a number of personalisation parameters of the algorithm, e.g. distance metrics to compute the similarity between a user profile and a piece of content. These methods are widely explained in the literature and therefore are not further detailed here. Please refer to Chapter 8 for details.

In this particular step of testing the algorithm on available dataset, the feature dependencies and impact graph as determined by the method described in section 10.3.4, must help the designer discover if the dataset is changing the recommender environment (data, user, application) compared to the final targeted system. This graph should help interpret the results of the algorithms and determine how far tweaking of the algorithms and testing should go. Indeed, if the environment properties are really close to the ones of the targeted final system, it may be worth adjusting the algorithms as much as possible. But if significant differences have been analysed between the experimental environment and the targeted environment, this phase of experiments with existing datasets should probably be kept as short as possible and should mainly focus on debugging rather than improving the accuracy of the recommenders by small amounts. Many research papers have focused on improving the accuracy of the algorithms [33], but even if a recommender algorithm provides good accuracy results, this is still with respect to the instance of the data and user models considered and associated with the dataset. Therefore, it is very important to understand the importance of the features of the environmental models and their

¹⁵ Netflix dataset used for their competition, <http://www.netflixprize.com/download>

¹⁶ IMDB, the Internet Movie Database, <http://www.imdb.com/>

interdependencies to interpret correctly the evaluation results against a standardized dataset. For example, when we move from TV data to news data, the behaviour of the user is not the same, and the data do not have the same dynamics (see section 10.5). In such a case, spending too much time tweaking the algorithm to improve the accuracy on the TV dataset is certainly a waste of time.

10.4.2 Validation of the Recommendations

Whilst there are technical strategies for mathematically evaluating the accuracy of promoted items offered by a recommender, the acid test of ensuring measures of likely performance and satisfaction in the field are ultimately obtained through evaluating all areas of a recommender system *with end users*, with the focus upon making improvements to performance and increasing end user satisfaction.

The key to evaluating any interactive system is to follow a user-centred approach. This means evaluating *early, iteratively and frequently* based upon the changes and improvements made. This poses some challenges. Indeed as discussed earlier, recommenders are rarely at the centre of a development activity but instead are an integrated sub-feature of a much larger product or system. Common scenarios are that the development of the larger host system runs at a different development timescale to the recommendation engine. Such a situation poses problems for evaluation of the underlying recommender technology, leaving no method to collect user preferences upon which to build a user profile or a way to present recommendations to users.

These considerations often lead to the evaluation of the user-facing interaction aspects of the recommender (the data collection processes and the recommendation presentation) being split from the underlying engine, at least until both systems reach a prototype integration. From an evaluation perspective, this can sometimes make a lot of sense e.g. by preventing research findings from being confounded by competing negative or positive responses to other areas of the design.

There are many possible techniques in the user evaluation toolkit which can be deployed to investigate recommender developments. Some useful selective methods are introduced below together with when and in which context they can best be used. Each have been used in real life projects and are based upon our experiences of conducting user research in the area of recommender systems.

10.4.2.1 Card Sorting

Card sorting is not strictly an evaluation method but in fact a useful formative design research tool. It is used to create taxonomies of items based upon the naturalistic mental models users hold for the relationships between content or concepts. The method basically consists of asking users to sort a collection of cards, each which depicts a content item or sub classification, into groups based on similarity [44]. The resulting sorted groupings from a number of users can be analysed using cluster

analysis in order to find common shared patterns in user classification and to build a dendrogram of the associations. The time and complexity of this analysis can be reduced greatly by using one of the software tools such as IBM's EZsort [20].

We used this method when *investigating distance functions for television genre classifications*. Though many metadata structures exist in this area, it is very difficult without testing with users to understand if any one genre is perceived as being more similar to any single other genre more than others. As example, one result from the study showed that users classified a number of diverse genres as being close together due to a perception of them all as representing lighter viewing. This included genres such as game shows, comedy and soap operas. This was in contrast to more factual content genres classified as being less similar such as news and documentary. This information was integrated into the algorithm development to improve the accuracy of the recommendations.

The value of carrying out such an exercise is that the similarity measures for content items are based on a taxonomy created by real users, not artificial structures created by programmers or engineers. This method also unearths the subjective relative distances between disparate categories. These two factors increase the likelihood of recommendations being perceived as relevant and accurate from the perspective of the user. Due to its benefits in construction of the underlying algorithms, this tool is best used during formative development.

10.4.2.2 Low fidelity prototyping

Low fidelity prototyping is the umbrella term for a range of methodologies which cover exploration of an early interactive design idea with users. Traditionally this methodology has used paper prototyping to evaluate design ideas for how recommendations are to be presented, and also how user profile information may be captured and managed very early in the development of the recommender system. The major benefit of the method is that an evaluation can be achieved early without the need for costly and time consuming development. The method has no value however in evaluating the actual recommendation engine. It is executed in the form of a semi-structured interview using the prototype as a visual prompt. See [50] for a good introduction to the method and useful examples. When working with low fidelity prototypes it is important that they should not look like finished designs as this constrains users when discussing design ideas. Simple sketches and outlines allow users to openly postulate on highly valuable improvements and design ideas precisely because the designs appear so unfinished. This method is qualitative in nature as it attempts to gather rich data on the opinions of users. Evaluating a design with three or four users from each identified user group will often provide enough insight to significantly influence the design in a positive way.

We have used this type of investigation routinely to evaluate with users both actual design concepts but also general design principles. As an example, we used this methodology to introduce possible ideas for explicit rating scales in order to collect feedback which could be used to build user profiles. Simple line drawings

were enough to allow users to discern the differences between each concept and provide insights on their own perceived benefits and drawbacks for each. It allowed them to ponder more generally on the concept of providing explicit feedback to a recommender system, and what this meant in terms of both effort and privacy.

10.4.2.3 Subjective qualitative evaluation

This method is aimed at *evaluating the accuracy and satisfaction levels* attained by the recommendation engine from the perspective of end users. The major benefit of this method is *versatility*: it is a *qualitative* method which can be used as soon as a system is able to output recommendations even if the supporting data collection and recommendation presentation components do not yet exist. It can equally be used to evaluate fully implemented systems. The process is based on subjective assessment and a useful way of applying the method is through either a facilitator administered or participant completed standardised questionnaire¹⁷. Each recommendation generated for the user is presented within the questionnaire and a number of questions posed regarding the user's opinion towards it. The basic concept of the method consists of three steps.

1. Collection of users' data in order to build user profiles: the amount of data which needs to be collected is a function of the performance of the recommender. However it might equally be advisable to create participant samples within the study from whom varying amounts of information are captured to represent various typical usage patterns at given points in time, for example after two weeks. This allows the effect of the recommender's learning curve on user perceptions of accuracy and satisfaction to be analysed. The ease with which data can be collected from users is again a function of the maturity and design of the recommender. If the system is at a design stage where the preference data collection functions are operational then it may be possible to consider employing them within the study. This is especially true in systems where *implicit usage data* needs to be collected as it can be very difficult to elicit this type of data directly from users in other ways. Obviously, it is critical that the development team can access the actual user. Therefore the use of previously collected user data (for example in the form of web analytics) is unusable from the perspective of direct evaluation. Collecting implicit data from scratch requires the recruitment of participants for longer term monitored trials. This requires the use of techniques such as video observation or remote usage tracking in order to capture accurate preference data which can then be extracted and applied to implicit learning rules. This can be a laborious research activity but does also provide the added benefits of capturing real user insights and novel examples of use, which in turn drive innovation. In contrast, *explicit data collection* is far easier to simulate without a working system. Collecting feedback through an electronic or paper based survey form is a good method. An example we used was a simple spreadsheet. Users were asked to rate forty pieces of content on an explicit scale

¹⁷ Questionnaire development is a science so if the techniques of questionnaire design and attitude assessment are new to you a good practical reference guide is [36].

which was being proposed for the finished system. Fifty screened participants were recruited remotely via the internet and the survey sent via Email. The users simply filled in their responses and returned them to the design team.

2. Generation of recommendations: The preference data returned by participants is extracted from the survey responses and coded for input into the recommender engine. An important consideration in providing individualised recommendations is to keep track of the user ownership of data and feedback responses throughout the study. Firstly, obtain permission from the users to retain their information for the duration of the study. Give each participant an ID, and use that number on every document throughout the evaluation process. Securely retain the participant details for the duration of the study against which the participant ID can be resolved. Finally ensure that participants' sensitive personal details are destroyed after the study and the results of the study anonymised. This simple process maintains user privacy (and data protection).

3. Presentation of recommendations: once recommendations for each individual user have been generated then they need to be given back to participants for evaluation. The questions to ask participants in the questionnaire really depend upon the goals of the recommender. Three common user perspectives on the recommendations that are likely to be of interest are:

- Does the recommendation match their preferences?
- Is the recommendation of interest?
- Would they be satisfied in a system that delivered such a recommendation?

Importantly questions should primarily be phrased around specific recommended items, not the recommendation list as a whole, because they are less likely to provide insights that can be used to improve the recommender. Receiving information on individual items allows weaknesses and bugs in the recommender design to be identified. For a project related to TV recommendations that we worked on, it allowed investigation as to why a successful recommender was receiving small numbers of extremely poor anomalous outlier satisfaction ratings. Investigation of the outliers allowed the discovery of foreign language content recommendations not identified by the metadata. The design team made a fix to more intelligently identify and handle foreign language content of this type and solved the dissatisfaction issue.

Analysis of the data can allow in-depth pictures of likely user satisfaction. It can also allow direct comparisons to be drawn between competing systems which can be used either for benchmarking purposes or fed back into design direction decisions.

10.4.2.4 Diary studies

Once a recommender is out in the field, how can information be gathered as to the success of the development over the medium to longer term? Collecting usage metrics does not capture the subjective motivations, pleasure or frustration in using a system. An important consideration for recommenders is that such systems can have *long learning curves* with a *changing user experience over extended periods of use*. In such cases analysis of the total experience of living with a system may

be more interesting than snapshots of satisfaction. Diary studies typically can run anything from a week to months, dependant upon the system under investigation and the opportunities to access engaged users. Diary-based study as a method is particularly well suited to mobile contexts, when direct observation or monitoring becomes logistically difficult. We have recently used this method for a study related to the evaluation of the impact of different usage contexts upon video content selection on heterogeneous mobile devices where no remote user logging was possible. The method proved very useful at identifying particular pain points for users.

Diary is in fact a very well establish method in many areas of social research, (see [10] for examples). The method relies upon a user to create a self-reported record of their day-to-day interactions with the system of interest (it relates to real user contexts described in their own words). The insight from such data can be integrated to improve systems already in use to uncover user requirements and contextual use issues for the next generation of development. However, in terms of logistics the method does have possible pitfalls to be considered:

- *recruitment and retention of users*: it can often be difficult to recruit participants for longer duration studies, and even more difficult to retain them;
- *non-reporting or false reporting of information in the diary*: completing diary entries takes effort and engagement which can be difficult for users to sustain over the whole duration of a study.

In order to run a diary study successfully it is very important to maintain contact with participants. Regular face to face or telephone debriefs encourage users to maintain their diaries by instilling the expectations of the researchers and allows the investigator to monitor the data collected and query incidents or comments closer to the times that events are reported. Such strategies can be used to identify critical incidents in use which have led to episodes of user satisfaction or dissatisfaction.

Whilst by no means an exhaustive list, this range of methods has significant utility in the development process from the early design stages, through prototype development and finally to release and post release.

10.5 Use Case: a Semantic News Recommendation System

In the previous sections, we presented an approach to build a recommendation system, through the instantiation of three models: application, user, and data, which oblige one to think about possible choices for the recommender design. The purpose of this section is to compare what can be obtained from these models with what has been done in practice in the scope of a system - a News marketplace where news professionals or end-users can build, share (buy, sell) and consume multimedia content in a personalised way.

10.5.1 Context: MESH Project

We illustrate the usefulness of the models that may help in driving the design a recommender system through the description of MESH¹⁸ (*Multimedia sEmantic Syndication for enHanced news Services*), a research project in which we participated that created a framework for intelligent creation and delivery of semantically-enhanced multimedia news information [52].



Fig. 10.2: Overview of the MESH platform

The main purpose of MESH was to provide news professionals and end-users with powerful but intuitive means to organize, locate, access and present huge and heterogeneous amounts of multimedia information. A MESH system is a news content broker that guides the user through a multimedia content web (the “news mesh”), finds automatically what he/she needs or wants, and presents it in the best possible arrangement on any terminal. Core ideas in the system consisted of (see Fig. 10.2): (1) Content delivered (push model) to users based on semi-automatically extracted semantic metadata and users’ computed preferences; (2) Personalised multimedia summaries allowing easy digest of huge chunks of information, offering initial entry links to further access information through navigation aids that will prevent the “lost in multimedia cyberspace” syndrome; (3) Advanced syndication methods to allow rapid delivery of news from the end-sources (journalists) to the end-users (public) through fixed and mobile channels, proposing new flexible business and collaboration models.

In this context the role of the personalisation was to: (1) enable the system to proactively *push personalised news* items or personalised multimedia news summaries to user ; (2) provide support to users (both professional and end-users) to provide a *personalised search* of news items.

¹⁸ MESH IST project (FP6-027685) (03/2006 - 02/2009), <http://www.mesh-ip.eu>

10.5.2 Environmental Models in MESH

Based on the purpose of the application described above, we can illustrate how to instantiate the environmental models we described in section 10.3, in order to determine constraints on the recommender design.

10.5.2.1 Instantiation of the environmental models

Table 10.4: Application model instantiation.

Recommender purpose	<i>increase system efficiency</i> , both for professional users (save time when gathering news information for example to build a “dossier” on a specific theme) and for end users (receive or browse relevant news with respect to their profile, recent interests, current situation or recent queries)
Recommender type	<i>multiple items</i> , provides a list of recommended multimedia news documents
Integration with navigation features	<i>tight</i> : e.g. through coupling with the information retrieval to deliver a personalised search service
Performance criteria	different criteria have a primordial importance depending of the kind of users of the system: journalists and professional users are much more interested in characteristics such as <i>correctness</i> , <i>response speed</i> , <i>reliability</i> and <i>robustness to attacks</i> , whereas end users are more interested in <i>correctness</i> , <i>transparency</i> , <i>serendipity</i>
Device to support the application	mainly <i>fixed</i> devices (only a limited subset of the application is available on a mobile terminal)
Number of users	<i>single user</i> application
Application infrastructure	<i>browser-based</i> application, with two modes of content delivery: pull mode (personalised search) and push mode (proactive delivery of multimedia news summaries)
Screen real-estate	<i>not limited</i>

10.5.2.2 Links between the different models and constraints on design

In the MESH project, the study of the recommender system environment shows the important features of each model, how they influence each other, and which features have a direct impact on the choices about filtering/recommendation algorithms, infrastructure of the recommender and adaptation recommender option (see Fig. 10.3). This schema is essential in our method for analysing the key features of the system,

Table 10.5: User model instantiation.

Demographic information	only the <i>job</i> is considered as a discriminatory demographic factor: we distinguish between professional users such as journalists, photographers, etc. and end-users
Goal's existence and nature	in pull mode, explicit goals are expressed through user queries in the retrieval engine; in push mode : end users have no other goal than being informed about their favourite subjects and headlines news, Professionals still have the underlying goal of their current task that must be detected and inferred from their reading of items through the pull mode
Level of expectation	<i>medium</i> : the users can still use the system if the recommender does not work or perform poorly (though with a degraded efficiency); the recommender is not considered central to the user's activity
Change of expectation over time	<i>yes</i> : expectations of users should increase when they progressively discover the benefits of personalised search functions
Importance of user situation	<i>high</i> : for example, the users do not consume the same kind of news at home, at work or during holidays
Social environment	<i>alone</i> : by nature news consumption is rather an individual activity
Trust and privacy concerns	Not considered (assumption of a necessary trade-off between user benefits and divulgation of personal information and data)

Table 10.6: Data model instantiation.

Data type	<i>semi-structured</i> : news items are described through a number of categories and concepts; methods to extract metadata from unstructured parts of the items (text, speech, video) are applied
Quality of metadata	overall, <i>medium</i> : good expressiveness through semantic annotation of content, but many metadata extracted by semi-automatic means (less reliable)
Description based on standards	<i>yes</i> : use of ontologies ¹⁹ based on IPTC
Volume / diversity of items	huge number of news items on a large variety of topics
Distribution of items	long-tail and mainstream: wide distribution of concepts, from those about mainstream events such as Obama's election to those related to the discovery of new species of frogs
Stability and persistence of items	the news item set is not really persistent, as it constitutes a continuous stream, therefore the items taken into account for recommendations are changing (in general news older than x days are ignored)
User ratings	<i>explicit</i> ratings (such as star rating) as well as <i>implicit</i> ratings (by monitoring clicks and time spent on news items) are considered

and helps in determining that when a change occurs in the instantiation of one of the models, what the impact is on system constraints and requirements.

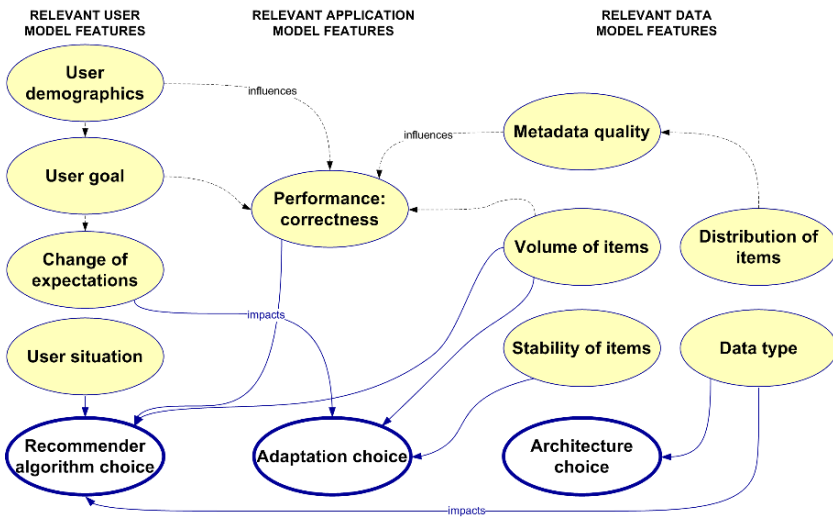


Fig. 10.3: Dependencies and impacts of model features in MESH

Impact on recommendation algorithms: In MESH, the primary purpose of the recommender is to improve efficiency of the system; therefore the selected algorithm should overcome limitations of content-based recommenders and of collaborative filtering [7, 14, 1]. The use of hybrid algorithms [14] can overcome these limitations. In addition, several categories of users cohabit, who may have different goals and interact with the system according to two different modes (push/pull). In pull mode, because the recommendations are driven by user queries, it may be preferable to use a content-based solution, whereas in push mode, for end-users, it may be interesting to have more diversity brought by collaborative filtering methods. Therefore, it appears that hybrid solutions, combining content-based approaches and collaborative filtering may be more appropriate than one single approach. Another argument is related to the metadata: although there are metadata (which makes possible the use of content-based methods), their quality is still medium, which can lead to average quality recommendations, regardless of the intrinsic quality of the recommendation algorithms. Therefore it is safer not to rely solely on content-based methods.

Once it is decided that it may be worth combining different kinds of recommenders, the designer should analyze the environmental elements that help choose more precise methods within these families. Thus:

- for the collaborative filtering (CF) method: the study of the data shows that it is quite unlikely to apply a classical item-based CF [45], because due to the high volume and the dynamic nature of the data (news items arrive every few min-

utes), it is quite unlikely to have a lot of user ratings (either implicit or explicit) on specific news items. This particularly big sparsity requires the designer to imagine other ways of doing collaborative filtering, e.g. exploit user profile information when calculating user similarities in user-based CF [15].

- for the content-based algorithm: the highly structured metadata (expressed through ontological concepts) allow the use of more sophisticated recommenders that can exploit this structure to enhance the accuracy of recommendations [17].

Lastly, another important point is to determine how to combine these algorithms. Based on the environmental study, several heuristics are possible: static or dynamic combinations based on user type (end-user vs. professional) or based on the application mode (pull vs. personalised push). The user's context may also be used to parameterize the different algorithms and their combinations.

Additionally, in the context of news, it is essential to consider that there is a lot of similar content (such as news about Obama's election but coming from different sources); therefore some additional mechanisms to avoid recommending duplicated or equivalent content may be needed - for example clustering methods. In that case, it is important to identify and integrate into the overall recommendation process the criteria that users usually apply to distinguish between two similar news items.

Decision of building an adaptive recommender: the study of news data shows that news items change often (new news items are arriving fast) and also topics change often. Because of these properties (high variation and high coverage), user studies we have conducted have proven that users are not able to express exhaustively their interests relating to news, and that therefore a user profile may vary a lot. In addition, in the system, users have the opportunity to freely browse through specific news they are interested in. This information about possible new topics of interests for the users must be taken into account into their profile to maintain them up-to-date, so that it can be used also to deliver truly relevant personalised news in push mode. So, this makes it necessary to have an adaptive recommender, where user profiles evolve automatically and rapidly over time[38].

Impact on architecture: from the beginning, as MESH was a new application, we had two options for the recommender system: 1) a distributed solution or 2) a completely centralised solution. A distributed solution had some advantages: better management of privacy, and application independence, but some serious drawbacks: in the case of use of the application from different devices, it required profile synchronization between devices, data transfer between the devices and therefore back-end costs could be high. Therefore, it was decided to favour a centralized system, which eases the integration of the recommender with the other parts of the system such as content retrieval or content syndication.

10.5.3 In Practice: Iterative Instantiations of Models

As the recommender system was built in parallel to other modules of the system, it was not possible to wait until we had the real data to start building the recommender. So, it was decided to build algorithms and evaluate them with publicly available datasets:

- for the recommender algorithms: all tests were done with MovieLens + IMDB;
- for the profile adaptation algorithms, BARB²⁰ data. The choice of this dataset was driven by the need to have temporal evolution of content consumption.

The results showed that the hybrid recommendation methods developed in MESH provided more accurate results than state of the art methods (content-based or collaborative filtering). In this case, the quality of metadata was close to what we could expect from MESH and the user context (situation and goal) was not taken into account. However, in the experiments we carried out with the BARB dataset, which collects data related to TV programme consumption during a 6 month period, we unconsciously created a new data model and a slightly different user model:

- Data model: items were TV programmes described by limited metadata, that were high level categories of programmes and a set of keywords extracted from the programme description text. The item distribution was mainstream with a poor to medium quality of annotation (there is significantly less diversity in TV programmes than in news). The stability of items was also different since TV programmes were predefined for a long period and we can consider them as quite stable (TV is fairly repetitive, and therefore new programmes do not appear every days). Two features influencing the performance of the recommender were changed by this dataset: *metadata quality* and *distribution of items* and one feature influencing the adaptation results was changed: *stability of items*.
- For the user model, professionals were not represented in the BARB dataset, so no specific goal was linked to the recommendation. It had an impact mainly on the correctness of the recommendation.

The BARB dataset was supposed to help us in evaluating our learning algorithm for user preferences but the results showed that after one week of learning, the user profiles remain stable, meaning that the preference learning mechanism had no further effect on the user preferences [38]. We had to interpret our results not only according to the behaviour of the algorithm by itself, but to the behaviour of the algorithm and the characteristics of the chosen dataset. The algorithm was supposed to deal with a large volume of items, not persistent and with mainstream to long-tail distributions with a good quality of metadata. It was supposed to be very reactive and built based on an average consumption of items that was largely superior to the TV programmes consumption per day. We learnt through this experience that changing the recommender environment (data, user and application models) has a huge implication on the expectation of the result and in the interpretation of the filtering algorithm and other mechanisms linked to the recommender system.

²⁰ BARB - Broadcasters' Audience Research Board, <http://www.barb.co.uk>

Therefore, we tried to use data as close as possible to the ones being used within MESH. We launched new experiments with a tool called “News@Hand”[16]. News@Hand combines textual features and collaborative information to make news suggestions to an end-user. News items are automatically and periodically retrieved from several on-line news services via RSS feeds. The title, summary and category of the retrieved news items are annotated with concepts from the system domain ontologies. During a period of 3 weeks we performed subjective qualitative evaluations (see section 10.4.2.3) to analyse the evolution of the preference learning algorithm with twenty users (they were divided into sub-groups in order to compare the evolution in different configurations of the algorithm). As the data model for this experiment was equivalent to the one in the MESH project and the user model just slightly different (no goal), we obtained results that were closer to our expectations (increase of recommendations quality over time).

By carefully studying the recommender environment through the instantiation of the data, user and application models and by identifying dependencies and impacts of model’s features, we think that critical issues must be addressed at design time in order to avoid errors and avoid losing time in dead ends.

10.6 Conclusion

In this chapter, we showed that though the technologies available to those who want to implement a recommender system are numerous, diverse and often hard to compare, a systematic evaluation of the environment is a key to making appropriate choices. We proposed for this to build three models describing the features of the environment that are the most influential to the recommender system design: application, user and data. Building these three models and studying their interaction allows narrowing the choice of appropriate recommendation algorithms, system architectures and user models.

In practice (as shown by the example we described), these three models by themselves will not allow to have a perfect design at the first go. But they are a very useful support for an iterative design methodology, which is the best way to go beyond technical excellence and reach the goal that matters in the end: user satisfaction.

References

1. Adomavicius, G., Tuzhilin, E.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749 (2005)
2. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, vol. 22, pp. 207–216. ACM (1993)

3. Ali, K., van Stam, W.: Tivo: making show recommendations using a distributed collaborative filtering architecture. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. Seattle, WA, USA (2004)
4. Anand, S.S., Mobasher, B.: Contextual recommendation. In: From Web to Social Web: Discovering and Deploying User and Content Profiles, Lecture Notes in Computer Science, pp. 142–160. Springer-Verlag (2007)
5. Anderson, C.: The long tail. *Wired* (2004)
6. Ardissono, L., Goy, A., Petrone, G., Segnan, M., Torasso, P.: Intrigue: Personalized recommendation of tourist attractions for desktop and handset devices. In: Applied Artificial Intelligence, pp. 687–714. Taylor and Francis (2003)
7. Balabanović, M., Shoham, Y.: Fab: content-based, collaborative recommendation. *Communications of the ACM* **40**(3), 66–72 (1997)
8. Bernhaupt, R., Wilfänger, D., Weiss, A., Tscheligi, M.: An ethnographic study on recommendations in the living room: Implications for the design of itv recommender systems. In: EUROITV'08: Proceedings of the 6th European conference on Changing Television Environments, pp. 92–101. Springer-Verlag (2008)
9. Bias, R., Mayhew, D.: Cost-Justifying usability. Morgan Kaufman Publishing (1994)
10. Bolger, N., Davis, A., Rafaeli, E.: Diary methods: Capturing life as it is lived. In *Annual Review of Psychology* **54**(1), 579–616 (2003)
11. Bonnefoy, D., Bouzid, M., Lhuillier, N., Mercer, K.: More like this or not for me: Delivering personalised recommendations in multi-user environments. In: UM'07: Proceedings of the 11th international conference on User Modeling, pp. 87–96. Springer-Verlag (2007)
12. Bonnefoy, D., Drgelhorn, O., Kernchen, R.: Enabling Technologies for Mobile Services: The Mobilife Book, chap. Multimodality and Personalisation. John Wiley & Sons Ltd (2007)
13. Bonnefoy, D., Picault, J., Bouzid, M.: Distributed user profile. Patent applications EP1934901, GB2430281 & WO2007037870 (2005)
14. Burke, R.: Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction* **12**(4), 331–370 (2002)
15. Cantador, I.: Exploiting the conceptual space in hybrid recommender systems: a semantic-based approach. Ph.D. thesis, Universidad Autónoma de Madrid (UAM), Spain (2008)
16. Cantador, I., Bellogn, A., Castells, P.: News@hand: A semantic web approach to recommending news. In: Proceedings of the 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2008), *Lecture Notes in Computer Science*, vol. 5149, pp. 279–283. Springer-Verlag (2008)
17. Cantador, I., Fernández, M., Vallet, D., Castells, P., Picault, J., Ribière, M.: Advances in Semantic Media Adaptation and Personalization, *Studies in Computational Intelligence*, vol. 93, chap. A Multi-Purpose Ontology-Based Approach for Personalised Content Filtering and Retrieval, pp. 25–51. Springer (2008)
18. Cantador, I., Szomszor, M., Alani, H., Fernández, M., Castells, P.: Enriching ontological user profiles with tagging history for multi-domain recommendations. In: Proceedings of the 1st International Workshop on Collective Semantics: Collective Intelligence and the Semantic Web (CISWeb 2008), pp. 5–19 (2008)
19. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: Gate: A framework and graphical development environment for robust nlp tools and applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02) (2002)
20. Dong, J., Martin, S., Waldo, P.: A conference on human factors in computing systemser input and analysis tool for information architecture. In: Conference on Human Factors in Computing Systems, pp. 23–24 (2001)
21. Duda, R.O., Hart, P., Stork, D.G.: Pattern Classification. John Wiley, New York (2001)
22. Gilb, T.: Principles of software engineering management. Arron Marcus Associates (1988)
23. Ha, S.H.: An intelligent system for personalized advertising on the internet. LNCS 3182 pp. 21–30 (2004)

24. Herlocker, J.L., Konstan, J.A., Riedl, J.: Explaining collaborative filtering recommendations. In: CSCW'00: Proceedings of the 2000 ACM conference on Computer supported cooperative work, pp. 241–250. ACM, New York, NY, USA (2000)
25. Herlocker, J.L., Terveen, L.G., Konstan, J.A., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Transactions on information systems* **22**, 5–53 (2004)
26. International Organisation for Standardisation (ISO): ISO 13407: Human centred design processes for interactive systems.
27. Jameson, A., Baldes, S., Kleinbauer, T.: Two methods for enhancing mutual awareness in a group recommender system. In: AVI'04: Proceedings of the working conference on Advanced visual interfaces, pp. 447–449. ACM, New York, NY, USA (2004)
28. Lhuillier, N., Bonnefoy, D., Bouzid, M., Millerat, J., Picault, J., Rib iere, M.: A recommendation system and method of operation therefor. Patent application WO2008073595 (2006)
29. Lhuillier, N., Bouzid, M., Gadanho, S.: Context-sensitive user preference prediction. Patent application GB2442024 (2006)
30. Lhuillier, N., Bouzid, M., Mercer, K., Picault, J.: System for content item recommendation. Patent application GB2438645 (2006)
31. Masthoff, J.: Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Modeling and User-Adapted Interaction* **14**(1), 37–85 (2004)
32. McCarthy, K., Salam o, M., Coyle, L., McGinty, L., Smyth, B., Nixon, P.: Cats: A synchronous approach to collaborative group recommendation. In: G. Sutcliffe, R. Goebel (eds.) Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, pp. 86–91. AAAI Press (2006)
33. McNee, S.M., Riedl, J., Konstan, J.A.: Accurate is not always good: How accuracy metrics have hurt recommender systems. In: CHI'06 extended abstracts on Human factors in computing systems, pp. 1097–1101. ACM, New York, NY, USA (2006)
34. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Technol.* **7**(4) (2007)
35. Mobasher, B., Jin, X., Zhou, Y.: Semantically enhanced collaborative filtering on the web. *Web Mining: From Web to Semantic Web* pp. 57–76 (2004)
36. Oppenheim, A.: Questionnaire Design, Interviewing and Attitude Measurement. Continuum International Publishing, New York (2001)
37. Papadogiorgaki, M., Papastathis, V., Nidelkou, E., Kompatsiaris, Y., Waddington, S., Bratu, B., Rib iere, M.: Distributed user modeling for personalized news delivery in mobile devices. In: 2nd International Workshop on Semantic Media Adaptation and Personalization (2007)
38. Picault, J., Rib iere, M.: An empirical user profile adaptation mechanism that reacts to shifts of interests. Submitted to the 18th European Conference on Artificial Intelligence (2008). <http://www.mesh-ip.eu/upload/ecai2008.pdf>
39. Picault, J., Rib iere, M.: Method of adapting a user profile including user preferences and communication device. European Patent EP08290033 (2008)
40. Rib iere, M., Picault, J.: Progressive display of user interests. Tech. rep., Motorola (2008). <https://priorart.ip.com/download/IPCOM000167391D/>
41. Rib iere, M., Picault, J.: Method and apparatus for modifying a user preference profile. Patent application WO2009064585 (2009)
42. Rich, E.M.: User modeling via stereotypes. *Cognitive Science* **3**, 329–354 (1979)
43. Rokach, L. and Maimon, O. and Arbel, R., Selective voting-getting more for less in sensor fusion, *International Journal of Pattern Recognition and Artificial Intelligence* **20**(3):329–350 (2006)
44. Rugg, G., McGeorge, P.: The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems, The journal of Knowledge Engineering* **14**(2), 80–93 (2002)
45. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: WWW'01: Proceedings of the 10th international conference on World Wide Web, pp. 285–295. ACM, New York, NY, USA (2001)

46. Shepherd, M.: Tutorial on personalization and filtering on the web. Web Information Filtering Lab, Dalhousie University, Canada. <http://ncsi-net.ncsi.iisc.ernet.in/gsd/collect/icco/index/assoc/HASH011b.dir/Tutorial-Shepherd.ppt>
47. Shoal, P., Maidel, V., Shapira, B.: An ontology-content-based filtering method. *International Journal of Information Theories and Applications* (15), 303–318 (2008)
48. Sieg, A., Mobasher, B., Burke, R.: Ontological user profiles for personalized web search. In: *Proceedings of AAAI 2007 Workshop on Intelligent Techniques for Web Personalization*, pp. 84–91. Vancouver, BC, Canada (2007)
49. Signa, R.: Design strategies for recommender systems. UIE Web App Summit. <http://www.slideshare.net/rashmi/design-of-recommender-systems>
50. Snyder, C.: *Paper prototyping: The fast and east way to design and refine user interfaces*. Morgan Kaufmann Publishing, San Francisco (2003)
51. Terveen, L., Hill, W.: Human-Computer Interaction in the New Millennium, chap. Beyond Recommender Systems: Helping People Help Each Other, pp. 487–509
52. Villegas, P., Sarris, N., Picault, J., Kompatsiaris, I.: Creating a mesh of multimedia news feeds. In: *European Workshop on the Integration of Knowledge, Semantics and Digital Media Technologies (EWIMT)*, pp. 453–454. IEE (2005)
53. Yu, Z., Zhou, X., Hao, Y., Gu, J.: Tv program recommendation for multiple viewers based on user profile merging. *User Modeling and User-Adapted Interaction* **16**(1), 63–82 (2006)
54. Zhang, M., Hurley, N.: Avoiding monotony: improving the diversity of recommendation lists. In: *RecSys'08: Proceedings of the 2008 ACM conference on Recommender systems*, pp. 123–130. ACM, New York, NY, USA (2008)
55. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: *WWW'05: Proceedings of the 14th international conference on World Wide Web*, pp. 22–32. ACM, New York (2005)

Chapter 11

Matching Recommendation Technologies and Domains

Robin Burke and Maryam Ramezani

Abstract Recommender systems form an extremely diverse body of technologies and approaches. The chapter aims to assist researchers and developers to identify the recommendation technologies that are most likely to be applicable to different domains of recommendation. Unlike other taxonomies of recommender systems, our approach is centered on the question of knowledge: what knowledge does a recommender system need in order to function, and where does that knowledge come from? Different recommendation domains (books vs condominiums, for example) provide different opportunities for the gathering and application of knowledge. These considerations give rise to a mapping between domain characteristics and recommendation technologies.

11.1 Introduction

Unlike some other types of software systems, recommender systems are not defined by a particular kind of computation, like for example, a statistical computation package, or by the storage and use of a particular kind of data, as in a geographical information system. A recommender system is defined by a particular kind of semantics of interaction with the user: “any system that produces individualized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options” [11]. This expansive definition makes the scope of recommender systems research quite broad, but it fails to give much guidance to the implementer. A crucial question is there-

Robin Burke

Center for Web Intelligence, College of Computing and Digital Media, 243 S. Wabash Ave., DePaul University, Chicago, Illinois, USA e-mail: rburke@cs.depaul.edu

Maryam Ramezani

Center for Web Intelligence, College of Computing and Digital Media, 243 S. Wabash Ave., DePaul University, Chicago, Illinois, USA e-mail: mramezani@depaul.edu

fore how recommendation techniques can be matched to recommendation problems. That is the question that this chapter tries to address.

11.2 Related Work

There are several taxonomies for recommender systems. Burke [11] distinguishes between five different recommendation techniques: collaborative, content-based, utility-based, demographic, and knowledge-based. The article discusses the advantages and disadvantages of each technique and proposes hybrid recommender systems to gain better performance with fewer of the drawbacks of any technique in isolation. An early work in recommender systems [54], which focuses on collaborative recommenders, identifies 5 dimensions to place the systems in a technical design space. The dimensions characterize properties of the users' interactions with the recommender and the aggregation methods of users' evaluations (ratings). Konstan and Schafer [61] present a taxonomy of collaborative e-commerce recommender applications that separates their attributes into three categories: the functional I/O, the recommendation method, and other design issues such as degree of personalization and delivery methods. An eight-dimensional taxonomy of recommender systems is presented in [46] using two main criteria: user profile generation and maintenance, and user profile exploitation techniques. In this taxonomy, common patterns in recommender systems are extracted by an analysis of the systems in the same domain. A more recent survey on recommender systems [3] classifies recommendation methods (omitting knowledge-based) into three main categories: content-based, collaborative, and hybrid recommendation approaches and classifies recommenders in each category into either heuristic-based or model-based.

This work is distinguished from previous categorizations in that it is not aimed at classifying existing recommender systems along particular dimensions of interest as in the surveys above, but rather as an AI-centric approach, focused on the knowledge sources required for recommendation and the constraints related to them. The chapter discusses the applicability of different recommendation techniques to different types of problems and aims to guide decision making in choosing among these techniques. As such, it might be considered to serve as a sort of recommender for recommender system implementers.

11.3 Knowledge Sources

A fundamental choice for an implementer of an AI system is the source and type of knowledge that the system will employ. In the case of recommender systems, there are two primary entities about which we might have knowledge: because recommendation is personalized, a recommender must have knowledge of its users; the recommender may also have knowledge about the features of the items that it

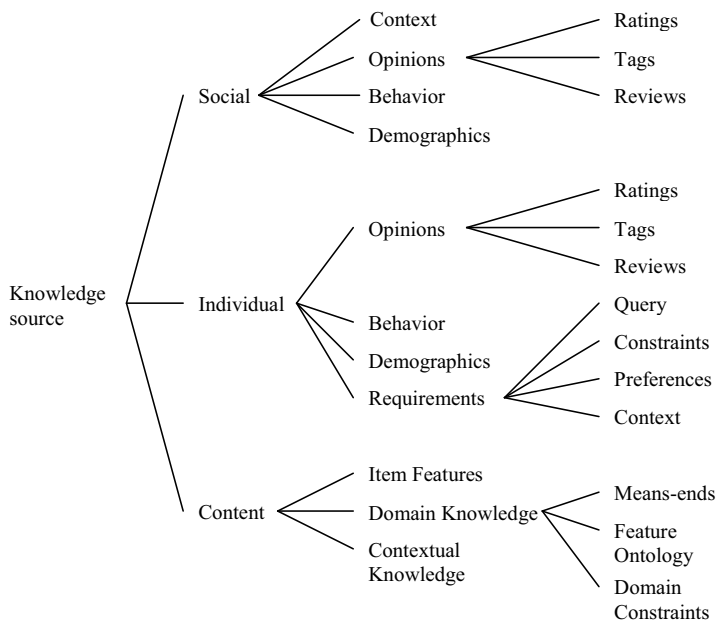


Fig. 11.1: Taxonomy of knowledge sources in recommendation

is recommending. It is considered one of the chief benefits of collaborative recommendation that domain knowledge or item features are not required, but all other recommendation techniques require them.

For an individual instance of recommendation, we are presented with a particular target user and seek to make personalized recommendations for him or her. In this situation, we can divide the knowledge of users into what we know about the target user, and what knowledge we have of the user community at large. There are therefore three broad categories of knowledge that may come into play in recommendation:

- **Social:** Knowledge about the larger community of users other than the target user.
- **Individual:** Knowledge about the target user.
- **Content:** Knowledge about the items being recommended and, more generally, about their uses.

Felfernig and Burke present a taxonomy of recommendation knowledge in [18]. Figure 11.1 shows this taxonomy, further expanding each category into subtypes of knowledge, all of which have been used in some existing recommender systems. These subtypes are explained below.

Social knowledge is the total sum of all of the user profiles stored in a system. Collaborative recommendation is intensive in its application of social knowledge,

usually with profiles of the simplest type: user opinions such as the liked-disliked scales used in MovieLens and other well-known collaborative recommenders, or interaction histories as seen in collaborative web personalization. Other types of knowledge can come into play, however. In the I-SPY collaborative web search application, user's queries (requirements) are recorded as well as their preferences (link selections) relative to those queries [63]. Demographic information about the user base is also employed by some recommender systems.

Individual knowledge is what drives a given interaction with the recommender system. It may be relatively implicit, in the sense of a user's profile being recalled to memory and used to initiate processing of recommendations, or it may be explicit in that the user specifies his or her interest before or during the recommendation process. In addition to the behavioral, opinion and demographic types of knowledge described relative to the social category, individual requirements are much more commonly found. For example, in the class of knowledge-based recommender systems known as critiquing systems, a user examines recommended items and responds with multi-dimensional critiques that act as constraints and/or preferences on the next round of retrieval [12]. Such systems often begin their interaction with a query that the user formulates.

Content knowledge has a variety of forms. In its simplest incarnation, the system might only have knowledge about the features of items that it is recommending, enabling it to learn what features a user seems to prefer. Domain knowledge refers to more complex notions of content knowledge such as means-ends knowledge, that is what means/features are appropriate for which goals/ends that the user might have in mind. A feature ontology relates features to each other so that similarity and difference between items can be more adequately assessed. Domain constraints may be necessary to prevent a system from recommending an item that is inconsistent with what the domain permits.

Context is an important factor in many application domains. What makes for an appropriate recommendation will often be the function of the context – a search for a restaurant to conclude a high-powered business transaction will be different from a search aimed at finding a place to celebrate a four-year-old's birthday even if the searcher's profile, and her query, "Italian" are the same. The use of context in different types of recommender algorithms including collaborative recommendation is an area of active research [2, 53, 31, 5, 62, 70]. However, there is no consensus on how best to profit from it or even how to define the term. Contextual issues in recommendation will receive limited treatment in this chapter.

11.3.1 Recommendation types

Recommendation types are explained in detail elsewhere in this volume. However, in conjunction with a discussion of knowledge sources, it is worth considering how different recommendation types operate.

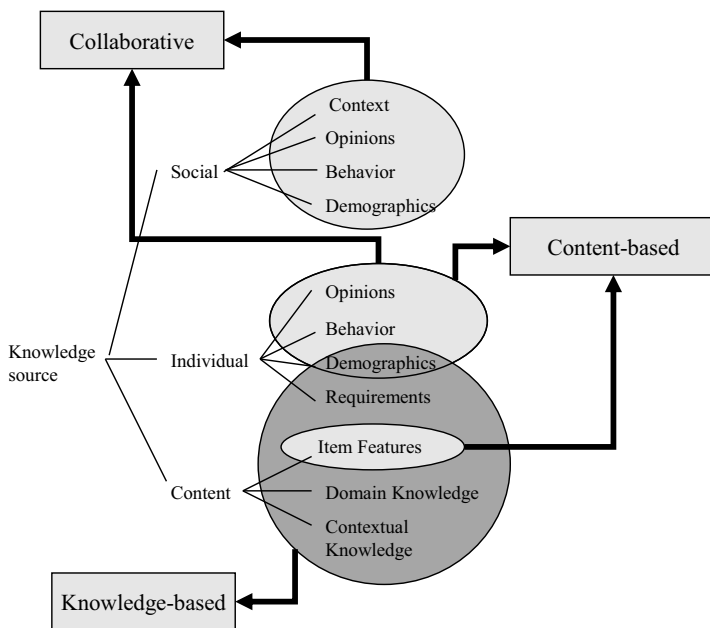


Fig. 11.2: Knowledge sources and recommendation types

- Collaborative recommendation matches an individual knowledge source with a social knowledge source of the same type and extrapolates the target user’s preferences from his or her peers. Usually, in collaborative recommendation, individual requirements are not used, or applied very simply as filters.
- Content-based recommendation on the other hand is individually-focused, using item features and user opinions to train a classifier that can predict user preferences on new items.
- Knowledge-based recommendation is more of a catch-all category in which the recommender applies any kind of domain knowledge more substantive than item features.

Figure 11.2 shows the connection between knowledge sources and the recommendation types.

From a knowledge source perspective, hybrid recommendation is really a matter of combining knowledge sources that have not traditionally been put together in the three types discussed above. Often, a hybrid is created by adapting an algorithm for one recommendation type to accept a knowledge source more typically associated with another type.

Of course, a knowledge source alone does not make a recommender system. An AI system also needs algorithms. It is difficult to generalize since new recommendation algorithms are put forward with great regularity, but in general, col-

laborative systems use multi-class classification algorithms for extremely sparse and high-dimensional spaces; content-based systems use binary learning algorithms for lower-dimensional spaces; and knowledge-based recommenders use inference schemes of various types.

Still, an algorithm can only function with the right knowledge sources, and it is the main topic of this chapter. Considerations about the domain of application and the style of interaction with the user lead us to conclusions about the availability and characteristics of different knowledge sources. These considerations in turn can be used to guide the selection of feasible recommendation algorithms. We turn next to the characteristics of domains.

11.4 Domain

A domain of recommendation is the set of items that the recommender will operate over, but may also include the set of aims or purposes that the recommender is intended to support. A specialized recommender, for example, a news recommender that identifies stories for the attention of government intelligence analysts, may have different implementation considerations than a generalized news recommender such as Google News. In turn the characteristics of the domain affect the availability and utility of different knowledge sources. In the online news case, there are a huge number of news sources and articles such that no user will ever have time to experience or rate more than a small fraction of them. In addition, the news itself is undergoing constant change. So, we can characterize the "Social / Opinions / Ratings" knowledge source as one of great sparsity and great dynamism.

Another aspect of the domain has to do with the larger application in which the recommender is embedded. If the recommender is a part of a larger system like an e-commerce site, it may be necessary for the recommender to impose as little as possible on the normal user interaction with the application. This may limit the degree to which detailed requirements can be collected. On the other hand, if the recommender is the primary application that users are accessing, it can gather data explicitly from users.

We have identified six important characteristics of the domain that an implementer should consider: heterogeneity, risk, churn, interaction style, preference stability, and scrutability.

11.4.1 Heterogeneity

A heterogeneous item space encompasses many items with different characteristics and most importantly, different goals they can satisfy. For example, an e-commerce recommender system as found at Amazon.com has a large number of heterogeneous items that can be recommended. Even within a single category like books, such

disparate categories as home repair, romance novels, cooking, and children's fantasy all coexist in the database.

A homogeneous recommendation space means that content knowledge relative to the domain will be easier to acquire and maintain. Consider a site that only recommends digital cameras versus one that has all kinds of electronics. The camera-only site would be able to invest in content knowledge specific to photography, whereas the general site would have a much more challenging task trying to do knowledge engineering for all of consumer electronics. Even a simple catalog of item features becomes difficult to design effectively if the items differ wildly from each other.

11.4.2 Risk

Recommendation domains can be distinguished by the degree of risk that a user incurs in accepting a recommendation. A 99 ¢ music track is low risk; a \$1.5 million condominium or a medical diagnosis could be very high risk. Risk determines the user's tolerance for false positives among the recommendations. In some domains, false negatives may also be important – if there is a cost or risk associated with not considering some options.

Another way to think of a high-risk domain is that there are likely to be some important constraints on a valid solution that the recommender system must obey. For example, a condominium buyer is likely to have some very strong constraints about location, price and amenities. As mentioned above, the tolerance for false positives is going to be low for high-risk items.

11.4.3 Churn

Recommender systems are used in domains with long-lived items like books, but they are also used in domains where the value or relevance of an item has a very short time span, such as news stories. A high churn domain is one in which items come and go rapidly.

In such a domain, a recommender system faces a continual stream of new items to be integrated into its knowledge sources. This greatly increases the sparsity of any kind of opinion data, as new items will necessarily have been seen by very few users. Items that have been around for some time may accumulate ratings, but by the time they do, they may no longer be relevant.

11.4.4 Interaction Style

In systems in which the user makes no special effort to interact with the recommender system, the system extracts the implicit expressed preferences from user behaviour. For example, when visiting a web site, a user leaves behind behavioral traces in web server logs that can be used to make recommendations. Implicit inputs may include the specific items that the user is currently viewing, user transaction history, elapsed time, and shopping cart / purchasing behavior. Explicit inputs require that the user make the effort to formulate an opinion or a query or to add personal data to the system. There must be some perceived benefit for doing so that justifies the effort.

Implicit inputs are naturally noisy because they are inferred from user behavior. This type of interaction may be best suited for gathering simple rating knowledge, although some researchers have explored the extraction of preferences and even domain knowledge from implicit data [58, 72]. Explicit inputs may be more sparse if the burden of generating them causes users to do so relatively rarely.

11.4.5 Preference stability

User preferences can also have varying degrees of duration. For example, a person buying a digital camera would typically switch preferences after purchase, since they would be no longer interested once the purchase was complete, while a person interested in comedy movies may wish to continue getting comedy recommendations for a long period of time. Also, preferences for some items may increase and wane naturally, for example, when one's favorite basketball team is in a big tournament, stories about it become highly preferred, but if they are knocked out or when the tournament is over, the user's preferences will change.

Stable preferences mean that opinion data collected in the past is still likely to be valid today. This makes it easier to maintain high-quality knowledge sources that use this data. Unstable preferences mean that any data collected in the past may have to be discounted or discarded. This increases the importance of gathering the user's specific requirements of the moment. If the user generates a large amount of implicit data in a single session, then it may be possible to use individual sessions as profile data.

The problem of preference instability can be ameliorated by collecting more data. If a user generates enough opinion data during a single session to adequately represent his or her current preferences, then there is no need to extrapolate from historical data and the issue of preference stability does not arise. This situation is found in web personalization applications. Users generate a large number of clicks while browsing a web site, enough implicit data to allow for recommendation using only the data from a single session.

11.4.6 Scrutability

Certain applications (for example, high-risk ones) may require that the system be able to explain its recommendations, to answer questions like “why was this item recommended?” Such explanations enhance user confidence that a recommendation is appropriate [42] and increase the likelihood of recommendations being accepted.

Explaining recommendations is most straightforward when content knowledge sources are employed [41, 57]. Explaining a recommendation based on social knowledge has proved more challenging. See Herlocker et al. [26] for an evaluation of some alternatives.

11.5 Knowledge Sources

As we have shown above, the choice of domain and the characteristics of the application place certain constraints on the kinds of knowledge sources that a recommender system may deploy. In turn, the availability and quality of knowledge sources influences what recommendation technologies a recommender can profitably use.

11.5.1 Social Knowledge

Social knowledge enables the use of collaborative algorithms in which predictions about individuals are extrapolated from their peers opinions. Ratings are the most straightforward type of data used to model this knowledge. Rating knowledge is often conceptualized as a $m \times n$ matrix where m is number of users and n is the number of items and each entry corresponds to a user’s rating of an item. Model-based techniques use this matrix to create a model in advance, whereas memory-based techniques use it at the time of recommendation generation to produce the prediction.

The use of other types of opinion data is an area of active research. User tags are a promising source of opinion knowledge for recommendation [49, 23, 69, 40, 68]. In this case the data will be a tripartite graph of user, item and tag. Textual data in the form of reviews are also been studied, for example in [1]. Multi-dimensional knowledge sources such as these present a challenge for existing collaborative algorithms [2].

In heterogenous domains, social knowledge should be considered as a knowledge source since it is gathered by user’s input and does not need extensive knowledge engineering. However, social knowledge is not sufficiently accurate and reliable for high risk domains or for domains which need explanation.

Social knowledge will tend to be sparse for high churn domains. When items come and go quickly, the odds are reduced that any given user will have a chance

to rate any given item. As with other sparsity effects, the problem of churn can be ameliorated by having a large user population. Google News, for example, can take advantage of the site's large and active user base to implement collaborative recommendation even for the high-churn news domain. [16]

Using social knowledge is appropriate for domains with implicit type of interaction since it is possible to mine the users' behavior using machine learning and statistical techniques which are the typical algorithms in collaborative filtering. In domains with unstable user preference, the social knowledge can be misleading since the historical data is unreliable.

11.5.2 Individual

Individual knowledge is essential requirement for a recommender system to produce personalized recommendations.

In collaborative recommendation, individual knowledge regarding the target user is matched against social knowledge drawn from the user population at large. The most straightforward version of the process is that these sources are of the same type, and all that is needed is a similarity metric by which individuals can be compared. Ratings work well for this, but researchers have also used demographic data. Krulwich [32] uses demographic groups from marketing research to suggest a range of products and services. In other systems, machine learning is used to train a classifier based on demographic data [52].

In heterogenous domains, it might be difficult to transfer user's input on certain items for recommending other items. For example, it is not certain that two users who have similar taste about movies, would also like similar music.

In the absence of social knowledge, individual knowledge especially in the form of ratings can also be combined with content knowledge in the form of item features to build a classic content-based recommender that uses supervised classification learning [34, 47, 51].

A domain that requires knowledge of the user's short-term requirements is most likely suited to some kind of knowledge-based recommendation. The query is the most fundamental form of input for requirements: the user states, in whatever form the system accepts, what it is they are looking for. Constraints and preferences allow the user to limit and to rank options. For example, a dog owner might have a strict constraint that any apartment he rents accept his pet. A parent with young children might have a preference to be close to parks and playgrounds. In high risk domains and domains which need explanation, it is usually necessary to have explicit requirements and constraints from the user. Similarly, user requirements are more likely to be needed in domains with unstable preferences since the historical data are unreliable.

In many recommender systems more than one type of user input is used. For example, [43] uses users' priorities and constraints in a CBR recommender. A survey of preference elicitation methods can be found in [15].

11.5.3 Content

The most basic kind of content knowledge is item features, the kind of data that would typically be available in a product database, such as numeric values like price or symbolic tokens (destination airport, for example). These features can typically be used as is in a recommender system, although implementers will often want to restrict the feature space. For example, the entire cast and crew list for a movie may be available as feature data but will contain many sparse features of little utility. Trimming this list to just the top billed actors, director, and screenwriter would probably be sufficient.

If items are represented by unstructured documents such as news stories, the implementer will need to draw from information extraction (IE) techniques to extract and select features for use in recommendation. Standard techniques include eliminating stop words, stemming to simplify the feature space. Features can be reduced further by applying more sophisticated feature selection techniques such as information gain, mutual information, cross entropy or odds ratio [44]. More structured documents such as HTML pages offer additional opportunity for feature extraction. Applications of IE techniques to extract content knowledge from semi-structured and structured documents are discussed in [30]. Content knowledge in multimedia format presents an additional challenge. Hauptmann [22] discusses techniques from multimedia information retrieval. Osmar et al. survey multimedia data mining in [71] with a number of techniques useful for recommendation.

The quality of recommendations produced by a content-based or knowledge-based recommender will be entirely dependent on the quality of the content data on which its decisions are based. Indeed, the lack of reliable item features is often cited as a motivating factor for avoiding content-based recommendation. The cost involved in creating and maintaining a database of useful item features should not be underestimated, particularly for heterogeneous domains. In a domain like digital cameras or cell phones, for example, new technical innovations arrive regularly, requiring that the schema and the individual entries for each item be updated. If there are a large number of not-entirely-independent features extracted in a variety of ways, the system may be tolerant of noisy feature data. On the other hand, applications with high risk will need to pay special attention to having clean item features. Typically, manual review of feature data or manual labeling will be required.

11.5.3.1 Domain Knowledge

A knowledge-based recommender will typically need to know more than just what features are associated with what items. The most basic form of domain knowledge that a recommender can employ is an ontology over the item features. Such an ontology allows the system to reason about the relationship between features at a level deeper than just raw equality or difference. For example, the restaurant recommender Entree [12] has an ontology of different types of cuisine and can determine

Characteristic	Social	Individual	Content
Heterogeneous		May transfer poorly to unseen items	Difficult to engineer / maintain
High risk	Not sufficiently accurate / reliable	Requirements and constraints usually needed	Domain constraints needed
High churn	Sparse data		
Implicit		Detailed requirements not available	
Unstable preferences	Historical data unreliable	Historical data unreliable Need user requirements	
Explanations needed	Explanations weak	Requirements can be mapped to items	Domain knowledge can be used

Table 11.1: Impact of recommendation domain on knowledge sources

that a Thai restaurant would be more similar to a Vietnamese restaurant than a German one would be.

Many high risk choices have constraints imposed by the domain that a recommender needs to obey. For example, a recommender for financial products [19] may know that certain investment instruments are only suitable for customers with certain characteristics – a particular life insurance policy might not be available to persons over the age of 55, for example. The recommendation problem can be in some cases formulated entirely as constraint satisfaction with constraints being contributed both by the user and by the system.

A final category of domain knowledge is means-ends knowledge, which is the knowledge that enables a system to map between the user’s goals (ends) and the products that might satisfy them (means). For example, a camera buyer might not know much about digital cameras, but he might know that he wants to take photos of his daughter’s basketball games. Part of the reason that users benefit from recommender systems is that they can make good choices without necessarily being conversant with all of the complexities of the product space.

Table 11.1 summarizes these domain considerations and their impact on knowledge sources.

11.6 Mapping Domains to Technologies

Some basic considerations come to the fore in considering the recommendation domain. First, there are some domain types for which social knowledge seems not very useful, in particular, high risk domains and ones with high churn. In high churn domains, there may not be enough time for an item to build up a reputation among a large number of peer users before it is replaced with other items.

Factor		Collaborative	Content-Based	Knowledge-Based
Heterogeneous	Low	✓	✓	✓
	High	+	-	-
Risk	Low	✓	✓	✓
	High	-	-	+
Churn	Low	✓	✓	✓
	High	-	+	+
Interaction style	Implicit	+	+	-
	Explicit	✓	✓	+
Preferences	Stable	✓	✓	✓
	Unstable	-	-	+
Scrutability	Required	-	-	+
	Not Required	✓	✓	✓

Table 11.2: Domain factors and recommendation techniques

When there is large risk associated with a domain, most users are going to need a more convincing explanation of the appropriateness of a recommendation beyond simply that others liked it. This is particularly important if we consider the problem of robustness in collaborative systems discussed in Chapter 25. Even a profile learned from the user's previous interactions might not be acceptable if adherence to it overrode considerations crucial to the current context.

Similarly, if we look at the interaction, we can see that it is not always possible to gather every kind of knowledge type from every type of interaction. In systems with implicit inputs, we do not gather any kind of direct requirements from the user (although it is sometimes possible to extract an implicit query from the user's activity with other applications, as done in the Watson system [8].)

Preference instability favors knowledge-based techniques. Learning over a user's prior interactions may turn out to be a hindrance rather than a help. However, in certain cases, such as web personalization, users may provide enough implicit data in a single session to form a useful profile that can be compared to others.

Table 11.2 shows the influence of the different domain factors on the choice of recommendation approach.

In cases where the criteria do not help to reach a definitive conclusion, it is worth noting that the different technologies do have different implementation and maintenance costs. Collaborative recommendation is likely to be the least expensive to implement. It requires a database of user ratings, but it does not require clean, well-engineered item features, which is the minimum requirement for the other recommendation technologies. Knowledge-based technologies are going to be the most expensive approach requiring knowledge engineering and continuing maintenance. So, a developer might wish to start by implementing the least expensive solution compatible with the domain.

Another factor to consider is that with hybrid recommendation it is possible to combine techniques. For example, to deal with a heterogeneous environment with unstable preferences, a hybrid between content-based and collaborative recommendation may be desirable. See [11] for more details.

11.6.1 Algorithms

If a domain can be clearly characterized as appropriate for one recommendation technology or another, a natural next question is which algorithms are appropriate? A thorough treatment of this question is beyond the scope of this chapter. Readers should review the relevant chapters of this volume. In the case of collaborative recommendation, it is possible to put forward some considerations.

In user-based collaborative recommendation, a subset of appropriate users are chosen based on their similarity to the active user, and a weighted aggregate of their ratings is used to generate predictions for the active user at run-time. Different implementations of collaborative filtering apply variations of the neighborhood-based prediction algorithms. Herlocker et al. [25] presents an empirical analysis of design choices in such algorithms and analyzes the variations of similarity metrics, weighting approaches, combination measures, and rating normalization.

Item-based collaborative filtering is a memory-based algorithm which explores the relationship between items as a function of how users have rated them. The item-based version of kNN algorithm has been shown to scale better and produce more accurate recommendation than user-based for large item collections [59].

Memory-based nearest-neighbor algorithms have two important computational limits: scale and sparsity. The need to compare each user against every other (n^2 comparisons) makes these techniques impractical for large collections. Also, the need to directly compare item ratings means that in very sparse collections, users may have very few neighbors.

In some databases, overall sparsity may hide the fact that there are dense sub-regions of the item space. Exponential popularity curves may make it possible to employ memory-based techniques because it is possible to find agreement among people or items in the dense sub-region and use that agreement to recommend in the sparse space [27]. (Jester [20] does this directly by creating a highly dense region of jokes rated by all users).

Dimensionality reduction (by way of singular value decomposition, latent semantic analysis, or other techniques) is by now a standard approach for coping with sparsity in ratings databases. [60, 74]. Various forms of compression and/or dimensionality reduction usually require extensive off-line computation, but as a result scale much better. The movie rating data released by Netflix prize which was also used for KDD cup competition in 2007 is an example of large, sparse data which motivated many research groups to develop new model-based algorithms [56, 33].

Other model-based collaborative algorithms include different machine learning techniques such as Bayesian networks [7], and clustering [7, 66]. Bayesian networks are more practical for domains with high user preferences stability so that the user preference changes slowly with respect to the time needed to build the model.

Clustering techniques identify groups of users who appear to have similar preferences. Once the clusters are created, predictions for an individual can be made by averaging the opinions of the other users in that cluster[59]. Clustering techniques usually produce less-personal recommendations than other methods, and in some cases, the clusters have worse accuracy than nearest neighbor algorithms [7].

Clustering techniques can also be applied as a first step for shrinking the candidate set in a nearest neighbor algorithm or for distributing nearest neighbor computation across several recommender engines. While dividing the users into clusters may reduce the accuracy of recommendations, pre-clustering may be a worthwhile trade-off between accuracy and throughput [59].

11.6.2 Sample Recommendation Domains

Table 11.3 illustrates the application of these criteria in 10 different domains where recommendation applications exist. Not all combinations of the six criteria are represented, but we can see that the considerations given above are fairly predictive. High-risk domains generally lead to knowledge-based recommendation; scrutability is also a good predictor of this. Heterogeneous domains are handled largely with collaborative recommendation. Web page recommendation looks a bit contradictory when we consider high churn and preference instability, which would seem to militate against collaborative methods. However, as discussed above, database size can compensate for preference instability and these recommenders do collect large amounts of implicit preference data in each session. Also, heterogeneity is high, which argues in favor of using social knowledge.

Domain	Risk	Churn	Heterogeneous	Preferences	Interaction Style	Scrutability	Examples	Technology
News	Low	High	Low	Stable?	Implicit	Not required	Yahoo news[6] ACR news[45] and [38] Google news[16]	Content-based Collaborative-Filtering
E-commerce	Low	High	High	Stable	Implicit	Not required	Amazon.com eBay	Collaborative-Filtering
Web Page Recommender	Low	High	High	Unstable	Implicit	Not required	[9, 36, 4]	Collaborative-Filtering Hybrid
Movie	Low	Low	Low	Stable	Implicit	Not required	Netflix[50, 64] Movielens[21]	Collaborative-Filtering
Music	Low	Low	Low	Stable?	Implicit	Not required	Pandora and [24, 28, 14]	Content-based Hybrid
Financial-services Life-insurance	High	Low	Low	Stable	Explicit	Required	Kobas4MS[17] FSAdvisor[19] [65]	Knowledge-Based
Software Engineering	Low	Low	Low	Stable	Explicit /Implicit	Required	[13] and [29]	Hybrid and Content-based
Tourism	High	Low	Low	Unstable	Explicit	Required	Travel Recommender [55] [37]	Content-based Knowledge-based
Job search Recruiting	High	Low	Low	Stable	Explicit	Required	CASPER [35] and [39]	Content-based
Real Estate	High	Low	Low	Stable	Explicit	Required	RentMe [10] FlatFinder[67] and [73]	Knowledge-based

Table 11.3: Sample domains for recommendation

11.7 Conclusion

This chapter considers recommender systems as intelligent systems, and as such, dependent on knowledge. The differences between recommendation approaches can be best understood through reference to the different knowledge sources that they employ. By considering how domain characteristics impact the availability and quality of knowledge sources, we can connect recommendation technologies and domain characteristics.

We have examined 6 different factors: heterogeneity, risk, churn, preference stability, interaction style, and scrutability. The factors allow us to characterize different recommendation domains and map those characteristics to appropriate recommendation technologies. Application of these criteria to some existing systems shows that they are good predictors of the technologies that have been successfully employed both in research and applications.

Acknowledgements

This article is based on research performed by Ms. Ramezani at IBM Watson Research Center during the summer of 2007. An abbreviated version of the article with additional authors Lawrence Bergman, Rich Thompson and Bamshad Mobasher appeared as “Selecting and Applying Recommendation Technologies” at the Workshop on Recommendation and Collaboration at the Intelligent User Interfaces conference 2008.

References

1. Aciar, S., Zhang, D., Simoff, S., Debenham, J.: Informed recommender agent: Utilizing consumer product reviews through text mining. In: WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology, pp. 37–40. IEEE Computer Society, Hong Kong (2006).
2. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.* **23**(1), 103–145 (2005).
3. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749 (2005)
4. Anand, S.S., Mobasher, B.: Introduction to intelligent techniques for web personalization. *ACM Trans. Internet Technol.* **7**(4), 18 (2007).
5. Berkovsky, S., Aroyo, L., Heckmann, D., Houben, G.J., Kröner, A., Kuflik, T., Ricci, F.: Providing context-aware personalization through cross-context reasoning of user modeling data. In: S. Berkovsky, K. Cheverst, P. Dolog, D. Heckmann, T. Kuflik, P. Mylonas, J. Picault, J. Vassileva (eds.) *UbiDeUM'2007 - International Workshop on Ubiquitous and Decentralized User Modeling*, at User Modeling 2007, 11th International Conference, UM 2007, Corfu, Greece, June 26, 2007, Proceedings (2007).

6. Billsus, D., Pazzani, M.J.: User modeling for adaptive news access. *User Modeling and User-Adapted Interaction* **10**(2-3), 147–180 (2000)
7. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 43–52 (1998).
8. Budzik, J., Hammond, K., Birnbaum, L.: Information access in context. *Knowledge based systems* **14**(1-2), 37–53 (2001)
9. Buono, P., Costabile, M.F., Guida, T., Piccinno, A.: Integrating user data and collaborative filtering in a web recommendation system. In: *Lecture Notes in Computer Science: Hypermedia: Openness, Structural Awareness, and Adaptivity*, pp. 192–196. SpringerLink (2002)
10. Burke, R.: Knowledge-based recommender systems. In: *Encyclopedia of Library and Information Systems*. Marcel Dekker (2000)
11. Burke, R.: Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction* **12**(4), 331–370 (2002)
12. Burke, R.: Interactive critiquing for catalog navigation in e-commerce. *Artif. Intell. Rev.* **18**(3-4), 245–267 (2002)
13. Castro-Herrera, C., Duan, C., Cleland-Huang, J., Mobasher, B.: Using data mining and recommender systems to facilitate large-scale, open, and inclusive requirements elicitation processes. In: *RE '08: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*, pp. 165–168. IEEE Computer Society, Washington, DC, USA (2008).
14. Chen, H.C., Chen, A.L.P.: A music recommendation system based on music data grouping and user interests. In: *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pp. 231–238. ACM (2001).
15. Chen, L., Pu, P.: Survey of preference elicitation methods. Tech. rep., Swiss Federal Institute of Technology in Lausanne (EPFL), Lausanne, Switzerland (2004).
16. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pp. 271–280. ACM, New York, NY, USA (2007).
17. Felfernig, A.: Koba4ms: Selling complex products and services using knowledge-based recommender technologies. In: *CEC '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*, pp. 92–100. IEEE Computer Society (2005).
18. Felfernig, A., Burke, R.: Constraint-based recommender systems: technologies and research issues. In: *ICEC '08: Proceedings of the 10th international conference on Electronic commerce*, pp. 1–10. ACM, New York, NY, USA (2008).
19. Felfernig, A., Kiener, A.: Knowledge-based interactive selling of financial services with fsadvisor. In: *Proceedings of the National Conference on Artificial Intelligence*, pp. 1475–1482 (2005)
20. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.* **4**(2), 133–151 (2001).
21. Good, N., Schafer, J.B., Konstan, J.A., Borchers, A., Sarwar, B.M., Herlocker, J.L., Riedl, J.: Combining collaborative filtering with personal agents for better recommendations. In: *AAAI:Conference on Artificial Intelligence*, pp. 439–446 (1999).
22. Hauptmann, A.G.: Integrating and using large databases of text, images, video, and audio. *Intelligent Systems and Their Applications, IEEE* **14**(5), 34 – 35 (1999)
23. Hayes, C., Avesani, P., Veeramachaneni, S.: An analysis of the use of tags in a blog recommender system. In: M.M. Veloso (ed.) *IJCAI-07, the International Joint Conference on Artificial Intelligence*, pp. 2772–2777 (2007)
24. Hayes, C., Cunningham, P.: Smart radio: Building music radio on the fly. In: *Proceedings of Expert Systems 2000*, Cambridge, UK (2000)
25. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 230–237. ACM Press, New York, NY, USA (1999).

26. Herlocker, J.L., Konstan, J.A., Riedl, J.: Explaining collaborative filtering recommendations. In: CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work, pp. 241–250. ACM Press (2000).
27. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **22**(1), 5–53 (2004).
28. Hijikata, Y., Iwahama, K., Nishida, S.: Content-based music filtering system with editable user profile. In: SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, pp. 1050–1057. ACM (2006).
29. Holmes, R., Walker, R.J., Murphy, G.C.: Approximate structural context matching: An approach to recommend relevant examples. *IEEE Transactions on Software Engineering* **32**(12), 952–970 (2006).
30. Kosala, R., Blockeel, H.: Web mining research: a survey. *SIGKDD Explor. Newsl.* **2**(1), 1–15 (2000).
31. Kovacs, A.I., Ueno, H.: Recommending in context: A spreading activation model that is independent of the type of recommender system and its contents. In: G. Uchyigit (ed.) Proceedings of Workshop on Web Personalisation, Recommender Systems and Intelligent User Interfaces In conjunction with AH 2006: International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems. Dublin, Ireland (2006)
32. Krulwich, B.: Lifestyle finder: Intelligent user profiling using large-scale demographic data. *Artificial Intelligence Magazine* **18**(2) (1997)
33. Kurucz, M., Benczúr, A.A., Kiss, T., Nagy, I., Szabó, A., Torma, B.: Kdd cup 2007 task 1 winner report. *SIGKDD Explor. Newsl.* **9**(2), 53–56 (2007).
34. Lang, K.: Newsweeder: Learning to filter netnews. In: Proceedings of the Twelfth International Conference on Machine Learning, pp. 331–339 (1995)
35. Lee, D.H., Brusilovsky, P.: Fighting information overflow with personalized comprehensive information access: A proactive job recommender. In: ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems, p. 21. IEEE Computer Society, Washington, DC, USA (2007).
36. Li, J., Zaiane, O.R.: Combining usage, content, and structure data to improve web site recommendation. *Lecture Notes in Computer Science : E-Commerce and Web Technologies* pp. 305–315 (2004)
37. Mahmood, T., Ricci, F., Venturini, A., Hpkén, W.: Adaptive recommender systems for travel planning. In: P. O'Connor, W. Hpkén, U. Gretzel (eds.) *Information and Communication Technologies in Tourism 2008*, pp. 1–11. Springer (2008).
38. Maidel, V., Shoval, P., Shapira, B., Taieb-Maimon, M.: Evaluation of an ontology-content based filtering method for a personalized newspaper. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, pp. 91–98. ACM (2008).
39. Malinowski, J., Keim, T., Wendt, O., Weitzel, T.: Matching people and jobs: A bilateral recommendation approach. In: HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences, pp. 137c–137c. IEEE Computer Society (2006).
40. Markines, B., Stoilova, L., Menczer, F.: Bookmark hierarchies and collaborative recommendation. In: Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence. AAAI Press (2006)
41. McSherry, D.: Explaining the pros and cons of conclusions in cbr. In: P.A.G. Calero, P. Funk (eds.) *Proceedings of the European Conference on Case-Based Reasoning (ECCBR-04)*, pp. 317–330. Springer (2004). Madrid, Spain
42. McSherry, D.: Explanation in recommender systems. *Artif. Intell. Rev.* **24**(2), 179–197 (2005)
43. McSherry, D., Aha, D.W.: Mixed-initiative relaxation of constraints in critiquing dialogues. In: ICCBR '07: Proceedings of the 7th international conference on Case-Based Reasoning, vol. 4626, pp. 107–121 (2007)
44. Mladenic, D., Grobelnik, M.: Feature selection for unbalanced class distribution and naive bayes. In: ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning, pp. 258–267. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)

45. Mobasher, B., Cooley, R., Srivastava, J.: Automatic personalization based on web usage mining. *Communications of the ACM* **43**(8), 142–151 (2000).
46. Montaner, M., López, B., Rosa, J.L.D.L.: A taxonomy of recommender agents on the internet. *Artif. Intell. Rev.* **19**(4), 285–330 (2003).
47. Mooney, R.J., Roy, L.: Content-based book recommending using learning for text categorization. In: *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, pp. 195–204. ACM Press (2000).
48. Moskovitch, R., Elovici Y., Rokach L., Detection of unknown computer worms based on behavioral classification of the host. *Computational Statistics and Data Analysis*, **52**(9):4544–4566 (2008)
49. Niwa, S., Doi, T., Honiden, S.: Web page recommender system based on folksonomy mining for itng 06. In: *Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on*, pp. 388–393 (2006).
50. Paterek, A.: Improving regularized singular value decomposition for collaborative filtering. In: *Proc. of the of the KDD Cup and Workshop 2007 (KDD 2007)* (2007)
51. Pazzani, M., Billsus, D.: Learning and revising user profiles: The identification of interesting web sites. *Machine Learning: Special issue on multistrategy learning* **27**(3), 313–331 (1997).
52. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.* **13**(5-6), 393–408 (1999)
53. Rack, C., Arbanowski, S., Steglich, S.: A Generic Multipurpose recommender System for Contextual Recommendations, pp. 445–450. IEEE Computer Society, Washington, DC, USA (2007).
54. Resnick, P., Varian, H.R.: Recommender systems. *Commun. ACM* **40**(3), 56–58 (1997)
55. Ricci, F.: Travel recommender systems. In: *IEEE Intelligent Systems*, pp. 55–57 (2002)
56. Rosset, S., Perlich, C., Liu, Y.: Making the most of your data: Kdd cup 2007 "how many ratings" winner's report. *SIGKDD Explor. Newsl.* **9**(2), 66–69 (2007).
57. Roth-Berghofer, T.R.: Explanations and case-based reasoning: Foundational issues. In: *Advances in Case-Based Reasoning*, pp. 389–403. Springer Verlag (2004)
58. Salam, M., Reilly, J., McGinty, L., Smyth, B.: Knowledge discovery from user preferences in conversational recommendation. In: *Knowledge Discovery in Databases: PKDD 2005*, pp. 228–239. Springer Berlin / Heidelberg (2005)
59. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pp. 285–295. ACM (2001).
60. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Application of dimensionality reduction in recommender systems—a case study. In: *Proceedings of ACM WebKDD Workshop* (2000)
61. Schafer, J.B., Konstan, J.A., Riedl, J.: E-commerce recommendation applications. *Data Mining and Knowledge Discovery* **5**(1-2), 115–153 (2001)
62. van Setten, M., Pokraev, S., Koolwaaij, J.: Context-aware recommendations in the mobile tourist application compass. In: W. Nejdl, P. De Bra (eds.) *Adaptive Hypermedia 2004*, pp. 235–244. Springer Verlag (2004).
63. Smyth, B., Balfé, E., Freyne, J., Briggs, P., Coyle, M., Boydell, O.: Exploiting query repetition and regularity in an adaptive community-based web search engine. *User Modeling and User-Adapted Interaction* **14**(5), 383–423 (2005).
64. Takács, G., Pilászy, I., Németh, B., Tikk, D.: On the gravity recommendation system. In: *Proc. of the of the KDD Cup and Workshop 2007 (KDD 2007)*, pp. 22–30 (2007)
65. Tartakovski, A., Schaaf, M., Bergmann, R.: Retrieval and configuration of life insurance policies. In: *Lecture Notes in Computer Science, Case-Based Reasoning Research and Development*, pp. 552–565. Springer Berlin / Heidelberg (2005)
66. Ungar, L., Foster, D., Andre, E., Wars, S., Wars, F.S., Wars, D.S., Whispers, J.H.: Clustering methods for collaborative filtering. In: *Workshop on Recommender Systems at the 15th National Conference on Artificial Intelligence. AAAI Press* (1998)

67. Viappiani, P., Pu, P., Faltings, B.: Conversational recommenders with adaptive suggestions. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 89–96. ACM (2007).
68. Wu, H., Zubair, M., Maly, K.: Harvesting social knowledge from folksonomies. In: HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia, pp. 111–114. ACM Press (2006).
69. Xu, Z., Fu, Y., Mao, J., Su, D.: Towards the semantic web: Collaborative tag suggestions. In: Proceedings of the Collaborative Web Tagging Workshop at the WWW 2006. Edinburgh, Scotland (2006)
70. Yu, Z., Zhou, X., Zhang, D., Chin, C.Y., Wang, X., Men, J.: Supporting context-aware media recommendations for smart phones. *Pervasive Computing* 5(3), 68–75 (2006).
71. Zaiiane, O.R., Han, J., Li, Z.N., Chee, S.H., Chiang, J.Y.: Multimediaminer: a system prototype for multimedia data mining. In: SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, pp. 581–583. ACM (1998).
72. Zanker, M.: A collaborative constraint-based meta-level recommender. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems, pp. 139–146. ACM, New York, NY, USA (2008).
73. Zhang, J., Pu, P.: A comparative study of compound critique generation in conversational recommender systems. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 4018 NCS, pp. 234–243. Springer Verlag, Heidelberg, D-69121, Germany (2006)
74. Zhang, S., Wang, W., Ford, J., Makedon, F., Pearlman, J.: Using singular value decomposition approximation for collaborative filtering. In: CEC '05: Proceedings of the Seventh IEEE International Conference on E-Commerce Technology, pp. 257–264. IEEE Computer Society, Washington, DC, USA (2005).

Chapter 12

Recommender Systems in Technology Enhanced Learning

Nikos Manouselis, Hendrik Drachsler, Riina Vuorikari, Hans Hummel and Rob Koper

Abstract Technology enhanced learning (TEL) aims to design, develop and test socio-technical innovations that will support and enhance learning practices of both individuals and organisations. It is therefore an application domain that generally covers technologies that support all forms of teaching and learning activities. Since information retrieval (in terms of searching for relevant learning resources to support teachers or learners) is a pivotal activity in TEL, the deployment of recommender systems has attracted increased interest. This chapter attempts to provide an introduction to recommender systems for TEL settings, as well as to highlight their particularities compared to recommender systems for other application domains.

Nikos Manouselis
Greek Research and Technology Network (GRNET S.A.), 56 Messogeion Av., 115 27, Athens, Greece e-mail: nikosm@ieee.org

Hendrik Drachsler
Centre for Learning Sciences and Technologies (CELSTEC), Open Universiteit Nederland e-mail: hendrik.drachsler@ou.nl

Riina Vuorikari,
European Schoolnet (EUN), 24, Rue Paul Emile Janson, 1050 Brussels, Belgium e-mail: riina.vuorikari@eun.org

Hans Hummel
Centre for Learning Sciences and Technologies (CELSTEC), Open Universiteit Nederland e-mail: hans.hummel@ou.nl

Rob Koper
Centre for Learning Sciences and Technologies (CELSTEC), Open Universiteit Nederland e-mail: rob.koper@ou.nl

12.1 Introduction

Technology enhanced learning (TEL) aims to design, develop and test socio-technical innovations that will support and enhance learning practices of both individuals and organisations. It is therefore an application domain that generally covers technologies that support all forms of teaching and learning activities. Since information retrieval (in terms of searching for relevant learning resources to support teachers or learners) is a pivotal activity in TEL, the deployment of recommender systems has attracted increased interest.

As in any other field where there is a massive increase in product variety, in TEL there is also a need for better findability of (mainly digital) learning resources. For instance, during the past few years, numerous repositories with digital learning resources have been set up [96]. Prominent US examples are repositories such as MERLOT (<http://www.merlot.org>) that has more than 20,000 learning resources (and about 70,000 registered users), and OER Commons (www.oercommons.org) with about 18,000 resources. In Europe, a typical example is European Schoolnet's Learning Resource Exchange (<http://lreforschools.eun.org>) that federates more than 43,000 learning resources from 25 different content providers in Europe and beyond. Apart from learning content, learning resources may also include learning paths (that can help them navigate through appropriate learning resources) or relevant peer-learners (with whom collaborative learning activities can take place).

In this plethora of online learning resources available, and considering the various opportunities for interacting with such resources that often occur in both formal and non-formal settings, all user groups of TEL systems can benefit from services that help them identify suitable learning resources from a potentially overwhelming variety of choices. As a consequence, the concept of recommender systems has already appeared in TEL. Latest efforts to identify relevant research in this field, and to bring together researchers working on similar topics, have been the annual workshop series of Social Information Retrieval for Technology Enhanced Learning (SIRTEL), and a Special Issue on Social Information Retrieval for TEL in the *Journal of Digital Information* [31]. These efforts resulted in a number of interesting conclusions, the main ones being that:

1. There is a large number of recommender systems that have been deployed (or that are currently under deployment) in TEL settings;
2. The information retrieval goals that TEL recommenders try to achieve are often different to the ones identified in other systems (e.g. product recommenders);
3. There is a need to identify the particularities of TEL recommender systems, in order to elaborate on methods for their systematic design, development and evaluation.

In this direction, the present chapter attempts to provide an introduction to issues related to the deployment of recommender systems in TEL settings, keeping in mind the particularities of this application domain. The main contributions of this chapter are the following:

- Discuss the background of recommender systems in TEL, especially in relation to the particularities of the TEL context.
- Reflect on user tasks that are supported in TEL settings, and how they compare to typical user tasks in other recommender systems.
- Review related work coming from adaptive educational hypermedia (AEH) systems and the learning networks (LN) concept.
- Assess the current status of development of TEL recommender systems.
- Provide an outline of particularities and requirements related to the evaluation of TEL recommender systems that can provide a basis for their further application and research in educational applications.

12.2 Background

TEL as context

TEL relates to data generated in different types of educational settings, which are usually called macro-context [99]. This concept has significant influence on which user actions are possible and how they can be interpreted. Examples of these dimensions of macro-context include dimensions such as educational level, formal and informal learning, delivery setting and different user roles. Examples of the educational level are K-12 education, Higher Education (HE), Vocational Education and Training (VET) and workplace training.

A formal setting for learning includes learning offers from educational institutions (e.g. universities, schools) within a curriculum or syllabus framework, and is characterised as highly structured, leading to a specific accreditation and involving domain experts to guarantee quality. This traditionally occurs in teacher-directed environments with person-to-person interactions, in a live and synchronous manner.

An informal setting, on the other hand, is described in the literature as a learning phase of so called lifelong learners who are not participating in any formal learning and are responsible for their own learning pace and path [17, 64]. The learning process depends to a large extent on individual preferences or choices and is often self-directed [8]. The resources for informal learning might come from sources such as expert communities, work context, training or even friends might offer an opportunity for an informal competence development.

The TEL involvement can be characterised by the provision of blended learning opportunities to purely distant educational ones [71]. Blended learning combines traditional face-to-face learning with computer-supported learning [36]. Distance education, on the other hand, can be delivered using TEL environments in either synchronous or asynchronous ways. Traditionally, distance learning was more related to self-paced learning and learning-materials interactions that typically occurred in an asynchronous way [36]. However, live streaming and virtual, personal learning

Table 12.1: User tasks supported by current recommender systems and requirements for TEL recommender systems

<i>Tasks</i>	<i>Description</i>	<i>Generic recommender</i>	<i>TEL recommenders</i>	<i>New requirements</i>
Existing User Tasks supported by Recommender Systems				
1. ANNOTATION IN CONTEXT	Recommendations while user carries out other tasks	E.g. predicting how relevant the links are within a web page	E.g. predicting relevance/usefulness of items in the reading list of a course	Explore attributes for representing relevance/usefulness in a learning context
2. FIND GOOD ITEMS	Recommendations of suggested items	E.g. receiving list of web pages to visit	E.g. receiving a selected list of online educational resources around a topic	None
3. FIND ALL GOOD ITEMS	Recommendation of all relevant items	E.g. receiving a complete list of references on a topic	E.g. suggesting a complete list of scientific literature or blog postings around a topic	None
4. RECOMMEND SEQUENCE	Recommendation of a sequence of items	E.g. receive a proposed sequence of songs	E.g. receiving a proposed sequence through resources to achieve a particular learning goal	Explore formal and informal attributes for representing relevancy to a particular learning goal
5. JUST BROWSING	Recommendations out of the box while user is browsing	E.g. people that bought this, have also bought that	E.g. receiving recommendations for new courses on the university site	Explore formal and informal attributes for representing relevance/usefulness in a learning context
6. FIND CREDIBLE RECOMMENDER	Recommendations during initial exploration/testing phase of a system	E.g. movies that you will definitely like	E.g. restricting course recommendations to ones with high confidence /credibility	Explore criteria for measuring confidence and credibility in formal and informal learning
TEL User Tasks that could be supported by Recommender Systems				
1. FIND NOVEL RESOURCES	Recommendations of particularly new or novel items	E.g. receiving recommendations about latest additions or particularly controversial items	E.g. receiving very new and/or controversial resources on covered topics	Explore recommendation techniques that select items beyond their similarity
2. FIND PEERS	Recommendation of other people with relevant interests	E.g. being suggested profiles of users with similar interests	E.g. being suggested peer students in the same class	Explore attributes for measuring the similarity with other people
3. FIND GOOD PATHWAYS	Recommendation of alternative learning paths through learning resources	E.g. receive alternative sequences of similar songs	E.g. receiving a list of alternative learning paths over the same resources to achieve a specific learning goal	Explore criteria for the construction and suggestion of alternative (but similar) sequences

environments (e.g. Web 2.0) have facilitated the development of synchronous distance learning services in formal educational settings.

Lastly, different actors and needs can be identified in TEL. A distinction can be made between the teacher-directed interaction and learner-directed learning processes. This has ramifications concerning the intended users of TEL environments. While macro-context has large implications for interpretation and design, its aspects are fairly agreed upon, and it is comparatively easy to measure. Micro-context is a more contested notion and more difficult to measure. However, while macro-context is domain-specific, concepts for micro-context range over more diverse fields.

TEL Recommendation goals

In the past, the development of recommender systems has been related to a number of relevant user tasks that the recommender system supports within some particular application context (see Chapter 7). More specifically, Herlocker et al. [38] have related popular (or less popular) user tasks with a number of specific recommendation goals that are included in Table 1. Generally speaking, most of these already identified recommendation goals and user tasks are valid in the case of TEL recommender systems as well. For example, in a recommender system supporting learners to achieve a specific learning goal, “providing annotation in context” or “recommending a sequence” of learning resources are relevant tasks. In Table 1, examples are given of how recommendation could support TEL-relevant activities for all the tasks that Herlocker et al. [38] have identified. In addition, it includes a comment about any additional requirements that this brings forward for the developers of TEL recommender systems.

On the other hand, in comparison to the typical item recommendation scenario, there are several particularities to be considered regarding what kind of learning is desired, e.g. learning a new concept or reinforce existing knowledge may require different type of learning resources. This is reflected in the second part of Table 1, with examples of user tasks that are particularly interesting for TEL. Again, a comment on any additional requirements for developers of TEL recommenders is included.

Apart from this initial identification of tasks, recommendation in a TEL context has many particularities that are based on the richness of the pedagogical theories and models. For instance, for learners with no prior knowledge in a specific domain, relevant pedagogical rules such as Vygotsky’s “zone of proximal development” could be applied: e.g. ‘recommended learning objects should have a level slightly above learners’ current competence level’ [102]. Different from buying products, learning is an effort that often takes more time and interactions compared to a commercial transaction. Learners rarely achieve a final end state after a fixed time. Instead of buying a product and then owning it, learners achieve different levels of competences that have various levels in different domains. In such scenarios, it is important to identify the relevant learning goals and support learners in achieving them. On the other hand, depending on the context, some particular user task may

be prioritised. This could call for recommendations whose time span is longer than the one of product recommendations, or recommendations of similar learning resources, since recapitulation and reiteration are central tasks of the learning process [68].

As for teacher-centered learning context, different tasks need to be supported. These tasks cover both the ones related to the preparation of lessons, the delivery of a lesson (i.e. the actual teaching), and the ones related to the evaluation/assessment. For instance, to prepare a lesson the teacher has certain educational goals to fulfil and needs to match the delivery methods to the profile of the learners (e.g. their previous knowledge). Lesson preparation can include a variety of information seeking tasks, such as finding content to motivate the learners, to recall existing knowledge, to illustrate, visualise and represent new concepts and information. The delivery can be supported in using different pedagogical methods (either supported with TEL or not), whose effectiveness is evaluated according to the goals set. A TEL recommender system could support one or more of these tasks, leading to a variety of recommendation goals.

Thus, although the previously identified user tasks and recommendation goals can be considered valid in a TEL context, there are several particularities and complexities. This means that simply transferring a recommender system from an existing (e.g. commercial) content to TEL may not accurately meet the needs of the targeted users. In TEL, careful analysis of the targeted users and their supported tasks should be carried out, before a recommendation goal is defined and a recommender system is deployed. This means that the TEL recommendation goals can be rather complex: for example, a typical TEL recommender system could suggest a number of alternative learning paths throughout a variety of learning resources, either in the form of learning sequences or hierarchies of interacting learning resources. This should take place in a pedagogically meaningful way that will reflect the individual learning goals and targeted competence levels of the user, depending on proficiency levels, specific interests and the intended application context. A number of context variables have to be considered, such as user attributes, domain characteristics, and intelligent methods that can be engaged to provide personalised recommendations. Extensive work on these topics has been carried out in the past, in the area of adaptive educational hypermedia systems.

12.3 Related Work

Web systems generally suffer from the inability to satisfy the heterogeneous needs of many users. To address this challenge, a particular strand of research that has been called *adaptive web systems* (or *adaptive hypermedia*) tried to overcome the shortcomings of traditional ‘one-size-fits-all’ approaches by exploring ways in which Web-based could adapt their behaviour to the goals, tasks, interests, and other characteristics of interested users [12]. A particular category of adaptive systems has been the one dealing with educational applications, called *adaptive educational hy-*

permedia (AEH) systems. Since one can say that AEH systems address issues of high relevance to TEL recommender systems, this section provides a brief overview of related work, trying to identify commonalities and differences that could be of relevance for TEL recommenders.

Adaptive Educational Hypermedia

Adaptive web systems belong to the class of user-adaptive software systems [87]. According to Oppermann [73] a system is called adaptive “if it is able to change its own characteristics automatically according to the user’s needs”. Adaptive systems consider the way the user interacts with the system and modify the interface presentation or the system behaviour accordingly [108]. Jameson [43] adds an important characteristic: “A user-adaptive system is an interactive system which adapts its behaviour to each individual user on the basis of nontrivial inferences from information about that user”.

Adaptive systems help users find relevant items in a usually large information space, by essentially engaging three main adaptation technologies [12]: adaptive content selection, adaptive navigation support, and adaptive presentation. The first of these three technologies comes from the field of adaptive information retrieval (IR) [6] and is associated with a search-based access to information. When the user searches for relevant information, the system can adaptively select and prioritise the

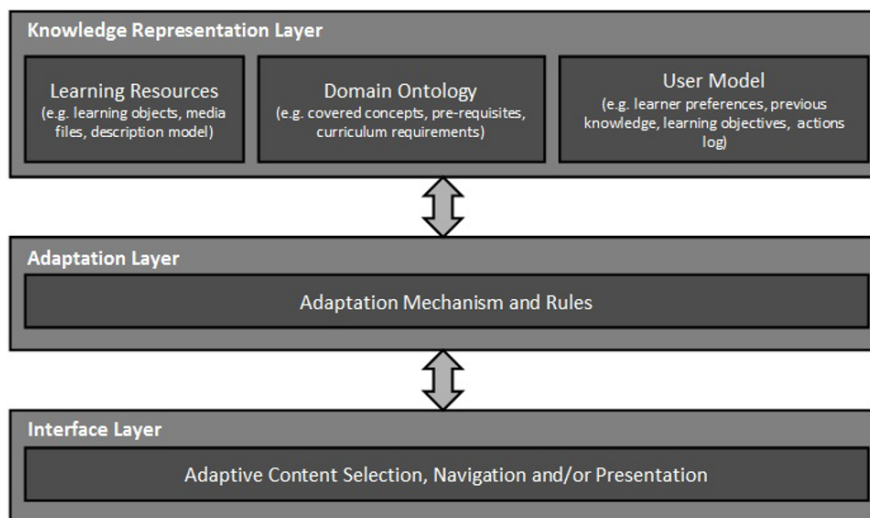


Fig. 12.1: Generic layers within a simplified example architecture of an educational AEH (adapted from [47])

most relevant items. The second technology was introduced by adaptive hypermedia systems [9] and is associated with a browsing-based access to information. When the user navigates from one item to another, the system can manipulate the links (e.g., hide, sort, annotate) to guide the user adaptively to most relevant information items. The third technology has its roots in the research on adaptive explanation and adaptive presentation in intelligent systems [70, 76]. It deals with presentation, not access to information. When the user gets to a particular page, the system can present its content adaptively.

As Brusilovksy [10] describes, educational hypermedia was one of the first application areas of adaptive systems. A simplified architecture of the layers within an educational AEH system is presented in Figure 12.1. This architecture includes: a layer including the representation and organisation of knowledge about educational content (*learning resources*), the domain (*domain ontology*), and the user (*user model*); a layer that includes the adaptation mechanisms and rules; and a layer that provides the run-time adaptation results to the user. A number of pioneer adaptive educational hypermedia systems were developed between 1990 and 1996, which he roughly divided into two research streams. The systems of one of these streams were created by researchers in the area of intelligent tutoring systems (ITS) who were trying to extend traditional student modelling and adaptation approaches developed in this field to ITS with hypermedia components [14, 34, 77]. The systems of the other stream were developed by researchers working on educational hypermedia in an attempt to make their systems adapt to individual students [19, 21, 39, 48]. AEH research has often followed a top-down approach, greatly depending on expert knowledge and involvement in order to identify and model TEL context variables. For example, Cristea [18] describes a number of expertise-demanding tasks when AEH content is authored: initially creating the resources, labelling them, combining them into what is known as a domain model; then, constructing and maintaining the user model in a static or dynamic way, since it is crucial for achieving successful adaptation in AEH. Generally speaking, in AEH a large amount of user-related information (characterising needs and desires) has to be encoded in the content creation phase. This can take place in formal educational settings when the context variables are usually known, and there is a large amount of AEH research (e.g. dealing with learner and domain models) that can be considered and reused within TEL recommender research. On the other hand, in non-formal settings less expert-demanding approaches need to be explored.

Learning Networks

Another strand of work includes research where the context variables are extracted from the contributions of the users. A category of such systems includes *learning networks*, which connect distributed learners and providers in certain domains [53, 54]. The design and development of learning networks is highly flexible, learner-centric and evolving from the bottom upwards, going beyond formal course

and programme-centric models that are imposed from the top downwards. A learning network is populated with many learners and learning activities provided by different stakeholders. Each user is allowed to add, edit, delete or evaluate learning resources at any time.

The concept of learning networks [54] provides methods and technical infrastructures for distributed lifelong learners to support their personal competence development. It takes advantages of the possibilities of the Web 2.0 developments and describes the new dynamics of learning in the networked knowledge society. A learning network is learner-centered and its development emerges from the bottom-up through the participation of the learners. Emergence is the central idea of the learning network concept. Emergence appears when an interacting system of individual actors and resources self-organises to shape higher-level patterns of behaviour [35, 45, 103].

We can imagine users (e.g. learners) interacting with learning activities in a learning network while their progress is being recorded. Indirect measures like time or learning outcomes, and direct measures like ratings and tags given by users allow identify paths in a learning network which are faster to complete or more attractive than others (e.g. [28, 100]). This information can be fed back to other learners in the learning network, providing collective knowledge of the ‘swarm of learners’ in the learning network. Most learning environments are designed only top-down as often times their structure, learning activities and learning routes are predefined by an educational institution. Learning networks, on the other hand, take advantage of the user-generated content that is created, shared, rated and adjusted by using Web 2.0 technologies. In the field of TEL, several European projects address these bottom-up approaches of creating and sharing knowledge. A large EU initiative that addresses the creation of informal learning networks is the TENcompetence project [110].

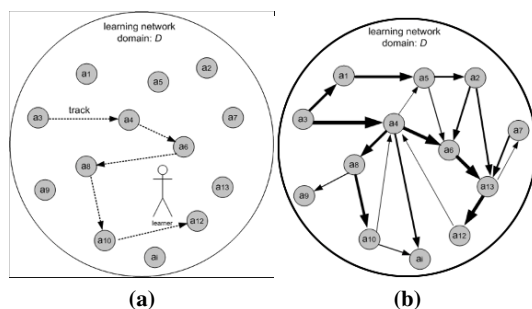


Fig. 12.2: Evolution of a learning network (left: starting phase with a first learner moving through possible learning activities; right: advanced phase showing emergent learning paths from the collective behavior of all learners)

Another category of systems that formulate and define their context variables following a bottom-up approach, are Mash-Up Personal Learning Environments

(MUPPLE) [109]. First such approaches were created in [62, 63, 109]. The iCamp EU-initiative explicitly addresses the integration of Web2.0 sources into MUPPLE, by creating a flexible environment that allows learners to create their own environments for certain learning activities. MUPPLEs are a kind of instance of the learning network concept and therefore share several characteristics with it. They also support informal learning as they require no institutional background and focus on the learner instead of institutional needs like student management or assessments. The learners do not participate in formal courses and neither receive any certification for their competence development. A common problem for MUPPLEs is the amount of data that is gathered already in a short time frame and the unstructured way it is collected. This can make the process of user and domain modelling demanding and unstructured. On the other hand, this is often the case in recommender systems as well, when user and item interactions are explored, e.g. in order to identify user and item similarities.

Similarities and differences

Many of the AEH systems address formal learning (e.g. [3, 20, 57]), have equally fine granulated knowledge domains and can therefore offer personalised recommendations to the learners. They take advantage of technologies like metadata and ontologies to define the relationships, conditions, and dependencies of learning resources and learner models. These systems are mainly used in ‘closed-corpus’ applications [16] where the learning resources can be described by an educational designer through semantic relationships and is therefore a formal learning offer. As mentioned before, in formal educational settings (such as universities) there are usually well-structured formal relationships like predefined learning plans (curriculum) with locations, student/teacher profiles, and accreditation procedures. All this metadata can be used to recommend courses or personalise learning through the adaptation of the learning resources or the learning environment to the students [5]. One interesting direction in this research is the work on adaptive sequencing which takes into account individual characteristics and preferences for sequencing learning resources [47]. In AEH there are many design activities needed before the runtime and also during the maintenance of the learning environment. In addition, the knowledge domains in the learning environment need to be described in detail. These aspects make adaptive sequencing and other adaptive hypermedia techniques rather demanding in TEL recommendation.

Informal learning networks emerge without some highly structured domain model representation. Mining techniques need to be used in order to create some representation of the user or domain model. For instance, prior knowledge in informal learning is a rather diffuse parameter because it relies on information given by the learners without any standardisation. To handle the dynamic and diffuse characteristic of prior knowledge, and to bridge the absence of a knowledge domain model, probabilistic techniques like latent semantic analysis are promising [97]. The ab-

sense of maintenance and structure in informal learning is also called the ‘open corpus problem’. The open corpus problem applies when an unlimited set of documents is given that cannot be manually structured and indexed with domain concepts and metadata from a community [16] and applies to informal learning networks. Therefore, bottom-up recommendation techniques like collaborative filtering are more appropriate because they require nearly no maintenance and improve through the emergent behaviour of the community. Drachsler et al. [25] analysed how various types of collaborative filtering techniques can be used to support learners in informal learning networks. Following their conclusions we have to consider the different environmental conditions of informal learning, such as the lack of maintenance and less formal structured learning objects, in order to provide an appropriated navigation support to recommender systems. Learning networks are mainly structured based on user-generated information and interactions.

Besides the already mentioned differences for prior knowledge in informal learning, there are also differences in the data sets which are derived from environmental conditions. Normally, the numbers of ratings obtained in recommender systems is usually very small compared to the number of ratings that have to be predicted. Effective prediction by ratings based on small amounts is very essential for recommender systems and has an effect on the selection of a specific recommendation technique. Formal learning can rely on regular evaluations of experts or students upon multiple criteria (e.g., pedagogical quality, technical quality, ease of use) [67], but in informal learning environments such evaluation procedures are unstructured and few. Formal learning environments like universities often have integrated evaluation procedures for a regular quality evaluation to report to their funding body. With these integrated evaluation procedures more dense data sets can be expected. As a conclusion, the data sets in informal learning context are characterised by the “Sparsity problem” caused by sparse ratings in the data set. Multi-criteria ratings (see Chapter 24) could be beneficial for informal learning to overcome the “Sparsity problem” of the data sets. These multi-criteria ratings have to be reasonable for the community of lifelong learners. The community could rate learning resources on various levels, such as required prior knowledge level (novice to expert), the presentation style of learning resources, and even the level of attractiveness, because keeping students satisfied and motivated is a vital criteria in informal learning. These explicit rating procedures should be supported with several indirect measures like ‘Amount of learners using the learning resource, ‘Amount of adjustments of a learning resources”, in order to measure how up-to-date the learning resource is.

Informal learning is therefore different from well-structured domains like formal learning. Recommender systems for informal learning could have no official maintenance by an institution, mostly rely on its community, and not contain well-defined metadata structures. Moreover, where formal learning is characteristically top-down designed and develop learning offers (closed-corpus), informal learning offers are emerging from the bottom-up through the communities (open-corpus). Therefore, it will be difficult to transfer and apply recommender systems even from formal to non-formal settings (and vice-versa), since user tasks and recommendation goals are often substantially different.

Table 12.2: Recommendation techniques and their usefulness for TEL by Drachler et al. [24]

<i>Name</i>	<i>Short description</i>	<i>Advantages</i>	<i>Disadvantages</i>	<i>Usefulness for TEL</i>
Collaborative filtering (CF) techniques				
1. User-based CF	Users that rated the same item similarly probably have the same taste. Based on this assumption, this technique recommends unseen items already rated by similar users.	<ul style="list-style-type: none"> - No content analysis - Domain-independent - Quality improves over time - Bottom-up approach - Serendipity 	<ul style="list-style-type: none"> - New user problem - New item problem - Popular taste - Scalability - Sparsity - Cold-start problem 	<ul style="list-style-type: none"> - Benefits from experience - Allocates learners to groups (based on similar ratings)
2. Item-based CF	Focus on items, assuming that items rated similarly are probably similar. It recommends items with highest correlation (based on ratings to the items).	<ul style="list-style-type: none"> - No content analysis - Domain-independent - Quality improves over time - Bottom-up approach - Serendipity 	<ul style="list-style-type: none"> - New item problem - Popular taste - Sparsity - Cold-start problem 	<ul style="list-style-type: none"> - Benefits from experience
3. Stereotypes or demographics CF	Users with similar attributes are matched, then recommends items that are preferred by similar users (based on user data instead of ratings).	<ul style="list-style-type: none"> - No cold-start problem - Domain-independent - Serendipity 	<ul style="list-style-type: none"> - Obtaining information - Insufficient information - Only popular taste - Obtaining metadata information - Maintenance ontology 	<ul style="list-style-type: none"> - Allocates learners to groups - Benefits from experience - Recommendation from the beginning of the RS
Content-based (CB) techniques				
4. Case-based reasoning	Assumes that if a user likes a certain item, s/he will probably also like similar items. Recommends new but similar items.	<ul style="list-style-type: none"> - No content analysis - Domain-independent - Quality improves over time 	<ul style="list-style-type: none"> - New user problem - Overspecialisation - Sparsity - Cold-start problem 	<ul style="list-style-type: none"> - Keeps learner informed about learning goal - Useful for hybrid RS
5. Attribute-based techniques	Recommends items based on the matching of their attributes to the user profile. Attributes could be weighted for their importance to the user.	<ul style="list-style-type: none"> - No cold-start problem - No new user / new item problem - Sensitive to changes of preferences - Can include non-item related features - Can map from user needs to items 	<ul style="list-style-type: none"> - Does not learn - Only works with categories - Ontology modeling and maintenance is required - Overspecialisation 	<ul style="list-style-type: none"> - Useful for hybrid RS - Recommendation from the beginning

A critical assessment of recommender techniques regarding their applicability and usefulness in TEL has taken place by Drachsler et al. [24], and is briefly presented in Table 2. This Table provides an initial overview of advantages and disadvantages of each technique, and reports the envisaged usefulness of each technique for TEL recommenders. Nevertheless, it aims to serve as an initial basis for such a discussion, since a more detailed and elaborate survey of all existing recommendation methods and techniques can take place in the future.

12.4 Survey of TEL Recommender Systems

In the TEL domain a number of recommender systems have been introduced in order to propose learning resources to users. Such systems could potentially play an important educational role, considering the variety of learning resources that are published online and the benefits of collaboration between tutors and learners [81, 82, 59]. The following paragraphs review some recent approaches and provide an assessment of their status of development and evaluation.

One of the first attempts to develop a collaborative filtering system for learning resources has been the Altered Vista system [81, 82, 83]. The aim of this study was to explore how to collect user-provided evaluations of learning resources, and then to propagate them in the form of word-of-mouth recommendations about the qualities of the resources. The team working on Altered Vista explored several relevant issues, such as the design of its interface [82], the development of non-authoritative metadata to store user-provided evaluations [81], the design of the system and the review scheme it uses [83], as well as results from pilot and empirical studies from using the system to recommend to the members of a community both interesting resources and people with similar tastes and beliefs [83, 104].

Another system that has been proposed for the recommendation of learning resources is the RACOFI (Rule-Aplying Collaborative Filtering) Composer system [2, 60, 61]. RACOFI combines two recommendation approaches by integrating a collaborative filtering engine, that works with ratings that users provide for learning resources, with an inference rule engine that is mining association rules between the learning resources and using them for recommendation. RACOFI studies have not yet assessed the pedagogical value of the recommender, nor do they report some evaluation of the system by users. The RACOFI technology is supporting the commercial site inDiscover (<http://www.indiscover.net>) for music tracks recommendation. In addition, other researchers have reported adopting RACOFI's approach in their own systems as well [32].

The QSIA (Questions Sharing and Interactive Assignments) for learning resources sharing, assessing and recommendation has been developed by Rafaeli et al. [78, 79]. This system is used in the context of online communities, in order to harness the social perspective in learning and to promote collaboration, online recommendation, and further formation of learner communities. Instead of developing a typical automated recommender system, Rafaeli et al. chose to base QSIA

Table 12.3: Implemented TEL recommender systems reported in literature

System	Status	Evaluator focus	Evaluation roles
Altered Vista [80, 81, 82, 105]	Full system	Interface, Algorithm, System usage	Human users
RACOFI [2, 61]	Prototype	Algorithm	System designers
QSAI [78, 79]	Full system	—	—
CYCLADES [4]	Full system	Algorithm	System designers
CoFind [29, 30]	Prototype	System usage	Human users
Learning object sequencing [88]	Prototype	System usage	Human users
Evolving e-learning system [90, 91, 92, 93]	Full system	Algorithm, System usage	Simulated users, Human users
ISIS - Hybrid Personalised Recommender System [28]	Prototype	Algorithm, System usage	Human users
Multi-Attribute Recommendation Service [67]	Prototype	Algorithm	System designers
Learning Object Recommendation Model [95]	Design	—	—
RecoSearch [32]	Design	—	—
Simulation environment [72]	Full system	Algorithm	Simulated users
ReMashed [26, 27]	Full system	Algorithm, System usage	Human users
CourseRank [55, 56]	Full system	System usage	Human users
CBR Recommender Interface [33]	Prototype	—	—
APOSDLE Recommendation Service [1]	Prototype	—	—
A2M Recommending System [86]	Prototype	—	—
Moodle Recommender System [44]	Prototype	Algorithm, System usage	Human users
LRLS [41]	Prototype	System usage, Learner performance	Human users
RPL recommender [49]	Prototype	System usage	System designers, Human users

on a mostly user-controlled recommendation process. That is, the user can decide whether to assume control on who advises (friends) or to use a collaborative filtering service. The system has been implemented and used in the context of several learning situations, such as knowledge sharing among faculty and teaching assistants, high school teachers and among students, but no evaluation results have been reported so far [78, 79].

In this strand of systems for collaborative filtering of learning resources, the CYCLADES system [4] has proposed an environment where users search, access, and evaluate (rate) digital resources available in repositories found through the Open Archives Initiative (OAI, <http://www.openarchives.org>). Informally, OAI is an agreement between several digital archives providers in order to offer some minimum level of interoperability between them. Thus, such a system can offer recommendations over resources that are stored in different archives and accessed through an open scheme. The recommendations offered by CYCLADES have been evaluated through a pilot study with about 60 users, which focused on testing the performance (predictive accuracy) of several collaborative filtering algorithms.

A related system is the CoFind prototype [29, 30]. It also used digital resources that are freely available on the Web but it followed a new approach by applying for the first time folksonomies (tags) for recommendations. The CoFind developers stated that predictions according to preferences were inadequate in a learning context and therefore more user driven bottom-up categories like folksonomies are important.

A typical, neighborhood-based set of collaborative filtering algorithms have been tried in order to support learning object recommendation by Manouselis et al. [67]. The innovative aspect of this study is that the engaged algorithms have been multi-attribute ones, allowing the recommendation service to consider multi-dimensional ratings that users provide on learning resources. An interesting outcome from this study in comparison to initial experiments using the same algorithms (e.g. [65]), is that it seems that the performance of the same algorithms is changing, depending on the context where testing takes place. For instance, the results from the comparative study of the same algorithms in an e-commerce [65] and a TEL setting [67] has led to the selection of different algorithms from the same set of candidate ones. This can be an indicator that the performance of recommendation algorithms that have been proved to be performing well in one context (e.g. movie recommendation) should not be expected to do the same in another context (e.g. TEL), an area which requires further experimentation (see Chapter 7).

A different approach to learning resources' recommendation has been followed by Shen and Shen [88]. They have developed a recommender system for learning objects that is based on sequencing rules that help users be guided through the concepts of an ontology of topics. The rules are fired when gaps in the competencies of the learners are identified, and then appropriate resources are proposed to the learners. A pilot study with the students of a Network Education college has taken place, providing feedback regarding the users' opinion about the system.

A similar sequencing system has been introduced by Huang et al. [41]. It uses a Markov chain model to calculate transition probabilities of possible learning ob-

jects in a sequenced course of study. The model is supported by an entropy-based approach for discovering one or more recommended learning paths. A pilot implementation has been deployed and tested in a Taiwanese university, involving about 150 users.

Tang and McCalla proposed an evolving e-learning system, open into new learning resources that may be found online, which includes a hybrid recommendation service [89, 90, 91, 92, 93]. Their system is mainly used for storing and sharing research papers and glossary terms among university students and industry practitioners. Resources are described (tagged) according to their content and technical aspects, but learners also provide feedback about them in the form of ratings. Recommendation takes place both by engaging a Clustering Module (using data clustering techniques to group learners with similar interests) and a Collaborative Filtering Module (using classic collaborative filtering techniques to identify learners with similar interests in each cluster). The authors studied several techniques to enhance the performance of their system, such as the usage of artificial (simulated) learners [93]. They have also performed an evaluation study of the system with real learners [94].

A rather simple recommender system without taking into account any preferences or profile information of the learners was applied by Janssen et al. [44]. However, they conducted a large experiment with a control group and an experimental group. They found positive effects on the effectiveness (completion rates of learning objects) though not on efficiency (time taken to complete the learning resources) for the experimental group as compared to the control group.

Nadolski et al. [72] created a simulation environment for different combination of recommendation algorithms in hybrid recommender system in order to compare them against each other regarding their impact on learners in informal learning networks. They compared various cost intensive ontology based recommendation strategies with light-weight collaborative filtering strategies. Therefore, they created treatment groups for the simulation through combining the recommendation techniques in various ways. Nadolski et al. [72] tested which combination of recommendation techniques in recommendation strategies had a higher effect on the learning outcomes of the learners in a learning network. They concluded that the light-weight collaborative filtering recommendation strategies are not as accurate as the ontology-based strategies but worth-while for informal learning networks when considering the environmental conditions like the lack of maintenance in learning networks. This study confirmed that providing recommendations leads towards more effective, more satisfied, and faster goal achievement than no recommendation. Furthermore, their study reveals that a light-weight collaborative filtering recommendation technique including a rating mechanism is a good alternative to maintain intensive top-down ontology recommendation techniques.

Moreover, the ISIS system adopts a hybrid approach for recommending learning resources is the one recently proposed by Hummel et al. [42]. The authors build upon a previous simulation study by Koper [52] in order to propose a system that combines social-based (using data from other learners) with information-based (using metadata from learner profiles and learning activities) in a hybrid recommender

system. They also designed an experiment with real learners. Drachsler et al. [28] recently reported the experimental results the ISIS experiment. They found a positive significant effect on efficiency (time taken to complete the learning objects) of the learners after a runtime of four months. It is a very good example of a system that is following the latest trends in learning specifications for representing learner profiles and learning activities.

The same group recently developed a recommender system called ReMashed [26, 27] that addresses learners in informal learning networks. They created a mash-up environment that combines sources of users from different Web2.0 services like Flickr, Delicious.com or Sildeshare.com. Again, they applied a hybrid recommender system that takes advantage of the tag and rating data of the combined Web2.0 sources. The tags that are already given to the Web2.0 sources are used for the cold-start of the recommender system (see Chapter 19). The users of ReMashed are able to rate the emerging data of all users in the system. The ratings are used for classic collaborative filtering recommendations based on the Duine prediction engine [98].

A similar approach is followed by the proposed Learning Object Recommendation Model (LORM) that also follows a hybrid recommendation algorithmic approach and that describes resources upon multiple attributes, but has not yet reported to be implemented in an actual system [95].

Another hybrid recommendation approach has been adopted in the CourseRank system (<https://courserank.stanford.edu/CourseRank/main>) that is used as an unofficial course guide for Stanford University students. In this system, the recommendation process is viewed under the prism of querying a relational database with course and student information [55]. To this end, a number of tuple operators have been defined in order to allow the system to provide flexible recommendations to its users. The system has been first deployed in September 2007, attracting lots of interest from its users: it has been reported that more than 70% of the Stanford students use it [56].

A hybrid approach is also adopted by the prototype system that has been implemented in the course repository of the Virtual University of Tunis (RPL platform, <http://cours.uvt.rnu.tn/rpl/>). This prototype includes a recommendation engine that combines a collaborative filtering algorithm with a content-based filtering algorithm, using data that has been logged and mined from user actions. The usage logs of the RPL platform are used for this purpose, and a preliminary evaluation experiment has already taken place [49].

Finally, there have been some recent proposals for systems or algorithms that could be used to support recommendation of learning resources. These include a variety of work-in-progress systems, such as a case-based reasoning recommender that Gomez-Albarran and Jimenez-Diaz [33], the contextual recommendations that the knowledge-sharing environment of the APOSDLE EU-project (<http://www.aposdle.tugraz.at>) offers to the employees of large organisations [1], and the A2M prototype [86]. Recommendation of multimedia learning resource onto mobile devices such as cell phones and PDAs have been explored in [51].

Nevertheless, despite the increasing number of systems proposed for recommending learning resources, a closer look to the current status of their development

and evaluation reveals the lack of systematic evaluation studies in the context of real-life applications. As Table 3 indicates:

- More than half of the proposed systems (12 out of 20) still remain at a design or prototyping stage of development;
- Only 10 systems have been reported to be evaluated through trials that involved human users.

Another interesting observation is that, very often, experimental investigation of the recommendation algorithms does not take place. This is a common evaluation practice in systems examined for other domains (e.g. [7, 22, 37, 74]), since careful testing and parameterisation of the algorithms has to be carried out before a recommender system is finally deployed in a real setting (see Chapters 8 and 10). One of the main reasons is that the performance of recommendation algorithms seems to be dependent on the particularities of the application context, therefore, it is advised to experimentally analyse various design choices for a recommender system, before its actual deployment.

12.5 Evaluation of TEL Recommenders

Worthen et al. [111] define evaluation as the “identification, clarification, and application of defensible criteria to determine an evaluation object’s value, quality, utility, effectiveness, or significance in relation to those criteria”. An evaluation of an interactive system ensures that it behaves as expected by the designer and that it meets the requirements of the user [23]. As far as recommender systems in general, and TEL recommenders in particular are concerned, evaluation becomes a critical point at the systems lifecycle for its improvement and success. Until today, evaluation of recommender systems gives emphasis to rather “technical” measures coming from information retrieval research, although the importance of including user-related evaluation methods has been highlighted (see [38, 69] and Chapter 8). In TEL recommender systems evaluation becomes an even more demanding task, considering the particularities of the educational contexts. To this end, we try to provide a first overview of relevant evaluation requirements, adopting the different perspectives covered in this chapter.

Evaluating the different components

The evaluation of AEH systems has generally been considered to be challenging, due to a number of issues that can generally be categorised under two types [108]:

- First, adequately defining the reference variables against which the adaptivity of the system will be evaluated is difficult for those systems that either cannot

switch off the adaptivity, or where a non-adaptive version appears to be absurd because adaptivity is an inherent feature of these systems [40]. In TEL recommenders, this concerns defining the variables that can successfully measure if switching off the recommendation in a TEL system actually affects its perceived usefulness.

- Second, adequately defined criteria for the success of adaptivity are not well defined or there are rarely commonly accepted criteria: on the one hand, objective standard criteria (e.g. duration, number of interaction steps, knowledge gain) regularly failed to find a difference between adaptive and non-adaptive versions of a system. On the other hand, subjective criteria that are standard in human-computer interaction research (e.g. usability questionnaires) have been rarely applied to measure the success of adaptive systems. In TEL, the issue is related to the definition of appropriate evaluation methods (e.g. techniques, metrics and instruments) to measure the success of a successful recommendation strategy in comparison to a non-successful one.

A common problem arising in such evaluation efforts is when treating the adaptation process as a “monolithic” entity and trying to assess it as a whole [11]. This cannot provide results at a level of granularity that can be of practical use and help the system designer to decide which part of the system needs improvement (e.g. the user modelling approach, the domain modelling approach, the recommendation technique). An interesting approach has been proposed by Brusilovsky et al. [13]: to decompose the adaptation process into two layers that are evaluated separately. The main idea behind the approach was that the evaluation of adaptive systems should not treat adaptation as a “monolithic”/singular process happening behind the scenes. Rather, adaptation should be “broken down” into its constituents, and each of these constituents should be evaluated separately where necessary and feasible [46]. The seeds of this idea can be traced back to Totterdell and Boyle [94] who propose that a number of adaptation metrics could be related to different components of a logical model of adaptive user interfaces, to provide what amounts to adaptation-oriented design feedback. Furthermore, Totterdell and Boyle present two types of assessment performed to validate what is termed “success of the user model” (note that, in their case, the “user model” is also responsible for adaptation decision making):

“Two types of assessment were made of the user model: an assessment of the accuracy of the model’s inferences about user difficulties; and an assessment of the effectiveness of the changes made at the interface.” [94]

Simultaneously with the idea of evaluating adaptation at two different layers [13], two other *layered* (also referred to as *modular*) evaluation frameworks have been proposed. The process-based framework presented by Weibelzahl [106] consisted from four layers that referred to the information processing steps within the adaptation process: evaluation of input data, evaluation of the inference mechanism, evaluation of the adaptation decision, and evaluation of the total interaction. A second framework has been presented by Paramythis et al. [75] and is more detailed in terms of different components involved in the adaptation process. It also addressed

the question of methods and tools appropriate for the evaluation of different adaptation “modules” to yield input for the development process. A merged version of the two frameworks was finally proposed, identifying both criteria to be taken into consideration in the evaluation of an adaptive system, and the methods and tools that can be engaged to do so [109]. This modular evaluation approach has been explored by several studies that evaluate adaptive systems (e.g. [15]), but has not been yet formally developed and applied for recommender systems. It therefore still needs to be validated before applying it into TEL recommender systems. Nevertheless, we believe that this approach can be incorporated in the rich variety of perspectives and measures to be considered when evaluating TEL recommenders. In the following, an initial elaboration on relevant issues is carried out.

Issues to consider for evaluating TEL recommenders

In the world of consumer recommender systems, several data sets with specific characteristics are available (e.g. the MovieLens data set, the Book-Crossing data sets, or the Jester data set). These data sets are used as a common standard or benchmark to evaluate new recommendation algorithms. Furthermore, consumer product recommendation algorithms are evaluated based on common technical measures like accuracy, coverage, and performance in terms of execution time.

In the application domain of TEL, to evaluate pedagogy driven recommender systems for formal or informal learning, no standardised data sets nor standardised evaluation procedures are available. Moreover, focusing only on technical measures for recommender systems in TEL, without considering the actual needs and characteristics of the learners, is questionable. Thus, further evaluation procedures that complement the technical evaluation approaches are needed (see Chapter 8). For example, learners only benefit from TEL supported and enhanced systems when they make the learning more effective, efficient, and/or more attractive. Common measures to evaluate the success of such systems in educational settings thus include *Effectiveness*, *Efficiency*, *Satisfaction* and the *Drop-out rate*. *Effectiveness* is a sign of the total amount of completed, visited or studied content objects during a learning phase. *Efficiency* indicates the time that learners need to reach their learning goal. It is related to the effectiveness variable through counting the actual study time. *Satisfaction* reflects the individual satisfaction of the learners with the given recommendations. Satisfaction is close to the motivation of a learner and therefore an important measure for learning. Finally, the *Drop-out rate* mirrors the numbers of learners that drop out during the learning phase. In educational research the drop-out rate is an important measure when the aim is to graduate as many learners as possible during a learning phase. As far as learning networks are concerned, methods and metrics originating from Social Network Analysis (SNA) (e.g. [105]) could also be engaged to measure the success of TEL recommenders. The SNA measures can be used to estimate the benefits coming from the contributions of the learners for the network as a whole. These are more specific measures that are mainly re-

lated to informal learning networks. SNA gives various insights into the different roles learners can have in a learning network. Typical SNA measures are *Variety*, *Centrality*, *Closeness* and *Cohesion*. *Variety* measures the level of emergence in a learning network through the combination of individual learning paths to the most successful learning routes. *Centrality* is an indicator for the connectivity of a learner in a learning network. It counts the number of ties to other learners in the network. *Closeness* measures the degree a learner is close to all other learners in a network. It represents the ability to access information direct or indirect through the connection to other network members. *Cohesion*, on the other hand, indicates how strongly learners are directly connected to each other by cohesive bonds. Peer-groups of learners can be identified if every learner is directly tied to every other learner in the learning network. Drachler et al. [24] followed this approach by using simulations to evaluate the impacts of a recommender system for learners in informal learning networks (see Chapter 18).

Synthesising all the various components into an overall evaluation framework has several methodological and practical difficulties. As a general guideline, however, classical evaluation frameworks from educational research could be adopted and adapted to the recommender systems' context. As an example, we illustrate how the Kirckpatrick's model [50], which measures the success of training using four different layers, could be used to evaluate the success of a recommender system in a TEL context:

1. **Reaction** of user - what they thought and felt (*"Did I enjoy the recommendations I receive?"*);
2. **Learning** - the resulting increase in gaining new knowledge or capabilities (*"Did I learn what I needed to and get some new ideas, with the help of the recommender?"*);
3. **Behaviour** - extent of how acquired knowledge and capability can be implemented/applied in real life (*"Will I use the new information and ideas I was recommended?"*);
4. **Results** - the effects on the user's performance in the learning or working environment (*"Do the ideas and information I was recommended improve my effectiveness and results?"*).

Therefore, the definition of an overall evaluation framework of TEL recommenders could include:

- A detailed analysis of the evaluation methods and tools that can be employed for evaluating TEL recommendation techniques against a set of criteria that will be proposed for each of the selected components (e.g. user model, domain model, recommendation strategy and algorithm). For the presented example of the Kirckpatrick's dimensions, this would include an identification of the evaluation methods that could be engaged to measure the effect of the recommender in a particular TEL context, upon each one of the four dimensions.
- The specification of evaluation metrics/indicators to measure the success of each component (e.g. evaluating accuracy of the recommendation algorithm,

evaluating coverage of the domain model). For the presented example, this would include a specification of the particular metrics that can measure the effect of introducing the recommender in this TEL context.

- The elaboration of a number of methods and instruments that can be engaged in TEL settings, in order to collect evaluation data from engaged stakeholders, explicitly or implicitly, e.g. measuring user satisfaction, assessing impact of engaging the TEL recommender from improvements in working tasks. For the presented example, this would include the proposal of specific instruments that can be used to measure each one of the metrics that measure the effect of introducing the recommender in this TEL context.

12.6 Conclusions and further work

This chapter provides an introduction to the issues related to the deployment of recommender systems in the Technology Enhanced Learning (TEL) settings emphasising the particularities of this application domain. It first discussed the context in which TEL recommenders are deployed, and reflected on related user tasks and recommendation goals. A review of related work coming from the research strands of Adaptive Educational Hypermedia and Learning Networks has been provided, with a particular emphasis on how it applies to TEL recommenders for formal and informal settings. Then, a survey of TEL recommenders proposed in the literature was presented with a critical view on the actual implementation of systems. Particular emphasis was given to the evaluation leading to a discussion on evaluation requirements and issues for TEL recommender systems. To our knowledge, this is the first study attempting to systematically cover the design and deployment of recommender systems in the TEL settings (see Chapter 11). Nevertheless, it can only provide a brief overview of related issues, leaving several aspects to be further explored and researched.

As indicated in the previous section, one of the main research challenges related to the introduction of recommender systems in TEL is how to perform a systematic development and evaluation of such systems and their effect. To this end, a systematic analysis of TEL-related tasks that can be supported by recommender systems took place, in order to identify the particular requirements that need to be considered. Furthermore, the development of concrete evaluation frameworks that will follow a layered approach is an open issue. These frameworks can focus on incorporating as many evaluation dimensions as possible, also addressing pedagogical dimensions, by combining a variety of evaluation methods, metric, and instruments.

In addition, for the various groups of researchers involved in TEL, a number of topics are of high research interest. For example, the recommendation support for learners in formal and informal learning that takes advantage of contextualised recommender systems has become an important one. These recommender systems, also called context-aware recommender systems (see [61] and Chapter 7), use for example geographical location of a user to recommend relevant resources. Such contextu-

alisation becomes important, for example, when multilingual educational resources are recommended from a number of repositories residing in different countries and complying with various curricula requirements [101]. Additionally, context awareness could include pedagogical aspects like prior knowledge, learning goals or study time to embed pedagogical reasoning into collaborative filtering driven recommendations.

Another promising approach is the use of multi-criteria input for recommender system in TEL (see Chapter 24). Users (learners and teachers) can not only rate learning resource based on the level of complexity, curriculum alignment or how much time is required to cover the learning material, but input could also be inferred from different implicit sources. Such multidimensional input can potentially have a high impact on the suitability of recommendations. A related problem is the lack of TEL specific rated data sets for informal and formal learning. Different to the recommender system world, where many data sets are available (e.g. MovieLens, BookCrossing, Jester Collaborative Filtering Dataset), the TEL community is still working with rather small home-made data sets, which are rarely public available [66].

Acknowledgements

Research of N. Manouselis was funded with support by the European Commission and more specifically, the project ECP-2006-EDU-410012 ‘Organic.Edunet: A Multilingual Federation of Learning Repositories with Quality Content for the Awareness and Education of European Youth about Organic Agriculture and Agroecology’ of the *eContentplus* Programme. Research of H. Drachler was funded with support by the European Commission and more specifically, the project IST 027087 ‘TEN-Competence’ of the FP6 Programme. Riina Vuorikari thanks the HS-säätiö for the stipend.

References

1. Aehnelt M., Ebert M., Beham G., Lindstaedt S., Paschen A.: A Socio-technical Approach towards Supporting Intra-organizational Collaboration. In: Dillenbourg P. and Sprechtl M., (eds.) *Times of convergence: Technologies across learning contexts. Proceedings of the 3rd European Conference on Technology Enhanced Learning (EC-TEL 2008)*, Maastricht, The Netherlands, LNCS 5192, pp. 33-38, Berlin: Springer (2008).
2. Anderson M., Ball M., Boley H., Greene S., Howse N., Lemire D., McGrath S.: RACOFI: A Rule-Applying Collaborative Filtering System. Paper presented at the conference IEEE/WIC COLA 2003, 13 October, Halifax, Canada.
3. Aroyo, L., Mizoguchi, R., and Tzolov, C.: OntoAIMS- Ontological Approach to Courseware Authoring. Paper presented at the International Conference on Computers in Education (ICCE 2003), 2-5 December, Hong Kong, China

4. Avancini H., Straccia U.: User recommendation for collaborative and personalised digital archives, *International Journal of Web Based Communities* **1**(2), 163–175 (2005).
5. Baldoni, M., Baroglio, C., Brunkhorst, I., Marengo, E., Patti, V.: Reasoning-Based Curriculum Sequencing and Validation: Integration in a Service-Oriented Architecture. In *Proceedings of the 2nd European Conference on Technology Enhanced Learning (EC-TEL 2007)*, LNCS 4753, pp. 426. Berlin: Springer (2007).
6. Baudisch P.: *Dynamic Information Filtering*. PhD Thesis, GMD Forschungszentrum Informationstechnik GmbH, Sankt Augustin (2001).
7. Breese, J. S., Heckerman, D., and Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Cooper G. F. and Moral S., (eds.) *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pp. 43-52, Morgan-Kaufmann, Californian, San Francisco (1998).
8. Brockett, R. G., & Hiemstra, R.: *Self-direction in adult learning: perspectives on theory, research, and practice*. London: Routledge (1991).
9. Brusilovsky, P.: Methods and techniques of adaptive hypermedia, *User Modeling and User-Adapted Interaction* **6**(2-3), 87–129 (1996).
10. Brusilovsky, P.: Adaptive hypermedia. *User Modeling and User-Adapted Interaction* **11**(1-2), 87–110 (2001).
11. Brusilovsky, P. and Eklund, J.: A study of user-model based link annotation in educational hypermedia, *Journal of Universal Computer Science* **4**(4), 429–448 (1998).
12. Brusilovsky P., Nejd W.: *Adaptive Hypermedia and Adaptive Web*. Practical Handbook of Internet Computing, Chapman & Hall / CRC Press LLC (2005).
13. Brusilovsky, P., Karagiannidis, C., and Sampson, D. G.: The benefits of layered evaluation of adaptive applications and services. In: Weibelzahl, S., Chin, D. N., and Weber, G. (eds.) *Empirical Evaluation of Adaptive Systems*. Proceedings of 8th International Conference on User Modeling, (UM2001), pp 1-8, Berlin: Springer (2001).
14. Brusilovsky, P., Pesin, L., & Zyryanov, M.: Towards an adaptive hypermedia component for an intelligent learning environment. In: Bass, L.J., Gornostaev, J., & Unger, C. (eds.) *Human-Computer Interaction (LNCS 753)*. pp. 348-358, Berlin: Springer (1993).
15. Brusilovsky P., Karagiannidis C., Sampson D., Layered evaluation of adaptive learning systems. *International Journal of Continuing Engineering Education and Lifelong Learning* **14**(4/5), 402–421 (2004).
16. Brusilovsky, P., & Henze, N.: Open Corpus Adaptive Educational Hypermedia. In P. Brusilovsky, A. Kobsa & W. Nejd (eds.) *The Adaptive Web: Methods and Strategies of Web Personalization*. (LNCS 4321), pp. 671-696 Berlin: Springer (2007).
17. Colley, H., Hodgkinson, P., & Malcolm, J.: Non-formal learning: mapping the conceptual terrain. A consultation report. (2008). Accessed 11 January 2010.
18. Cristea, A.: Authoring of Adaptive Hypermedia. *Educational Technology & Society* **8**(3), 6–8 (2005).
19. De Bra, P. M. E.: Teaching Hypertext and Hypermedia through the Web. *Journal of Universal Computer Science* **2**(12), 797–804 (1996).
20. De Bra, P., Aerts, A., Smits, D., & Stash, N.: *AHA! Version 2.0, More Adaptation Flexibility for Authors*. Paper presented at the World Conference on e-Learning in Corporate, Government, Healthcare & Higher Education. 15-19 October 2002, Montreal, Canada (2002).
21. De La Passardièrre, B., & Dufresne, A.: Adaptive navigational tools for educational hypermedia. In: I. Tomek (ed.), *Computer Assisted Learning*. Proceedings of the 4th International Conference on Computers and Learning (ICCAL'92), LNCS 602, pp. 555-567, Berlin: Springer (1992).
22. Deshpande, M., & Karypis, G.: Selective Markov models for predicting Web page accesses. *Transactions on Internet Technology*. **4**(2), 163–184 (2004).
23. Dix, A. J., Finlay, J. E., Abowd, G. D., and Beale, R.: *Human-Computer Interaction*. Harlow, England: Prentice Hall (1998).

24. Drachsler H., Hummel H. G. K., Koper R.: Personal recommender systems for learners in lifelong learning: requirements, techniques and model. *International Journal of Learning Technology* 3(4), 404–423 (2008a).
25. Drachsler, H., Hummel, H.G.K., Koper, R.: Using Simulations to Evaluate the Effects of Recommender Systems for Learners in Informal Learning Networks. In: Vuorikari, R., Kieslinger, B., Klamma, R., Duval, E. (eds.): SIRTTEL workshop at the 3rd EC-TEL conference. CEUR Workshop Proceedings VOL-382, Maastricht, The Netherlands (2008b).
26. Drachsler, H., Pecceu, D., Arts, T., Hutten, E., Rutledge, L., Van Rosmalen, P., Hummel, H.G.K., Koper, R.: ReMashed-Recommendations for Mash-Up Personal Learning Environments. In: Cress, U., Dimitrova, V., Specht, M. (eds.): Learning in the Synergy of Multiple Disciplines, Proceedings of the 4th European Conference on Technology Enhanced Learning (EC-TEL 2009), LNCS 5794, pp. 788–793, Berlin: Springer (2009a).
27. Drachsler, H., Pecceu, D., Arts, T., Hutten, E., Rutledge, L., Van Rosmalen, P., Hummel, H.G.K., Koper, R.: ReMashed-An Usability Study of a Recommender System for Mash-Ups for Learning. 1st Workshop on Mashups for Learning at the International Conference on Interactive Computer Aided Learning, Villach, Austria (2009b).
28. Drachsler, H., Hummel, H. G. K., Van den Berg, B., Eshuis, J., Berlanga, A., Nadolski, R., Waterink, W., Boers, N., & Koper, R.: Effects of the ISIS Recommender System for navigation support in self-organized learning networks. *Journal of Educational Technology and Society* 12, 122–135 (2009c).
29. Dron, J., Mitchell, R., Boyne, C., & Siviter, P.: CoFIND: steps towards a self-organising learning environment. Proceedings of the World Conference on the WWW and Internet (WebNet 2000), San Antonio, Texas, USA, October 30–November 4, pp. 146–151, USA AACE (2000a).
30. Dron, J., Mitchell, R., Siviter, P., & Boyne, C.: CoFIND—an experiment in n-dimensional collaborative filtering. *Journal of Network and Computer Applications* 23(2), 131–142 (2000b).
31. Duval E., Vuorikari R., Manouselis N. (Eds.), Special Issue on Social Information Retrieval in Technology Enhanced Learning, *Journal of Digital Information (JoDI)*, Vol. 10, (2009).
32. Fiaidhi J.: RecoSearch: A Model for Collaboratively Filtering Java Learning Objects. *International Journal of Instructional Technology and Distance Learning* 1(7), 35–50 (2004).
33. Gomez-Albarran M., Jimenez-Diaz G.: Recommendation and Students' Authoring in Repositories of Learning Objects: A Case-Based Reasoning Approach. *International Journal of Emerging Technologies in Learning (iJET)* 4(1), 35–40 (2009).
34. Gonschorek, M., & Herzog, C.: Using hypertext for an adaptive helpsystem in an intelligent tutoring system. In: J. Greer (Ed.), *Artificial Intelligence in Education, Proceedings of the 7th World Conference on Artificial Intelligence in Education, (AI-ED'95)*, 16–19 August, Washington, DC, AACE, pp. 274 (1995).
35. Gordon, D.: *Ants at Work: How an Insect Society is Organized*. Free Press, New York (1999).
36. Graham, A.: Blended learning systems: definitions, current trends and future directions. In: Bonk, C. J. & Graham, C. R. (eds.) *Handbook of blended learning: Global Perspectives, local designs*. San Francisco, CA: Pfeiffer Publishing, pp. 3–21 (2005).
37. Herlocker, J., Konstan, J., Riedl, J.: An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information Retrieval* 5(4), 287–310 (2002).
38. Herlocker, J.L., Konstan, J.A., Terveen, L.G., & Riedl, J.T.: Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems* 22(1), 5–53 (2004).
39. Hohl, H., Böcker, H.-D., Gunzenhäuser, R.: Hypadapter: An adaptive hypertext system for exploratory learning and programming. *User Modeling and User-Adapted Interaction* 6(2–3), 131–156 (1996).
40. Höök, K.: Steps to take before intelligent user interfaces become real. *Interacting With Computers* 12(4), 409–426 (2000).

41. Huang, Y.-M., Huang, T.-C., Wang, K.-T., Hwang, W.-Y.: A Markov-based Recommendation Model for Exploring the Transfer of Learning on the Web. *Educational Technology & Society* **12**(2), 144–162 (2009).
42. Hummel, H.G.K., Van den Berg, B., Berlanga, A.J., Drachler, H., Janssen, J., Nadolski, R.J., Koper, E.J.R.: Combining Social- and Information-based Approaches for Personalised Recommendation on Sequencing Learning Activities. *International Journal of Learning Technology* **3**(2), 152–168 (2007).
43. Jameson, A.: *Systems That Adapt to Their Users: An Integrative Perspective*. Saarbrücken: Saarland University (2001).
44. Janssen, J., Tattersall, C., Waterink, W., Van den Berg, B., Van Es, R., Bolman, C., et al.: Self-organising navigational support in lifelong learning: how predecessors can lead the way. *Computers & Education* **49**, 781–793 (2005).
45. Johnson, S.: *Emergence*. Scribner, New York (2001).
46. Karagiannidis, C., Sampson, D. G.: Layered evaluation of adaptive applications and services. In: Brusilovsky, P. and Stock, C. S. O. (Eds.), *Proc. of International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, AH2000, Trento, Italy*, pp. 343–346. Berlin: Springer (2000).
47. Karampiperis, P., Sampson, D.: Adaptive Learning Resources Sequencing in Educational Hypermedia Systems. *Educational Technology & Society* **8**(4), 128–147 (2005).
48. Kay, J., Kummerfeld, R. J.: An individualised course for the C programming language. In: *Proceedings of Second International WWW Conference*. Chicago, USA (1994).
49. Khribi, M.K., Jemni, M., Nasraoui, O.: Automatic Recommendations for E-Learning Personalization Based on Web Usage Mining Techniques and Information Retrieval. *Educational Technology & Society* **12**(4), 30–42 (2009).
50. Kirkpatrick, D.L.: *Evaluating Training Programs* (2nd ed.). Berrett Koehler, San Francisco (1959).
51. Klamma, R., Spaniol, M., Cao, Y.: Community Aware Content Adaptation for Mobile Technology Enhanced Learning. In: *Innovative Approaches for Learning and Knowledge Sharing*, pp. 227–241 (2006).
52. Koper, R.: Increasing Learner Retention in a Simulated learning network using Indirect Social Interaction. *Journal of Artificial Societies and Social Simulation*, **8**(2) (2005).
53. Koper, E.J.R., Tattersall, C.: New directions for lifelong learning using network technologies. *British Journal of Educational Technology* **35**(6), 689–700 (2004).
54. Koper, R., Rusman, E., & Sloep, P.: *Effective Learning Networks*. Lifelong Learning in Europe **1**, 18–27 (2005).
55. Koutrika, G., Ikeda, R., Bercovitz, B., Garcia-Molina, H.: Flexible Recommendations over Rich Data. In: *Proc. of the 2nd ACM International Conference on Recommender Systems (Rec-Sys'08)*. Lausanne, Switzerland, (2008).
56. Koutrika, G., Bercovitz, B., Kaliszan, F., Liou, H., Garcia-Molina, H.: CourseRank: A Closed-Community Social System Through the Magnifying Glass. In: *Proc. of the 3rd International AAAI Conference on Weblogs and Social Media (ICWSM'09)*. San Jose, California (2009).
57. Kravcik, M., Specht, M., Oppermann, R.: Evaluation of WINDS Authoring Environment. In: P. De Bra & W. Nejdl (Eds.), *Adaptive Hypermedia and Adaptive Web-Based Systems, LNCS 3137*, 166–175. Berlin: Springer (2004).
58. Krulwich, B.: Lifestyle Finder: Intelligent User Profiling Using Large-Scale Demographic Data. *Artificial Intelligence Magazine* **18**(2), 37–45 (1997).
59. Kumar, V., Nesbit, J., Han, K.: Rating Learning Object Quality with Distributed Bayesian Belief Networks: The Why and the How. In: *Proc. of the Fifth IEEE International Conference on Advanced Learning Technologies, ICALT'05* (2005).
60. Lemire, D.: Scale and Translation Invariant Collaborative Filtering Systems. *Journal of Information Retrieval* **8**(1), 129–150 (2005).

61. Lemire, D., Boley, H., McGrath, S., Ball, M. (2005). Collaborative Filtering and Inference Rules for Context-Aware Learning Object Recommendation. *International Journal of Interactive Technology and Smart Education* **2**(3) (2005).
62. Liber, O.: Colloquia - a conversation manager. *Campus-Wide Information Systems* **17**, 56–61 (2000).
63. Liber, O., Johnson, M. (2008). Personal Learning Environments. *Interactive Learning Environments* **16**, 1–2 (2008).
64. Longworth, N.: *Lifelong learning in action - Transforming education in the 21st century*. Kogan Page, London (2003).
65. Manouselis, N., Costopoulou, C.: Experimental Analysis of Design Choices in Multi-Attribute Utility Collaborative Filtering. *International Journal of Pattern Recognition and Artificial Intelligence, Special Issue on Personalization Techniques for Recommender Systems and Intelligent User Interfaces* **21**(2), 311–331 (2007).
66. Manouselis, N., Vuorikari, R.: What if annotations were reusable: a preliminary discussion. In: *Proc. of the 8th International Conference on Web-based Learning (ICWL 2009)*. Aachen, Germany (2009).
67. Manouselis, N., Vuorikari, R., Van Assche, F.: Simulated Analysis of MAUT Collaborative Filtering for Learning Object Recommendation. In: *Proc. of the Workshop on Social Information Retrieval in Technology Enhanced Learning (SIRTEL 2007)*. Crete, Greece (2007).
68. McCalla, G.: The Ecological Approach to the Design of E-Learning Environments: Purpose-based Capture and Use of Information About Learners. *Journal of Interactive Media in Education, Special Issue on the Educational Semantic Web*, **7**, ISSN:1365-893X (2004).
69. McNee, S.: *Meeting User Information Needs in Recommender Systems*. Doctoral dissertation, University of Minnesota-Twin Cities, Minneapolis, MN, USA (2006).
70. Moore, J.D., Swartout, W.R.: Pointing: A way toward explanation dialogue. In: *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 457–464. AAAI (1990).
71. Moore, M.G., Anderson, W.G.: *Handbook of distance education*. Lawrence Erlbaum, Mahwah N.J. (2004).
72. Nadolski, R.J., Van den Berg, B., Berlanga, A., Drachsler, H., Hummel, H., Koper, R., Sloep, P.: Simulating Light-Weight Personalised Recommender Systems in Learning Networks: A Case for Pedagogy-Oriented and Rating-Based Hybrid Recommendation Strategies. *Journal of Artificial Societies and Social Simulation (JASSS)*, **12**(14) (2009).
73. Oppermann, R.: Adaptively supported adaptability. *Int. J. Hum.-Comput. Stud.* **40**(3), 455–472 (1994).
74. Papagelis, M., Plexousakis, D., Kutsuras, T.: Alleviating the Sparsity Problem of Collaborative Filtering Using Trust Inferences. In: *Proceedings of the 3rd International Conference on Trust Management*, pp. 224–239. Springer (2005).
75. Paramythis, A., Totter, A., Stephanidis, C.: A modular approach to the evaluation of adaptive user interfaces. In Weibelzahl, S., Chin, D., Weber, G. (eds.) *Empirical Evaluation of Adaptive Systems. Proceedings of workshop at the Eighth International Conference on User Modeling, UM2001*, pp. 9–24. Freiburg (2001).
76. Paris, C.: Tailoring object description to a user's level of expertise. *Computational Linguistics* **14**(3), 64–78 (1988).
77. Pérez, T., Gutiérrez, J., Lopistéguy, P.: An adaptive hypermedia system. In: *Proceedings of AI-ED'95, 7th World Conference on Artificial Intelligence in Education*, pp. 351–358. AACE (1995).
78. Rafaeli, S., Barak, M., Dan-Gur, Y., Toch, E.: QSIA-a Web-based environment for learning, assessing and knowledge sharing in communities. *Computers, Education* **43**(3), 273–289 (2004).
79. Rafaeli, S., Dan-Gur, Y., Barak, M.: Social Recommender Systems: Recommendations in Support of E-Learning. *International Journal of Distance Education Technologies* **3**(2), 29–45 (2005).
80. Recker, M.M., Walker, A.: Supporting “Word-of-Mouth” Social Networks through Collaborative Information Filtering. *Journal of Interactive Learning Research* **14**(1), 79–99 (2003).

81. Recker, M.M., Wiley, D.A.: A non-authoritative educational metadata ontology for filtering and recommending learning objects. *Interactive learning environments* 9(3), 255–271 (2001).
82. Recker, M.M., Walker, A., Wiley, D.: An interface for collaborative filtering of educational resources. In: *International Conference on Artificial Intelligence*, pp. 26–29. Las Vegas, Nevada, USA (2000).
83. Recker, M.M., Walker, A., Lawless, K.: What do you recommend? Implementation and analyses of collaborative information filtering of web resources for education. *Instructional Science* 31(4/5), 299–316 (2003).
84. Rokach, L., Maimon, O., Averbuch, M., *Information Retrieval System for Medical Narrative Reports*, Lecture Notes in Artificial intelligence 3055, page 217–228 Springer-Verlag (2004)
85. Rokach, L., Maimon, O., Arbel, R., Selective voting-getting more for less in sensor fusion, *International Journal of Pattern Recognition and Artificial Intelligence* 20 (3), pp. 329–350 (2006)
86. Santos, O.C.: A recommender system to provide adaptive and inclusive standard-based support along the eLearning life cycle. In: *Proceedings of the 2008 ACM conference on Recommender systems*, pp. 319–322 (2008).
87. Schneider-Hufschmidt, M., Kuhme, T., Malinowski, U.: *Adaptive User Interfaces: Principles and Practice*. Elsevier Science Inc (1993).
88. Shen, L., Shen, R.: Learning content recommendation service based-on simple sequencing specification. In: Liu W et al. (eds), pp. 363–370. *Lecture notes in computer science* (2004).
89. Tang T., McCalla G.: Smart Recommendation for an Evolving E-Learning System. In: *Proc. of the Workshop on Technologies for Electronic Documents for Supporting Learning, International Conference on Artificial Intelligence in Education (AIED 2003)* (2003).
90. Tang, T., McCalla, G.: Utilizing Artificial Learner on the Cold-Start Pedagogical-Value based Paper Recommendation. In: *Proc. of AH 2004: International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems* (2004a).
91. Tang, T., McCalla, G.: Beyond Learners' Interest: Personalized Paper Recommendation Based on Their Pedagogical Features for an e-Learning System. In: *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2004)*, pp. 301–310 (2004b).
92. Tang, T.Y., McCalla, G.: On the pedagogically guided paper recommendation for an evolving web-based learning system. In: *Proceedings of the 17th International FLAIRS Conference*, pp. 86–91 (2004c).
93. Tang, T., McCalla, G.: Smart Recommendation for an Evolving E-Learning System: Architecture and Experiment. *International Journal on E-Learning* 4(1), 105–129 (2005).
94. Totterdell, P., Boyle, E.: The evaluation of adaptive systems. *Adaptive User Interfaces* 161–194 (1990).
95. Tsai, K.H., Chiu T.K., Lee M.C., Wang T.I.: A learning objects recommendation model based on the preference and ontological approaches. In: *Proc. of 6th International Conference on Advanced Learning Technologies (ICALT'06)*. IEEE Computer Society Press (2006).
96. Tzikopoulos, A., Manouselis, N., Vuorikari, R.: An overview of Learning Object Repositories. In: *Learning Objects for Instruction, Design and Evaluation*, pp. 29–55. Idea Group Publishing, Hershey, PA (2007).
97. Van Bruggen, J., Sloep, P., van Rosmalen, P., Brouns, F., Vogten, H., Koper, R., Tattersall, C.: Latent semantic analysis as a tool for learner positioning in learning networks for lifelong learning. *British Journal of Educational Technology* 35(6), 729–738 (2004).
98. Van Setten, M.: Supporting people in finding information: hybrid recommender systems and goal-based structuring. *Telematica Instituut Fundamental Research Series NO. 016 (TI/FRS/016)*. Enschede, The Netherlands (2005).
99. Vuorikari, R., Berendt, B.: Study on contexts in tracking usage and attention metadata in multilingual Technology Enhanced Learning. In: Fischer, S., Maehle, E., Reischuk, R. (eds.) *Im Focus das Leben*, pp. 181, 1654–1663. *Lecture Notes in Informatics* (2009).

100. Vuorikari, R., Koper, R.: Ecology of social search for learning resources. *Campus-Wide Information Systems* **26**(4), 272–286 (2009).
101. Vuorikari, R., Ochoa, X.: Exploratory Analysis of the Main Characteristics of Tags and Tagging of Educational Resources in a Multi-lingual Context. *Journal of Digital Information* **10**(2) (2009).
102. Vygotsky, L.: *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press (1978).
103. Waldrop, M.: *Complexity: The Emerging Science at the Edge of Order and Chaos*. Simons, Schuster, New York (1992).
104. Walker, A., Recker, M., Lawless, K., Wiley, D.: Collaborative information filtering: A review and an educational application. *International Journal of Artificial Intelligence in Education* **14**(1), 3–28 (2004).
105. Wasserman, S., Faust, K.: *Social network analysis: Methods and applications*. Cambridge Univ Pr (1994).
106. Weibelzahl, S.: Evaluation of adaptive systems. In: *User Modeling: Proceedings of the Eighth International Conference, UM2001*, pp. 292-294 (2001).
107. Weibelzahl, S. Evaluation of adaptive systems. PhD dissertation. University of Trier, Germany (2003).
108. Weibelzahl, S., Paramythis, A., Totter, A.: A layered framework for the evaluation of interactive adaptive systems. In: *Proc. 2nd Workshop on Empirical Evaluation of Adaptive Systems*. Johnstown (2003).
109. Wild, F., Mödritscher, F., Sigurdarson, S.E.: Designing for change: mash-up personal learning environments. *eLearning Papers*. **9** (2008).
110. Wilson, S., Sharples, P., Griffiths, D.: Distributing education services to personal and institutional systems using Widgets. In: *Mash-Up Personal Learning environments, Proceedings of the 1st MUPPLE workshop*. CEUR-Proceedings (2008).
111. Worthen, B.R., Sanders, J.R., Fitzpatrick, J.L.: *Program evaluation*. Longman, New York (1997).

Part III
Interacting with Recommender Systems

Chapter 13

On the Evolution of Critiquing Recommenders

Lorraine McGinty and James Reilly

Abstract Over the past decade a significant amount of recommender systems research has demonstrated the benefits of conversational architectures that employ critique-based interfacing (e.g., *Show me more like item A, but cheaper*). The critiquing phenomenon has attracted great interest in line with the growing need for more sophisticated decision/recommendation support systems to assist online users who are overwhelmed by multiple product alternatives. Originally proposed as a powerful yet practical solution to the preference elicitation problem central to many conversational recommenders, critiquing has proved to be a popular topic in a variety of related areas (e.g., group recommendation, mixed-initiative recommendation, adaptive user interfacing, recommendation explanation). This chapter aims to provide a comprehensive, yet concise, source of reference for researchers and practitioners starting out in this area. Specifically, we present a deliberately non-technical overview of the critiquing research which has been covered in recent years.

13.1 Introduction

The evolution of the critiquing research landscape has largely been influenced by the changing needs of online users and the increasingly complex product domains that they have turned to explore. Over the past decade, a variety of critique-based recommendation methodologies have been proposed along with demonstrated evidence of their potential to improve recommendation performance. It could be argued that the significance of a piece of research can be often measured not just in terms of the

Lorraine McGinty
UCD School of Computer Science and Informatics, University College Dublin, Dublin 4, Ireland.
e-mail: lorraine.mcginnty@ucd.ie

James Reilly
Google Inc., 5 Cambridge Center, Cambridge, MA 02142, United States.
e-mail: jamesreilly@google.com

number of questions answered (e.g., methodology proposed to solve a given problem), but also in terms of the number of new issues raised; whose answers present as goals for future research.

This chapter is directed towards early-stage researchers starting out in this area. It aims to provide a useful and deliberately non-technical overview of the promises and pitfalls of critiquing that have been brought into focus in recent years. In Section 2 we start off by describing the benefits of the approach as recognised by early critiquing systems. Next, in Section 3, we outline some of the key issues and challenges that have been identified as well as the approaches that have been taken to improve: (1) critique presentation in view of optimising preference acquisition, and (2) the retrieval performance of critique-based recommenders. Section 4 provides an overview of the various design considerations that have influenced the integration of the critiquing mode of feedback across alternate platforms and user environments (e.g., mobile space, individual and collaborative platforms, etc.). Section 5 summarises the resources, methodologies and evaluation criteria that have been typically adopted by practitioners in the area to date. Finally, by way of conclusion we outline some of the open challenges and opportunities that exist for critique-based recommenders.

13.2 The Early Days: Critiquing Systems/Recognised Benefits

Interactive recommender systems typically engage users in a conversational dialogue to learn their preferences and use this feedback to improve the system's recommendation accuracy. Many conversational recommenders drive the preference elicitation task through the use of examples (for further details see Chapter 21). For instance, a user may be presented with one or more examples/recommendations (e.g., a movie, book, camera) and asked to provide feedback (e.g., provide ratings, indicate a preference). The *critiquing* mode of feedback has become a popular topic amongst those conducting recommender systems research and those developing example-based conversational architectures. The primary reason why it has become so popular is that it strikes an acceptable balance between the effort that a user must expend when providing feedback and the information value it provides. In comparison to the standard value elicitation approach it is a very *low-cost* form of feedback (i.e., in terms of user effort) that provides a relatively unambiguous indication of the user's current requirement (e.g., "Show me more like item A, but different in terms of feature X"). Critiquing is also well-suited to even the most basic interfaces and to users with only a rudimentary understanding of certain recommendation domains [7, 8, 65].

In many domains, it cannot be assumed that users are able to completely articulate their preferences from the outset [48], and/or have a clear understanding of the feature trade-offs/compromises that exist (for related discussion see Chapter 11). For example, a user hoping to buy a new laptop may not initially think about the trade-off that often exists between the products weight and it having an integrated

CD-ROM. If they were to formulate a query for a laptop that was less than 2kg in weight with a CD-rom facility they may not find any products that satisfy *both* of these preferences. Instead, as users become more familiar with the domain and the product options available, their preferences often change, becoming more rigid [48]. Users often tend to lack the motivation to completely specify their preferences up front without any perceived benefits [67]. In fact, this may be unnecessary, as sometimes only a partial set of preferences may be required to make good recommendations [17].

Critique-based conversational recommenders offer flexible support to users as they navigate product catalogues and help them to better understand their preference requirements. Instead of requiring users to specify their preferences from the outset, user preferences are built up over a series of *recommendation cycles*. In each cycle of a *recommendation session* the system makes one or more recommendations to the user and invites them to critique one of the examples. Feature critiques typically take the form of *directional* or *replacement* (a term that was initially defined by [37]). *Directional* critiques effect an increase or decrease over numerical feature values (e.g., *cheaper* implies [$<$ *price*]). *Replacement* critiques allow for the substitution of any value (i.e., aside from critiqued value) for a non-numeric feature (e.g., *different manufacturer* implies [\neq *manufacturer*]). A recommendation satisfying the applied critique is returned and the user is invited to critique this in line with their requirements. This process continues until the user: (1) accepts a recommendation, (2) exhausts all potential possibilities, or (3) terminates the session prematurely.

Early work in this area dates back to the early 1980's when the RABBIT systems [70, 71] introduced the critiquing as a new interface paradigm for formulating database queries. Users could critique fields of example records with options like *prohibit* or *specialise*. Based on user feedback the system would reformulate the query and present another example record. The FindMe Systems [7, 8, 19] developed by Burke *et al.*, were the first to employ critiquing in web-based recommenders recognising the need to focus on educating the user about the options space¹. Originally, FindMe recommenders were developed as *browsing assistants*, that helped users browse through large information-spaces by providing critiques on presented examples, such as restaurants (*Entrée*), automobiles (*Car Navigator*), apartments (*RentMe*), and movie rentals (*Video Navigator*)². Other examples of early example-based critiquing systems include: *Apt Decision* [64], *SmartClient* [53, 54, 55], and the *Automated Travel Assistant* (ATA) [22]. In the next section we outline key challenges in critique-based recommendation that have been addressed by these systems and other subsequent research.

¹ More recent research has highlighted that there is a tension between the need for a user to explore the space of items to understand the options and desire for short recommendation dialog.

² The Wasabi Personal Shopper [4] was developed as a general-purpose domain-independent version of the FindMe systems

13.3 Representation & Retrieval Challenges for Critiquing Systems

Early systems and more recent critiquing research can be reviewed along a number of dimensions. In this section we focus our review on two: critique *representation* and recommendation *retrieval*. In the first instance, we distinguish between the alternate critique representation/formulation approaches that have been developed to optimise preference acquisition. We describe how the nature of the critiques that are presented to the user as feedback options dictate very different limitations and benefits. In the second instance, we outline some of the key issues that have presented retrieval challenges in critiquing systems, and discuss (where appropriate) how these have influenced the development of more sophisticated approaches to preference modeling and revision.

13.3.1 Approaches to Critique Representation

Over the past decade a variety of alternate approaches to critique representation have been proposed. The Entrée [8] restaurant recommender, is one of the earliest and most well-known member of the FindMe family of critique-based recommenders. Entrée offers two ways for users to specify their initial preferences. One is to specify a known restaurant that is similar to what the user is looking for. The alternative requires that users start the recommendation session by specifying their dining interests according to a number of high-level features. Along with the returned recommendation are seven pre-defined *unit* critique options which can be applied over features; *cheaper* critiques the *price* feature, and *more formal* critiques the *style* feature, for example. The critiques serve as temporary filters over the product space, eliminating incompatible restaurants from consideration for the next recommendation cycle. In this section we discuss a number of challenges that motivated further research and advancement in this area.

13.3.1.1 Over-critiquing & protracted recommendation dialogues

Unit critiques only allow users to express preferences over a single feature in each recommendation cycle. This ultimately limits the ability of the recommender to narrow its focus, which can result in slow unnecessarily long recommendation dialogues. Furthermore, a user may not understand the feature trade-offs that exist within a particular domain and hence might be inclined to continue to critique a specific feature (e.g., *price*) until a recommendation with an acceptable value has been reached. However, they may later realise that the value of another important feature has since changed and is no longer acceptable [39]. An alternative strategy is to consider the use of *compound* critiques (the term itself was first introduced by

[26]). These are critiques that operate over multiple features and have the potential to improve recommendation efficiency because they allow the recommender system to gather more preference information in a single recommendation cycle. The promise is that the application of compound critiques enables users to take larger steps though the recommendation space towards preferred options, thus reducing session lengths/interaction times.

The idea of compound critiques is not new. In fact, the early FindMe Systems [7] introduced the *Car Navigator* System (see Fig. 13.1) which uses compound critiquing. Automobiles are described in terms of features such as *horsepower*, *price*, *sportier*, or *gas mileage* that can be directly manipulated by users. Compound critiques are also presented alongside individual feature-level unit critiques providing the user with two alternate ways to refine recommendations. When a user applies the *sportier* compound critique, for example, this has the effect of filtering the remaining options in terms of a number of features; that is, *engine size*, *acceleration and price* are all increased. Similarly, in the context of a PC recommender a *high performance* compound critique might simultaneously increase *processor speed*, *RAM*, *hard-disk capacity* and *price* features.

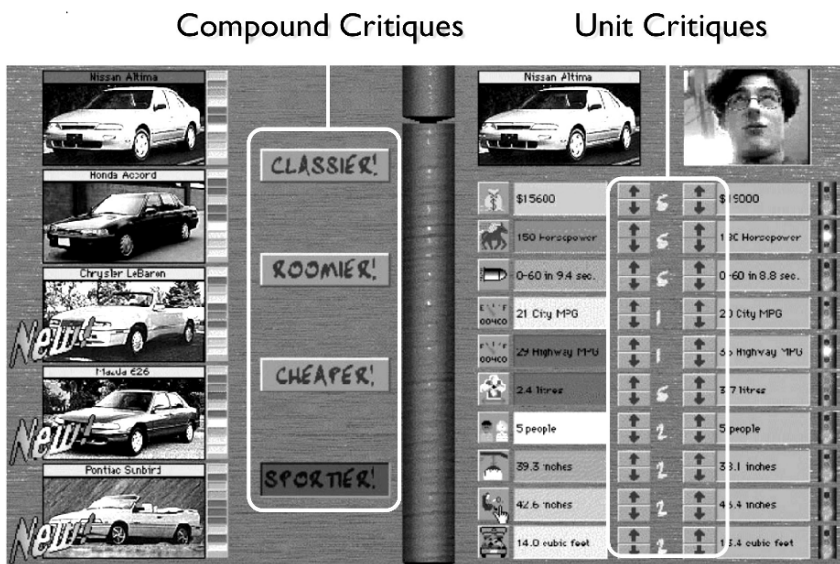


Fig. 13.1: Illustrating how *Car Navigator* presents both static unit and compound critiques.

13.3.1.2 Critique redundancy & hidden feature-dependency

The notion of automated critique generation was first proposed by McCarthy *et al.* [26], motivated by the observation that certain *static* critiques may not always be relevant. They identified that *session dead-ends* could result whereby a user may apply a critique filter (i.e., unit or compound) and find that there are no subsequent matching recommendations (related work refers to these as *retrieval failures* [18, 37, 39]). For instance, in the *Car Navigator* system the *sportier* critique would continue to be presented as an option even when there are no recommendation candidates remaining that match that criteria. Static critiques (both *unit* and *compound*) do not afford the user any understanding of what recommendation options are available beyond the current example. In fact, they could lead to unnecessary user confusion and interface redundancy. For instance, users may not all share the same understanding of what these static *labels* actually mean (i.e., they are uncertain about what hidden feature filters might be applied). Inevitably, this means that some users may find these static compound critiques to be of little use.

McCarthy *et al.* [26] argue the need for a more flexible *dynamic* approach to critique generation whereby compound critiques are composed on-the-fly, on each recommendation cycle, in view of presenting applicable critiques that (1) better focus the recommender, (2) eliminate interface redundancy, and (3) remove user apprehension. They make the point that only compound critiques that actually *cover* available recommendation candidates should be presented to the user. Of course, it is reasonable to apply the same rationale to the generation of *unit* critiques, as is implemented by the *Tweak* system [39] to ensure that the user is only presented with unit critique options that lead to at least one product option. Three approaches to the *dynamic* generation of compound critiques have been proposed by (see [26] and [73], & [12]). All of the approaches provide the user with multi-feature critique options that expose the user to the feature changes that will result from an application of that critique (i.e., the recommendation consequence). For example, a user could apply the critique (*more memory, less price, different manufacturer*) if they want to see an alternate PC from that is similar to the current recommendation but cheaper, with more memory, and from a different manufacturer. All approaches have been demonstrated to be effective in terms of the benefits they offer to varying degrees with respect to recommendation efficiency, accuracy, applicability, and usability. However, the key distinguishing factor between the approaches is the knowledge they use to influence critique generation (i.e., domain vs user preference knowledge). Further differences and the motivations for each methodology are described in the next subsections.

13.3.1.3 Limited product-space vision

Reilly *et al.* [59] demonstrate how to use only domain-knowledge about *available* recommendation candidates to automatically generate compound critiques before every recommendation cycle in their *QwikShop* system (see Fig. 13.2). They con-

centrate on increasing the users depth of understanding about characteristics governing the remaining recommendation candidates, through raising awareness about the feature trade-offs and dependencies that exist beyond the current recommendation. Their so-called *dynamic compound critiquing algorithm* uses the Apriori association data-mining algorithm [1] in order to uncover any frequently occurring feature relationships amongst the remaining recommendation candidates. Typically, large numbers of compound critiques of the form (e.g., *more resolution, more memory, and different manufacturer* [than the current PC recommendation]) are generated by this methodology. It is not practical to present the large lists of compound critiquing options from a user interfacing perspective. Various strategies could be employed for ranking compound critique options (e.g., support, confidence, leverage, lift, and conviction³) in order to present the user with *k* compound critiques alongside the standard unit critique options. McCarthy *et. al.* [26] and Reilly *et. al.* [56] investigate how best to rank compound critiques by their support values (where the *support* refers to the proportion of the products that satisfy the critique). They demonstrate that presenting users with low-support compound critique options provides the best balance with respect to their likely applicability and their ability to focus the search.

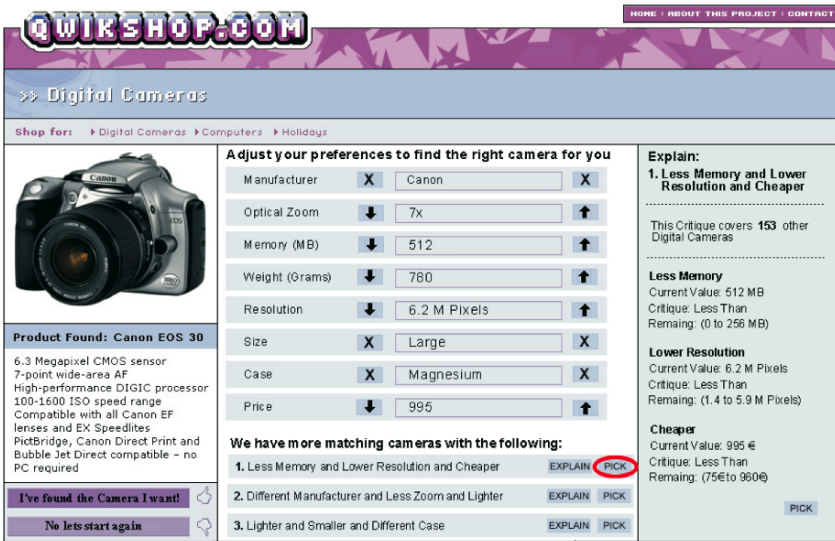


Fig. 13.2: Screen-shot of the QwikShop critiquing system.

It is possible that the ultimate selected critique options that are ultimately selected show *limited feature diversity* (i.e., their constituent individual features can overlap to a high degree) due to the high number of remaining options that exhibit the same feature differences to the current recommendation. The related issue of selecting rec-

³ These are commonly used interest measures for association rules [1].

ommendation candidates for a user, that are relevant to their preferences but different from each other, is a familiar problem to those working in the recommender systems area. In response, a number of diversity-enhancing solutions have been proposed (see Section 13.3.2). Approaches for dynamically presenting diverse compound critiques have been investigated [28] and shown to generate feedback options that are up to 32% more applicable to users. A further potential limitation of this approach is that it does not take into account a users' preferences. In the *QwikShop* system user preferences influence only how recommendations are selected, and not the feedback options presented to the user. However, being less tightly bound to the users preference model, this approach has more potential for the discovery of serendipitous recommendations.

13.3.1.4 Weak relevance of presented feedback options

As mentioned above there is no guarantee that the compound critiques generated by Reilly *et. al.*'s methodology will be relevant to the user as it is influenced only by product domain knowledge. Zhang and Pu [73] propose an alternative methodology that relies on user preference knowledge. Their Multi-Attribute Utility Theory (MAUT), approach that is similar to that implemented by the SmartClient system [53, 54, 55], as the means for dynamically generating compound critiques. Their *EasyShop* recommender, shown in Fig. 13.3, maintains user preference models based on their critiquing feedback and these are used to calculate product *utility* values for each recommendation candidates.

The product with the highest utility is selected as the next recommendation and the k products with the next highest utility values are represented by compound critiques and presented as feedback options. Comparative off-line evaluations [60] show how these preference oriented compound critiques tend to be more more aligned with the user's intended critiquing criteria than the approach described McCarthy *et. al.* [26]. In addition, they tend to result in shorter sessions with higher recommendation accuracy. However, the MAUT approach does suffer from some drawbacks. First, each MAUT-generated compound critique describes only one product, and so these critiques offer users limited exposure to the remaining recommendation opportunities. Secondly, this approach assumes an accurate user preference model. However, user preferences can be very inconsistent and are often subject to rapid change (see Section 13.3.2). Finally, it does not promote diversity across the critiques, and the compound critiques tend to contain a high number of features.

13.3.1.5 Limitations of domain & preference driven approaches

Chen & Pu [12] propose a methodology for dynamically generating compound critiques that aims to preserve the advantages (while minimising the limitations) of the previously described alternatives. Their *preference-based organisation* approach



Fig. 13.3: Screen-shot of the EasyShop critiquing system (from [73]).

aims to dynamically generate diverse critique options that are adaptive to user preferences and representative of the remaining recommendation candidates.

User preferences are represented using a multi-attribute utility model. A data-mining algorithm generates the set of compound critiques. These compound critiques are categories of available recommendation candidates that best match the user’s preference model. From each compound critique, a selection of products are extracted for presentation such that they exhibit high trade-off utilities against the current recommendation and are diverse from each other. The trade-off utility for a category indicates how well it adapts to the user model. The intuition is that a higher tradeoff utility category covers products that potentially offer more pros than cons over the current best candidate (see Fig. 13.4).

13.3.1.6 Restricted user control

In more recent work, Chen and Pu [9], make the distinction between system-proposed and user-motivated critiquing approaches (see Fig. 13.5). Up to this point we have concentrated our discussion on what they refer to as system-proposed compound critiques whereby the presented critiques are not defined by the user. Chen and Pu present the example of a user looking for a digital camera with higher resolution and more optical zoom relative to the current recommendation. Suppose

The top candidate according to your preferences									
Manufacturer	Price	MegaPixels	Optical zoom	Memory type	Flash memory	LCD screen size	Depth	Weight	
Canon	\$242.00	5.0 MP	3x	CompactFlash Card	32 MB	1.8 in	1.37 in	8.3 oz	choose

We have more products with the following									
they are cheaper and lighter, but have fewer megapixels									
Nikon	\$167.95	4 MP	3x	SD Memory Card	14 MB	1.9 in	1.4 in	4.6 oz	choose
Canon	\$230.00	4.1 MP	3x	CompactFlash Card	32 MB	1.5 in	1.09 in	6.53 oz	choose
Canon	\$180.00	3.3 MP	3x	SD Memory Card	16 MB	2 in	0.83 in	4.06 oz	choose
Canon	\$219.18	4.2 MP	4x	MultiMedia Card	16 MB	1.8 in	1.51 in	6.35 oz	choose
Canon	\$163.50	3.2 MP	4x	MultiMedia Card	16 MB	1.8 in	1.5 in	6.3 oz	choose
Canon	\$199.40	3.2 MP	2.2x	SD Memory Card	16 MB	1.5 in	1.4 in	5.8 oz	choose

they have more megapixels and bigger screens, but are more expensive									
Sony	\$365.00	7.2 MP	3x	Internal Memory	32 MB	2.5 in	1.5 in	6.9 oz	choose
Canon	\$439.99	7.1 MP	3x	SD Memory Card	32 MB	2 in	1.04 in	6 oz	choose
Fuji	\$253.00	6.3 MP	4x	XD-Picture Card	16 MB	2 in	1.4 in	7.1 oz	choose
Sony	\$336.00	7.2 MP	3x	Internal Memory	32 MB	2 in	1 in	5 oz	choose
Nikon	\$304.18	7.1 MP	3x	Internal Memory	13.5 MB	2 in	1.4 in	5.3 oz	choose
Olympus	\$334.00	7.4 MP	5x	XD-Picture Card	32 MB	2.0 in	1.7 in	7.1 oz	choose

they are lighter and thinner, but have less flash memory									
Pentax	\$238.99	5.3 MP	3x	Internal Memory	10 MB	1.8 in	0.8 in	3.7 oz	choose
Canon	\$273.18	4.0 MP	3x	SD Memory Card	16 MB	2 in	0.82 in	4.59 oz	choose
Nikon	\$329.95	5.1 MP	3x	Internal Memory	12 MB	2.5 in	0.8 in	4.2 oz	choose
Canon	\$316.18	5.3 MP	3x	SD Memory Card	16 MB	2 in	0.81 in	4.59 oz	choose
Casio	\$386.00	7.2 MP	3x	Internal Memory	8.3 MB	2.5 in	0.88 in	4.48 oz	choose
Fuji	\$309.18	6.3 MP	3x	XD-Picture Card	16 MB	2.5 in	1.1 in	5.5 oz	choose


they have more optical zoom with different memory type, but are thicker and heavier									
Panasonic	\$386.00	5.0 MP	12x	SD Memory Card	16 MB	1.8 in	3.34 in	11.52 oz	choose
Konica Minolta	\$349.99	5.0 MP	12x	SD Memory Card	16 MB	2 in	3.3 in	12 oz	choose
Fuji	\$259.18	4.23 MP	10x	XD-Picture Card	16 MB	1.5 in	3.1 in	11.9 oz	choose
Olympus	\$253.00	4.0 MP	10x	XD-Picture Card	16 MB	1.8 in	2.7 in	9.9 oz	choose
Olympus	\$284.99	4.0 MP	10x	XD-Picture Card	16 MB	1.8 in	2.7 in	10.6 oz	choose
Nikon	\$259.18	4.2 MP	8.3x	Internal Memory	13.5 MB	1.8 in	2.2 in	9 oz	choose

Fig. 13.4: Illustrating the *preference-based organisation* approach to critique presentation (from [12]).

there is no suggested compound critique matching the user’s current requirements, even though the proposed critiques can give them some information about subsequent recommendation options (e.g. *greater screen size & more memory*). At this point, they can only apply unit critiques to one feature at a time; at the risk of being involved in longer interaction cycles and lower levels of decision accuracy. They suggest that this limitation could be addressed by allowing users the flexibility to define their *own* compound critiques. The assumption is that only unit critique options that cover subsequent recommendation options are presented to the user (i.e., dynamic unit critiques). The *user-motivated* critiquing approach invites the user to make one or more critique selections over any combination of these category (i.e., compound critique) options.

Apt Decision, proposed by Shearin & Lieberman [64] is an example of an early critiquing system that also implements a user-motivated approach. Users in the apartment rental market are free to critique multiple features (from a set of 21 possibilities) relating to recommended apartment descriptions. Chen and Pu take a different approach, affording the user greater control over what preference constraints influence recommendation. They describe how users have the freedom to specify their tradeoff criteria in terms of improvement and compromise regarding the rec-

To find similar products with better values than this one



Canon PowerShot S2 IS Digital Camera Add to saved list
\$424.15
 Canon, 5.3 M pixels, 12x optical zoom, 16 MB memory, 1.8 in screen size, 2.97 in thickness, 404.7 g weight. [detail](#)

would you like to improve some values?

	Keep	Improve	Take any suggestion
Manufacturer	<input checked="" type="radio"/> Canon	<input type="radio"/> Sony	<input type="radio"/>
Price	<input type="radio"/> \$424.15	<input checked="" type="radio"/> less expensive	<input type="radio"/>
Resolution	<input checked="" type="radio"/> 5.3 M pixels	<input type="radio"/> \$100 cheaper	<input type="radio"/>
Optical Zoom	<input checked="" type="radio"/> 12x	<input type="radio"/> \$200 cheaper	<input type="radio"/>
Removable Flash Memory	<input checked="" type="radio"/> 16 MB	<input type="radio"/> \$300 cheaper	<input type="radio"/>
LCD Screen Size	<input checked="" type="radio"/> 1.8 in	<input type="radio"/> more memory	<input type="radio"/>
Thickness	<input checked="" type="radio"/> 2.97 in	<input type="radio"/> larger	<input type="radio"/>
Weight	<input checked="" type="radio"/> 404.7 g	<input type="radio"/> thinner	<input type="radio"/>
		<input type="radio"/> lighter	<input type="radio"/>

Show Results Reset

Fig. 13.5: Illustration of user-motivated critiquing interface taken from [9].

ommendation features. For example, users can indicate that they would prefer to see a recommendation that is (e.g., *X amount cheaper* [than the current recommendation]) as shown in Fig. 13.5.

Fig. 13.6 shows how Apt Decision, by contrast, provides direct access to the preference profile where the user can easily indicate preference revisions (both positive and negative). Aside from allowing users to indicate the importance of their feature preferences, Apt Decision does not support users to define any more-specific boundary constraints in terms of the extent of the compromises they are willing to accept. Chen & Pu show that users achieved higher confidence in choice and decision accuracy through user-motivated critiquing. However, some users still preferred the system-proposed critiques, reporting that they found them intuitive to use and quickly led to suitable products when relevant options were presented. *SmartClient* [53, 54, 55] also affords greater user-control over preference elicitation. Designed to help users find airline flights, it has also been used to recommend vacation packages, insurance policies and apartment rentals. Again, product selection is represented as a decision theory problem where the user is trying to select a product that optimally satisfies their preferences. Choosing the best product is often a trade-off problem, where the user must decide how and if he should compromise on some product features in order to optimise others. *SmartClient* employs critiques as soft constraints. The interface allows the user to directly construct a value function (by specifying weights) for each attribute and the recommender combines this with the constraints

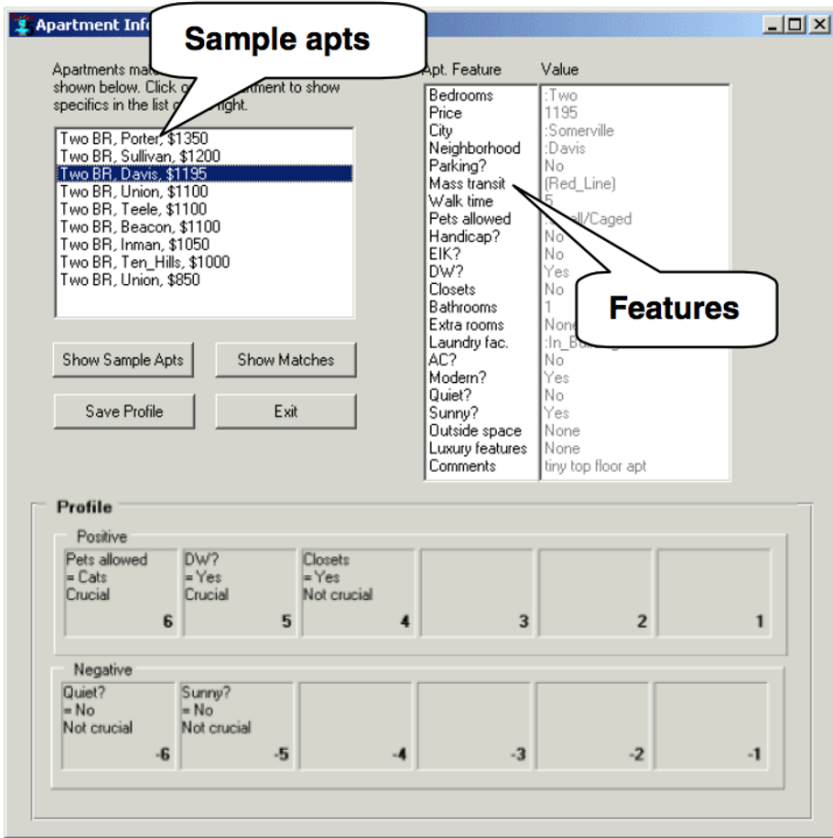


Fig. 13.6: Screen-shot of the Apt Decision user-motivated critiquing interface adapted from [64].

to compute a *utility* for each attribute and product. The system presents a ranked list of the best products with the highest utility. When the user revises his preferences, either through the addition of another critique or by changing its weight, the products are re-ranked to reflect the updated utilities.

13.3.2 Retrieval Challenges in Critique-Based Recommenders

Irrespective of the type of critiques that are used (e.g., unit/compound), or the manner in which the critiques have been generated (e.g., static versus dynamic, system-proposed vs user-motivated), recommendation success is heavily influenced by user critiquing behaviour. As an example, we mentioned earlier that users may *over-critique* a feature (e.g., *price*), and find that they are presented with recommenda-

tions where none of their other feature preferences are satisfied [39]. While little can be done to control how the user chooses to apply critiques, there are some retrieval challenges that *can* be avoided in view of improved recommendation performance. In this section we outline a number of these and briefly discuss how some of them have already been addressed.

13.3.2.1 Preference inconsistency and longevity

The preference model behind the *FindMe* system is a feature vector obtained from the entry example along with the user's initial high-level feature constraints. The application of a critique will update the model with the most recent feature critique and temporarily remove those restaurants that are incompatible with it from the recommendation candidates. The remaining restaurants are then sorted using a hierarchical sort on their similarities to the preference model.

As mentioned earlier in Section 13.2, users cannot be relied upon to provide consistent feedback throughout the course of a recommendation session. Traditional critique-based recommenders (e.g., *RentMe*, *Entrée*) focus on the current critique and the current recommendation, without considering the critiques that have been applied in the past. This can also lead to retrieval failures. Many users are unlikely to have a clear understanding of their requirements at the beginning of a recommendation session. As a result users may select apparently incompatible critiques during a session as they explore different areas of the product space in order to build up a clearer picture of what is available. For example, a given cycle may find a prospective digital camera owner looking for a camera that is *cheaper* than the current €500 recommendation, but later may ask for a camera that is *more expensive* than another €500 recommendation.

Apt Decision [64] maintains more flexible user preference models of user critiques. A key differentiator between it and others is that it has separate positive and negative preference models which are visible and accessible to users. It also supports preference-based comparisons of apartments from which further implicit preference information is gathered and added to the model. Shearin & Lieberman argue that learning an accurate profile is more beneficial than trying to constrain the search-space based on user preferences [64]. If the search-space is over-constrained, users will be unaware of the other potential options that exist containing features that they might be willing to compromise on. Instead, relaxing constraints allows users to explore the product space more, giving the recommender the opportunity to learn more preferences and make better recommendations.

The *SmartClient* system also treats applied critiques as explicit representations of user preferences. Both employ constraint solvers to obtain optimal solutions. User preferences in the form of critiques are modelled as constraints in the CSP formalism. An agent constantly observes the users modifications to the expressed preferences and refines the elicited model to improve solution accuracy. Zhang & Pu [73] describe a MAUT-preference model that adjusts the *utility* of feature preferences based on a user's critiquing feedback. Recommendations that maximize

overall preference utilities are subsequently retrieved. In similar work, Chen & Pu [12] describe a MAUT-preference modeling approach whereby a user can specify their tradeoffs by directly manipulating criteria.

Reilly *et. al.*'s *incremental critiquing* [58] approach maintains a user preference model which is made up the actual critiques that have been applied by the user so far. The intuition is that the critiques applied so far provides a representation of the user's evolving requirements. To maintain accuracy of the user model *inconsistent* critiques are eliminated, as are all existing critiques for which the most recent critique is a refinement. When retrieving recommendations priority is given to those candidates that: (1) satisfy the current critique; (2) are similar to the previous recommendation; and (3) satisfy a large proportion of previous critiques.⁴ Given two candidates that are equally similar to the previously recommended case, their algorithm will prefer the one that satisfies the greater number of recently applied critiques (i.e., that which returns the higher *compatibility* score). Consequentially, they report session-length reductions of up to 70% [58].

Finally, Nguyen & Ricci have more recently motivated the need to maintain alternate preference models that distinguish between long-term and session-specific user preferences. They propose a methodology for integrating both kinds of preference information in order to generate personalised recommendations [41]. Session-specific preferences include both contextual preferences (e.g., a restaurant open at the time in question or within proximity of the user) and product feature preferences (e.g., a cheap chinese restaurant). Long-term user preferences [44] refer to information (about the user) which persists along a relatively long timespan (i.e., through many consecutive recommendation sessions). These preferences are typically elicited at registration time, and revised later through continued system use.

13.3.2.2 Diminishing choices & unreachability

As described earlier, early critique-based recommenders applied critiques act as temporary filters over the remaining available product options and all *critiqued* recommendations are eliminated from further consideration. McSherry & Aha uncover a potential drawback - a problem they refer to as *the diminishing choices problem* [40]. They argue that by preventing users from navigating back to products they critiqued earlier can result in retrieval failures when the *only* acceptable product option has been eliminated. They present the example shown by Fig. 13.7 where a user is presented with Case 1 (i.e., a 3 bedroom detached property in location A). Suppose that the user would *prefer* a 4 bedroom detached property in that same location. Assuming that features are equally weighted and the similarity between two cases is the number of matching features, a critique over the bedroom feature (i.e., > Beds) would see the user presented with Case 2. Realising that there is no case that will satisfy all of their requirements the user may then which to re-evaluate Case 1 how-

⁴ In so doing they are implicitly treating the past critiques in the user model as *soft constraints* for future recommendation cycles; it is not essential for recommendations to satisfy all of the previous critiques, but the more they satisfy, the better they are regarded as recommendation candidates.

ever this has been removed they have no choice but to consider Case 3 instead (or re-start their recommendation session).

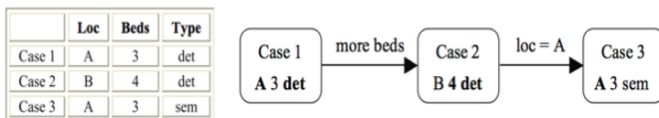


Fig. 13.7: Illustrating the diminishing choices problem that can result as a consequence of eliminating critiqued recommendations [from [40]].

On the contrary, the *unreachability problem* [40, 18] refers to the consequence of *not* eliminating previous recommendations. In [40] McSherry & Aha also demonstrate that this can potentially lead to the situation whereby acceptable products that satisfy the user’s requirements, if any, will never be retrieved. This problem is well illustrated by Fig. 13.8 where a user is sequentially presented a recommendation from the catalogue of options shown, and on the fourth critique application is brought back to the initial recommendation without ever being presented with Case 5. An important point here is that recommendation retrieval in this instance is influenced only by the most recently applied critique; that is, the initial query and any previous critiques are not considered.

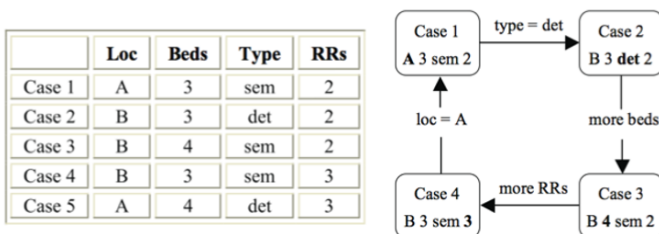


Fig. 13.8: Illustrating the unreachability problem that can result as a consequence of *not* eliminating critiqued recommendations [from [40]].

The *Tweak* system described by McSherry & Aha [39] implements the *progressive critiquing* approach, which like incremental critiquing [58] involves maintaining a history of a user’s previous critiques. The recommendation retrieved in response to a user’s critique must also satisfy any previous critiques and constraints specified by the initial query. Fig. 13.9 shows how a recommendation option satisfies a users set of *current constraints* which consists of their initial query (i.e., *make=Dell* and *price <= 1000*) and their previous critiques (i.e., *type=laptop* and *< screen size*). Note that *type=laptop* is the only constraint provided by the user at this point that is not satisfied by the current recommendation. The progressive critiquing approach retrieves the most similar product to the current recommendation

and critique, prioritising those preferences contained in the preferences model. This model records *explicit* (E), *assumed* (A), and *predicted* (P) preferences for feature values. Here explicit preferences refer to those set out by the initial query and/or as the result of subsequent replacement critique applications on nominal features (e.g., *type=laptop*). Assumed preferences refer to preferences decisions that are implicitly made in relation to features such as *price* (i.e., where it may be reasonable to assume that *less-is-better* (LIB)) and *memory* (i.e., *more-is-better* (MIB)). And, predicted preferences refer to those features where users tend to have an *ideal* value and would prefer values that are close to this (i.e., *nearer-is-better* (NIB))[37, 35].

Recommended Case	Current Constraints		Preferences	
make = Dell	make = Dell	Y	make = Dell	E
price = 719	price ≤ 1000	Y	price = 658	A
type = desktop	type = laptop	N	type = laptop	E
screen = 15	screen > 12	Y	screen = 13.3	P
speed = 1.6			speed = 2.2	A
memory = 512			memory = 2000	A
hard disk = 20			hard disk = 120	A
chip = Intel Pentium				

Fig. 13.9: Illustrating the knowledge available in a typical progressive critiquing session; current recommendation, current query, and user preferences [from [39]].

Unlike incremental critiquing, revisions to the preference model within the *Tweak* system are made on the basis of the most recent critiquing event without reference to the history of critique applications on that feature. That is, the application of a *make=Sony* critique replaces the value *Dell* within the preference model with *Sony*. For the NIB features, the preferred value is predicted to be that *nearest* to the critiqued recommendation (e.g., a *< screen* critique would result in setting the screen value to 14). Incremental critiquing, by contrast, accounts for the fact that a user may have been presented with a number of *Dell* options over a series of prior recommendation cycles but was content to accept it at that point (i.e., they prioritise other critique changes). The *compatibility* measure used by incremental critiquing takes into account each candidate product's similarity to the current and the number of previous critiques it satisfies. However, a potential problem here is that the existence of a highly similar product may play more influence and be retrieved over alternatives that satisfy more critiques. Similar to Nguyen & Ricci [41], Salamo *et al.* [63] addresses the issue by using similarity to the critiqued recommendation as a secondary retrieval criterion.

In agreement with Shearin & Lieberman [64], McSherry suggests the system's current understanding of user preferences should be accessible to the user. He feels this is especially important in the interest of avoiding what he terms *progression failures* (i.e., the non-existence of a product that satisfies all of the user's requirements). In Shearin & Lieberman's *Apt Decision* system, the user has constant and

direct access to their preference model. While this provides the user with some depth of understanding as to why certain recommendations are retrieved, it is still open to progression failures. McSherry takes an interesting explanation-based approach where in the event that all of the preference constraints are not met by any recommendation candidate the user is provided with an explanation of what compromises might be appropriate. A more recent version of the progressive critiquing algorithm [40] concentrates on addressing the unreachability problem. The key difference in this implementation of progressive critiquing is that previously recommended (i.e., critiqued) items are not eliminated, as is the case with the incremental approach.

13.3.2.3 Refining recommendation retrievals

Although the majority of critique recommenders, such as *QwikShop*, *EasyShop*, *CritiqueShop*⁵, *Entrée*, and *RentMe* concentrate on retrieving only one candidate in each recommendation cycle, others like *Apt Decision* and *MobyRek* present the user with a set of recommendations to choose from. Refining recommendation candidates can be challenging. Early work in this area introduced the *EntréeC* hybrid recommender system [5, 6] which adds a collaborative filtering (see Chapter 5) component to *Entrée* [7], creating a knowledge-based/collaborative cascade hybrid. Like *Entrée*, it uses its knowledge of restaurants to make recommendations based on the user's stated preferences. The collaborative recommender is called upon to refine the competing recommendations returned by the knowledge-based recommender. Key benefits for using the cascading approach here are that it is more efficient than, for example, a weighted hybrid that applies all of its techniques to all items, since the cascade's second step focuses only on those items for which additional discrimination is needed. Furthermore, the cascade is, by nature, tolerant of noise in the operation of a lesser-priority technique, since results returned by the high-priority recommender can only be refined, not overturned. Experiments with *EntréeC* indicate that collaborative filtering does improve the performance over the knowledge-based *Entrée* system acting alone. In [6] Burke discusses the trade-offs that exist between alternate recommendation techniques and reviews some of the combination methods that have been used in other conventional *hybrid* recommender systems, that also would be relevant to critique-based recommenders (see also Chapters 22, 2, and 24).

Faltings *et. al.* [15] argue that a key design question in critiquing systems is *what* recommendations to present users in order to best help them locate their most preferred solution. They propose two key requirements: (1) that the recommendations must stimulate the user to express further preferences (i.e., by showing the range of alternatives available), (2) that presented recommendations must contain the solution that the user would consider optimal (if the currently expressed preference model was complete) so that they could select it as a final solution. The *dynamic critiquing* approach attempts to address these requirements through the provision of

⁵ <http://www.critiqueshop.com/>

compound critiques that describe where the user can navigate to. However, systems that rely on *unit* critiques may find the low feature-level feedback it provides can be insufficiently detailed to sharply focus the next recommendation cycle [33]. For example, by specifying that they are interested in a digital camera with a *greater resolution* than one of the presented recommendations the user is helping the recommender to narrow its search but this may still lead to a large number of available candidates to choose from. Instead a combination of critiquing and value elicitation feedback modes (e.g., Show me a 5 megapixel camera) is likely to reduce the number of product options much more effectively in this instance.

McGinty & Smyth [32, 33] demonstrate how critique-based recommenders can suffer from protracted recommendation sessions, when compared to value elicitation approaches. As a potential solution they describe a novel *switching* strategy whereby the mode of retrieval is adapted in accordance with user critiquing behaviour. They demonstrate how their *Adaptive Selection* strategy, offers potential dialogue reductions of 60% over standard similarity-based and diversity-enhanced retrieval approaches.

In more recent work, Chen and Pu [9] describe their *example critiquing* approach, recognising that user preferences are often context dependent. Here, multiple recommendations are presented and the user chooses one to critique. They take a *preference-based organisation* [12] approach to recommending the highest utility products and also allow users to freely compose compound critiques over multiple features which indicate the simple and complex trade-offs they are prepared to make.

13.3.2.4 Multi-user preference handling

The task of recommending items/products to a group of users presents a number of challenges (see for example, Chapters 22, 18, 21 and 16 in addition to the works of [3], [47], and [24]). Early work in this area by Jameson [20] highlights a number of these (summarised by Fig. 13.10) and considers how they could be dealt with within the context of a prototype group recommender: *The Travel Decision Forum*. While these systems are susceptible to the same problems as single-user recommenders (e.g., preference inconsistency and volatility) other key distinguishing characteristics include the need to promote *mutual awareness* of individual preferences amongst group members, and *consensus negotiation* (see Section 13.4 for an overview of how the interface plays a critical role here). While the notion of generating a set of recommendations to satisfy a group of *distributed* users with potentially competing interests is challenging in itself, a further challenge is how to record and combine the preferences (and resolve conflicts) for multiple users as they engage in *live* synchronous recommendation dialogs [29]. Key objective questions here include: (1) how can multi-user interaction be managed to facilitate the harvesting of feedback and preferences from multiple simultaneous users?, and (2) how to dynamically maintain models of individual and group preferences with a

view to influencing recommendation such that the resulting suggestions are likely to satisfy *both* the individual and the group?

Recent work by McCarthy *et al.* [29, 31] has concentrated on preference aggregation and consensus negotiation within a critique-based, group recommendation architecture. They introduce the *Collaborative Advisory Travel System (CATS)* designed to provide assistance to a group of friends trying to arrive at a consensus in relation to planning a skiing vacation together. The system supports both individual and multi-user feedback modes through the use of both dynamic unit and compound critiques, and individual and group preference models. It is reasonable to assume that an individual user may need to revisit a previously seen recommendation in this kind of a system in the light of subsequent feedback from other group members. For example, a user may later be willing to compromise on the *price* of a holiday in the knowledge that other members are also willing to do so. *CATS* supports this by facilitating the generation of both *proactive* as well as *reactive* recommendations. *Reactive* recommendations refer to those suggestions presented to the individual user, in response to their own critiques based on their personal user model. *Proactive* recommendations, by contrast, are automatically generated by the system to the group when a recommendation candidate satisfies an unusually high proportion of group preferences; irrespective that a member may have previously rejected this candidate. In addition, individual group members can also identify to the system what they feel are potential recommendation candidates that may be of interest to the whole group. Preference inconsistencies within individual user preference models are managed through incremental critiquing. The group model may contain conflicting preferences and the objective when generating recommendations is that these inconsistencies are minimized by preferring candidates that are maximally compatible multi-user preferences.

Phase of the recommendation process	Difference from recommendation to individuals	Novel issue
1. Members specify their preferences.	It may be desirable for members to examine each other's preference specifications.	What benefits and drawbacks can such examination have, and how can it be supported by the system?
2. The system generates recommendations.	Some procedure for aggregating preferences must be applied.	How can the aggregation procedure effectively discourage manipulative preference specification?
3. The system presents recommendations to the members.	The (possibly different) suitability of a solution for the individual members becomes an important aspect of a solution.	How can relevant information about suitability for individual members be presented effectively?
4. Members decide which recommendation (if any) to accept.	The final decision is not necessarily made by a single person; negotiation may be required.	How can the system support the process of arriving at a final decision when members cannot engage in face-to-face discussion?

Fig. 13.10: Challenges facing group recommendation architectures as summarised by Jameson [20].

To the best of our knowledge (at the time of writing) there is no other work in the area of critique-based recommendation that has concentrated on modeling multi-user preferences. As such, in the interest of completeness, we draw some compar-

isons with typical (non-critique-based) group recommenders. Existing multi-user negotiation/collaborative applications range from virtual environments [50] to sales by action [16]. For the most part of these systems assume an automated negotiation that is based on existing static individual preference models in the system. The *live* interactive nature of the *CATS* system renders the use of static preference models inappropriate. In *CATS* individual preference models tend to be especially volatile as a consequence of the influence of multi-user feedback. Other research in the general area of *group* recommendation includes the *MusicFX* System [25]. *MusicFX* is a group preference arbitration system automatically adjusts the selection of music playing in a fitness center to best satisfy the musical tastes of a group in the same environment. *PolyLens* [46] is a generalization of the *MovieLens* system that recommends movies to groups of users through preference modelling. In contrast, it uses collaborative filtering which draws on the historical music preferences of other users in similar contexts. Like *CATS*, the *Travel Decision Forum* [20] helps a group of users to agree on a vacation that they are planning to take together. However, it concentrates on supporting users who are not co-located. In other related work, Plua & Jameson [49] propose a group recommendation approach where users can get help from others in their group about preferences when their domain knowledge may be incomplete. Unlike *CATS*, this system was intended for use by a group that interact asynchronously rather than simultaneously.

13.4 Interfacing Considerations Across Critiquing Platforms

Different domain and platform characteristics present recommender interface designers with very different technical and usability challenges. In this section we concentrate on design decisions that have been implemented within existing critique-based recommenders (see also Chapter 16). Unsurprisingly, a common theme is how best to manage transparency and control, while also ensuring the level of cognitive and interaction effort required of the user is kept to a minimum.

13.4.1 Scaling to Alternate Critiquing Platforms

While the majority of critique-based recommenders assume desktop web-based platforms (e.g., *The FindMe Systems*, *QuickShop*, *EasyShop*, *CritiqueShop*, *AptDecision*, *SmartClient*), critiquing has also been demonstrated to be an effective interfacing methodology across other platform settings. For instance, Ricci & Nguyen [61, 43, 62] concentrate on interface design and evaluation for critique-based recommenders targeted at mobile devices. They point out that very few web-based systems are designed for mobile users; none of them being conversational. Persistent direct manipulation of the interface is not practical here. Key challenges include: that these devices have much smaller screen-sizes, less keypad functionality than traditional

PDA's (e.g., Palm or Pocket PC), and limited computing power. Their *MobyRek* mobile portal interfaces with a web-based recommender that helps users make travel plans in advance. Essentially, *MobyRek* offers the *on-tour* support required when a user is traveling to or has arrived at their destination.

Staying in the mobile space, Fig. 13.11 illustrates how the *CritiqueShop* interface has been scaled to the iPhone. The key considerations influencing design were the limited screen area and increased opportunity for direct user manipulation (bearing in mind that the iPhone has a touch-sensitive interface). Another design decision made here was to move away from text-based compound critique representation, preferring instead to present more visual alternatives (this is discussed later).

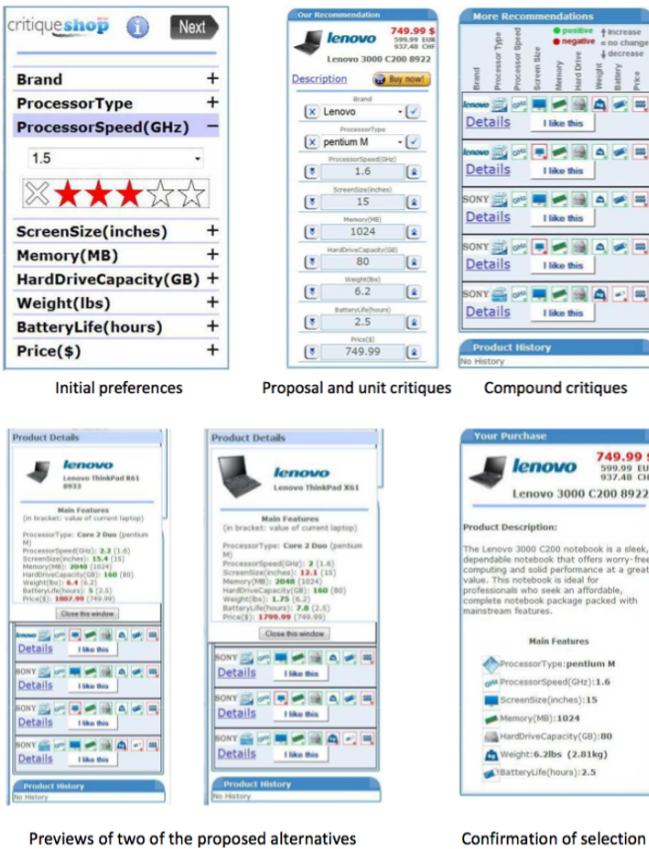


Fig. 13.11: Screen-shot of the CritiqueShop visual interface for the iPhone [www.critiqueshop.com].

In other work, McCarthy et. al. [30] describe how the *CATS* group recommender operates on the interactive, multi-user MERL *DiamondTouch*⁶ table-top device. The DiamondTouch table is a multi-user, touch-and-gesture-activated screen for supporting small group collaboration. Users interact with the display simultaneously (i.e., without having to take turns), browsing through the potential ski holiday options and critiquing recommendations returned to them (i.e., recommendations that are influenced by the evolving individual and group member critiquing patterns, as discussed earlier). Users can also copy and paste potential options of interest to other users, as well as confer on a face-to-face basis about their preferences as they interact in this unusual manner.

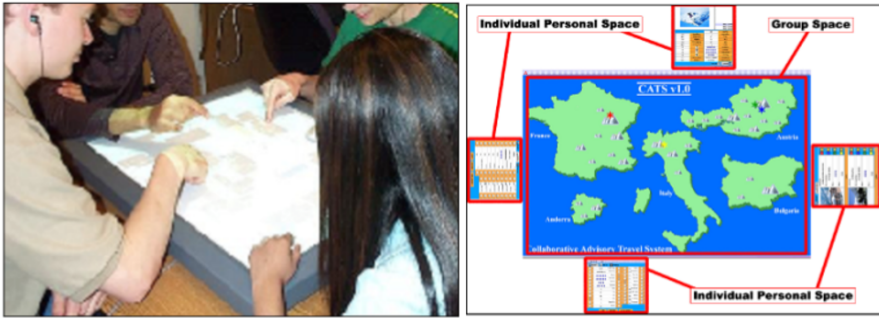


Fig. 13.12: Illustrating the illustrating the CATS interaction with the Diamond Touch.

13.4.2 Direct Manipulation Interfaces vs Restricted User Control

A fundamental consideration that has influenced the design of critiquing interfaces is the importance of finding the right balance between eliciting precise preference information and the cost associated with acquiring it. Early critique-based recommenders offered very restricted user control (i.e., *static* critiques were presented by the system and the user could only select *one* to apply). Later systems, such as *QwikShop*, *EasyShop* and *CritiqueShop*, offer the user a little more interfacing control through the provision of alternate (i.e., unit/compound) critiquing modes to choose from, but do not allow the user to directly revise their preference model (e.g., *Apt Decision*), or set explicit constraint boundaries on certain features (e.g., as is the case in *SmartClient*).

Chen and Pu [11] demonstrate how their hybrid interface supports both *user-motivated* and *system-proposed critiquing* and enables users to achieve a higher

⁶ The DiamondTouch product line has moved operations from the MERL research lab and into a separate company called Circle Twelve Inc.

level of decision accuracy and interfacing satisfaction, while consuming less cognitive effort, than that of a standalone system-proposed critiquing interface. Instead of suggesting pre-computed critiques for users to choose, the self-motivated critiquing approach focuses on showing examples and stimulating users to make unit or compound critique selections over any combination of features as they wish. Importantly, users also have the freedom to specify their tradeoff criteria in terms of improvement and compromise regarding the individual features, and see a new set of products better approaching their ideal choice (see previous Fig. 13.5). McSherry [38] seeks a middle-ground by introducing the notion of direct, user-motivated *relaxation* critiques in addition to system-generated relaxation mechanisms in a progressive critiquing recommender. This mixed-initiative approach implemented by the *Tweak 2* system allows the user to request an item *like* the current recommendation but with no restriction on the value of a particular (previously critiqued) feature of their choosing.

Mobile platforms present the user with a very limited control and interfacing license [42]. In addition, telecommunication service costs tend to discourage users from engaging in lengthy interaction sessions. In this space it is usual to measure efficiency by the number of clicks, scrolls, and keypad actions, the user needs to perform. User-control is often sacrificed in view of keeping costs low (i.e. time and money). Two key goal requirements that influenced the adoption of critiquing within MobyRek as a suitable direct feedback mechanism were: (1) useful preference information needed to be captured within a very small screen area, and (2) the user-system interaction had to be low-cost; that is, requiring minimal time to obtain a useful recommendation. Evaluations demonstrate that users are typically recommended acceptable recommendation options⁷ within 2-3 cycles/clicks [42, 61].

13.4.3 Supporting Explanation, Confidence & Trust

There are a number of Chapters in this handbook that are relevant to this heading (see for example, Chapters 20, 14, 15, 16, and 25). A recommender system's ability to establish trust with users and convince them of its recommendations, such as which camera or PC to purchase, is highlighted as a crucial design factor [51, 52]. If users perceive it to be capable and efficient at assisting them to make decisions, they are more likely to return to the interface. A number of researchers have developed design principles and strategies for building trust in critique-based recommenders through the explanation of recommendations to users (e.g., see [52, 57, 66]). Generally a product recommender may use explanations to explain: (1) the reasons *why* a particular product was (or *was not*) recommended [36], and/or (2) *what* opportunities remain: that is, "*where can I get to from here*", when presented with an unsuitable recommendation [57]. In the first instance, explaining why a product was recommended can be simply managed by showing the user the information con-

⁷ For the purpose of the evaluation a mobile restaurant application is presented whereby location information is collected using the GPS receiver connected via Bluetooth



Fig. 13.13: The MobyRek Mobile Restaurant Recommender [61].

tained in the user model, as is the case in with *Apt Decision* (see Fig.13.6) by way of *justification* [66]. McSherry [36, 37] highlights the greater importance of explaining the cause of any retrieval failure when a recommendation does not satisfy a user's requests. He describes a mixed-initiative approach to recovery from retrieval failure by highlighting subsets of query features that cannot be satisfied such that the user might revise their constraint boundaries (e.g. "there are no cameras with price less than €300 and resolution greater than 4 mega-pixels").

Reilly *et. al.* [57] argue that dynamic compound critiques help the user to better understand the *recommendation opportunities* that exist beyond the current cycle by helping them to appreciate common interactions between features (i.e., explanation through *increased system transparency* [66]). In many recommender domains, where the user is likely to have incomplete knowledge about the finer details of the feature-space, compound critiques will help to effectively map out this space and minimise decision error. For instance, with standard critiquing in the digital camera domain a user might naively select the $[Price, <]$ unit critique in the mistaken belief that this may deliver a cheaper camera that satisfies all of their other requirements. However, reducing price in this way may lead to a reduction in resolution that the user might not find acceptable and, as a result, they will have to backtrack. Hadzic & O'Sullivan [18] highlight a further potential problem here - that is, critiques suffer from a lack of symmetry that may prove to be counter-intuitive to a user. So, attempting to undo a critique by applying its opposite may not work like using the *back* button on a web-browser. This problem is less likely to occur if the compound critique $\{[Price, <], [Resolution, <]\}$ is presented because the user will come to understand the implications of a price-drop prior to selecting any critique. In addition, *QwikShop* reserves an area of the interface which provides further explanation support. The user is given information about the number of products that

relate to each presented critique option and the feature value ranges for each that it covers as illustrated by Fig. 13.2.

Providing system transparency through explanation allows users to more confidently assess the reliability of a system (i.e. increased user-confidence). Complementary to the concept of explanation is the concept of *system confidence* in a recommendation. In order to fully maximise user confidence in a recommendation, the system itself should be capable of assessing its own confidence, or lack thereof, in the recommendation. Reilly *et. al* [59] propose a methodology for modeling confidence at the system-level designed to work with critique-based recommenders. By informing the user of how confident the system itself is in a recommendation, the user can better judge how much trust to place in the recommendation. They propose a feedback influenced model that calculates measures of system confidence at both the feature and product-levels. A low system confidence score for the *price* feature, for example, would translate that the system is uncertain if the recommendation focus is concentrating on the correct price range for that user. Once users have a clearer understanding of those features needing clarification, they might be more inclined to refocus their feedback (i.e, refine and improve their preference model). The authors also demonstrate how product-level confidence scores supplement existing similarity knowledge, in order to guide the recommender towards more confident suggestions.

13.4.4 Visualisation, Adaptivity, and Partitioned Dynamicity

While Chapter 17 highlights the benefits of using visual interfaces in product recommenders here we concentrate on critique-based product recommenders. Zhang *et. al.* [72] introduce a visual interface where compound critiques are represented by meaningful icons as opposed to through plain text. Fig. 13.14 shows an example of how a single compound critique is represented by both approaches. Their studies demonstrate that users are more likely to apply visual compound critiques over the textual form (i.e., recording application frequency improvements of nearly 50%), and subsequently benefit from reduced interaction times (e.g., reductions in session length of up to 53%)⁸. Fig. 13.11 illustrates how the visual interface proposed by Zhang *et. al* operates on the iPhone.

Other research has demonstrated the power of highly adaptive visual interfaces that support the dynamic change of interface icons in response to user critiquing feedback. Specifically, Averjanova *et. al.* [2] demonstrate how recommendation effectiveness (e.g., average session length reductions of 17%) and user satisfaction can be improved in the *MobyRek* system [61] through the integration of a map-based visualisation interface. Similarly, the usability studies of *CATS* group recommender [30] show that the dynamic adaptive interface promotes the mutual awareness of changing multi-user preferences. A key design consideration common to both sys-

⁸ Results refer to data gathered using a version of the system that operated over a laptop dataset.

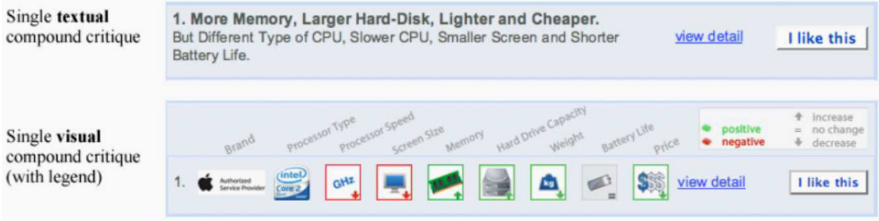


Fig. 13.14: Illustrating textual and visual representations of a single compound critique (from [72]).

tems was how best to address the problem of *limited critique influence on recommendations* whereby it was hard for the user to see the effect of a critique. This problem was identified in both cases through real-user usability evaluations. Both systems address this in a very similar way though the the use of dynamic interfacing components (i.e., icon resizing and colour coding recommendations). For example, in CATS if the level of interest in a particular holiday resort is high amongst the group then this resort is resized to be larger than those that are of lesser interest. Another concern reported by users of the early MobyRek interface was *limited distance perception* whereby it was difficult for users to compare distances to restaurants from their current position as they were on the move. A further extension in the revised *MapMobyRek System* included a map-based interface to address this problem. Other extensions included colour-coding to represent the degree of suitability of recommendation and the functionality to support side-by-side item comparisons.



Fig. 13.15: Illustrating the visualisation interface of the MapMobyRek Mobile Recommender [2].

Like *MobyRek*, *CATS* implements a map-based interface, however an additional design consideration was how best to communicate progress towards consensus

agreement to all group members. This is effectively achieved through the use of highly visual color-coded consensus barometers that summarize evolving user opinions on competing candidate vacation options. A further concern when designing adaptive interfaces that change rapidly *in-session* is the risk of user confusion. *Qwik-Shop*, *EasyShop*, *CritiqueShop* partition the dynamic and static elements of their interfaces [69]. This was also a key design consideration in the *CATS* recommender where it was necessary to keep members aware of each other's preferences and motivational orientations, without confusion. This is managed through the use of intuitive and distinct, *shared* and *individual* spaces and careful design of visual cues. In addition, a range of dynamic interfacing components are introduced to communicate information about evolving group preferences and monitor progress towards reaching a group consensus.

13.4.5 Respecting Multi-cultural Usability Differences

It is well documented that aspects of a user interface that are appropriate for one culture may not be suitable for another (see for example, [21], [23], [45]). In recent related work, Chen and Pu present a comprehensive cross-cultural evaluation of web-based critiquing interfaces [13]. Specifically, they compare user responses to two strategies for displaying e-commerce (i.e. laptop) recommendations: (1) as a ranked ordered list of items where each item has an explanation as to why it was retrieved, and (2) as a *preference-based organization*, whereby groups of recommendations are categorised and summarized in terms of their collective differences/trade-offs relative to the top ranked product. Very briefly, subjective evaluations over 120 participants (60 western culture/60 oriental culture) have shown that organisational view had the most impact on all users, in terms of how they perceived recommendation quality and their overall satisfaction. For a more comprehensive breakdown of comparisons regarding subjective perceptions please refer to [13].

13.5 Evaluating Critiquing: Resources, Methodologies and Criteria

In this section we first point to some of the resources that have been commonly used for evaluation purposes in this area, and provide details of where they can be accessed. Next, we summarise the typical evaluation methodology and outline some of the key evaluation criteria that are commonly used.

13.5.1 Resources & Methodologies

Those starting out in this area should be aware that a large number of the datasets that have been used for evaluation purposes in the published research are freely available for download⁹. Examples include the *Travel* (e.g., as used by [26, 58, 34]), *PC* (e.g., as used by [58, 39, 40, 34]), *Whiskey* (e.g. as used by [34]), and *Digital Camera* (used by [27]) datasets. The *Entrée* data is also freely accessible¹⁰. In other work, apartment [55, 73] and ski-holiday [29] datasets have also been used but these are not publicly available.

In general, most evaluation methodologies that are common to recommender systems evaluation can be applied here (see Chapters 8 and 15). In the ideal case, *live* usability studies and performance evaluations should be ultimately carried with real users when evaluating the performance (both subjective and objective) of critique-based recommenders. However, it can be difficult to recruit sufficient numbers of volunteers to participate in multiple trials. As a solution here, it is common for simulated studies to be conducted (the results of which are expected to be later validated in a real-user setting) across a range of different datasets (such as those mentioned earlier), and artificial user profiles. If sufficiently well designed, these off-line simulations can be reliable indicators of real-world performance. A common methodology that has been widely adopted in the literature on critique-based research when conducting performance evaluations is the *leave-one-out methodology* [33]. Very briefly, by this methodology, each product of a dataset is set as a recommendation *target* and is temporarily removed from the dataset. A subset of its features is chosen as the initial query. In each recommendation cycle the critique applications are applied such that they concur with the features of the *target*. The simulated recommendation session ends when the most similar product to the ideal product is recommended. Importantly, experimental setups should always have a corresponding *control* setup (e.g., using an alternate retrieval strategy, or feedback approach) to allow for the appropriate assessment of the significance of results. Unfortunately, the only acceptable way to evaluate usability is through real-user interaction trials.

13.5.2 Evaluation Criteria

Common objective performance measures that have been used by evaluations of critiquing research include:

Efficiency: Positively demonstrated by reductions in session length (i.e., the number of critiquing cycles/interactions) that a user engages in before they find their target product (see [39, 40, 73, 38, 72, 13, 61, 58, 60, 2, 27]) , the amount of time taken to reach a *target* recommendation [10, 13, 2, 68].

⁹ URL - <http://cbrwiki.fdi.ucm.es/wiki/index.php/CaseBases>

¹⁰ URL - <http://kdd.ics.uci.edu/databases/entree.data.html>

Accuracy: There are many ways to measure correctness. Some of these include:

Critique Prediction Accuracy: Refers to the number of times a presented critique *matches* that which was selected by a real user [12, 60, 6].

Critique Application Frequency: Refers to the proportion of the time critiques are applied by the user (i.e., their relevance to the preference model) [73, 72, 60, 11, 27].

Recommendation Accuracy: Measured by Chen & Pu in [12] as the % of cycles where following the application of an applied critique the *target* recommendation was *reachable* (i.e., located in the recommended products), and by [61, 40, 37, 68] as the % of successful sessions.

Decision Accuracy: Refers to the proportion of the time where the recommendation that was ultimately accepted by the user was actually the best solution. Measured in [9, 11] as % of cycles where the user changed their mind once they were shown all the alternatives. Measured by [44] as the degree of position displacement.

Interaction Effort: Usually refers to average session *click-distance* or the number of interactions (i.e., critiques) to arrive at a given target. Assuming one critique is applied in each cycle will be equivalent to measuring session length [12, 10, 13, 11].

Real-user usability performance evaluations of critiquing research has commonly surveyed participants and sought their subjective feedback on a *Likert* scale over criteria such as: (see for example, [9, 10, 62, 61, 13, 60, 26, 11, 2]).

System Design:

System Transparency: Does the user understand why recommendations were made?
 User Control: Did the user feel that they had control over the specification over their preferences throughout the interaction?

Competence

Perceived Ease of Use: Did the user find the system easy to use?
 Perceived Usefulness: Did the user feel that the system was useful (e.g., relevant recommendations, good explanations, etc.).
 Decision Confidence: How confident was the user that they found the *best* product for them?
 Perceived Effort: How easily did they find the information they were looking for?
 Perceived Accuracy: Where the suggestions accurate?

Trust

Satisfaction: How satisfied was the user with the interaction?
 Recommendation trust: Did the user feel that the recommender consistently provided suggestions that were suited to their preferences?

User Intention

Purchase Intention: Would the user purchase this product if given the opportunity?
 Return Intention: Would the user return to use the system (over another)?

13.6 Conclusion / Open Challenges & Opportunities

In this chapter we have presented, a non-technical overview of the evolution of critiquing research over the past decade. We believe that this could serve as a useful reference for researchers starting out in this area. We have described the benefits of the approach as they apply to a number of existing critiquing systems. Key issues and challenges brought into focus by the community have been discussed in terms of how advances have been made towards: (1) automated critique presentation in view of optimising preference acquisition, and (2) the improved retrieval performance of critique-based recommenders. In addition, we have presented an overview of the various design considerations that have influenced the integration of the critiquing mode of feedback across alternate platforms and user environments. Finally, we summarised some of the popular evaluation criteria that tend to be used when evaluating critique-based systems.

Although there has been a considerable level of research activity in the area of critiquing there are still many open challenges and opportunities. There are far too many to cover here but by way of concluding the chapter we will highlight a few examples. First, the majority of existing systems have assumed that users will execute only *positive critiques* that in the direction of *preferred* recommendation items (e.g., *Show me more like item A, but cheaper*). Sometimes it may make sense for a user to apply *negative critiques*, such as *Do not show me any more like X that are in location Y* as these are arguably just as important for the system when understanding the user's needs and preferences.

There has not been a lot of work on the topic of *entry-point* decision making, whereby the focus is on what recommendations to present first (see [15, 41]). Recent work by Hadzic & O'Sullivan introduced the notion of a *critique graph* as a formal basis for reasoning about the set of products that can be *reached* using critiquing from a given recommendation. They propose that a useful basis for calculating the best *entry recommendations* to select is to use the concept of *minimum catalogue cover* (i.e., identify the set of recommendation candidates from which every other recommendation can be reached if critiqued optimally). It is important to note that not all recommendations are reachable from a given recommendation candidate. In a similar vein, the authors describe how certain recommendations can exist that are not covered by *any* other product (i.e., that are not reachable through any series of critique applications). In this situation a useful principle might be to prioritise inclusion of these in the entry set. This idea has not been implemented or explored further by existing research as yet, although it has very good promise.

The challenge of modeling user preferences gathered from critiquing feedback is still a topic that a lot of current research continues to explore. Other approaches to modeling user preferences and comparative evaluations of existing approaches would be very interesting. Also, while recent work has looked at the idea of maintaining separate long-term and short-term preference models [44], there has been no work that has concentrated on how information might be transferred and maintained between them, and what the benefits/consequences of doing this might be. The investigation of further approaches for the aggregation of multi-user preferences is

another open challenge, as very little work has been carried out in this area to date. While early work in group critique-based recommenders [29] has demonstrated the benefits of one such approach in the context of a collaborative environment, assuming synchronous feedback, other opportunities remain. Examples include, the extension of critiquing to other asynchronous, and/or non-collaborative environments (perhaps where multiple users may interact in a non co-operative fashion). Such research could lead to the adoption of critiquing as an interaction mode in application areas currently unexplored by critiquing research such as game-play, for example.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules in large databases. *Advances in Knowledge Discovery and Data Mining* pp. 307–328 (1996)
2. Averjanova, O., Ricci, F., Nguyen, Q.N.: Map-based interaction with a conversational mobile recommender system. In: *UBICOMM '08: Proceedings of the 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pp. 212–218. IEEE Computer Society, Washington, DC, USA (2008)
3. Baatarjav, E.A., Phithakkitnukoon, S., Dantu, R.: Group recommendation system for facebook. In: *OTM '08: Proceedings of the OTM Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems*, pp. 211–219. Springer-Verlag, Berlin, Heidelberg (2008)
4. Burke, R.: The wasabi personal shopper: A case-based recommender system. In: *Proceedings of the 11th National Conference on Innovative Applications of Artificial Intelligence*, pp. 844–849. AAAI Press (1999). Menlo Park, CA, USA
5. Burke, R.: A case-based reasoning approach to collaborative filtering. In: E. Blanzieri, L. Portinale (eds.) *Proceedings of the Fifth European Conference on Case-Based Reasoning, EWCBR '00*, pp. 370–379. Springer (2000). Trento, Italy
6. Burke, R.: Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction* **12**(4), 331–370 (2002)
7. Burke, R., Hammond, K., Young, B.: Knowledge-based navigation of complex information spaces. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 462–468. AAAI Press/MIT Press (1996). Portland, OR
8. Burke, R.D., Hammond, K.J., Young, B.C.: The findme approach to assisted browsing. *IEEE Expert: Intelligent Systems and Their Applications* **12**(4), 32–40 (1997)
9. Chen, L., Pu, P.: Evaluating critiquing-based recommender agents. In: *In Proc. AAAI 2006*, pp. 157–162 (2006)
10. Chen, L., Pu, P.: The evaluation of a hybrid critiquing system with preference-based recommendations organization. In: *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pp. 169–172. ACM, New York, NY, USA (2007)
11. Chen, L., Pu, P.: Hybrid critiquing-based recommender systems. In: *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces*, pp. 22–31. ACM, New York, NY, USA (2007)
12. Chen, L., Pu, P.: *Preference-Based Organization Interfaces: Aiding User Critiques in Recommender Systems*, pp. 77–86. Springer-Verlag, Berlin, Heidelberg (2007)
13. Chen, L., Pu, P.: A cross-cultural user evaluation of product recommender interfaces. In: *RecSys '08: Proceedings of the 2008 ACM Conference on Recommender systems*, pp. 75–82. ACM, New York, NY, USA (2008)

14. Cohen S., Rokach L., Maimon O., Decision Tree Instance Space Decomposition with Grouped Gain-Ratio, *Information Science*, Volume 177, Issue 17, pp. 3592-3612 (2007)
15. Faltings, B., Pu, P., Torrens, M., Viappiani, P.: Designing example-critiquing interaction. In: *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*, pp. 22–29. ACM, New York, NY, USA (2004)
16. Faratin, P., Sierra, C., Jennings, N.: Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence* **142**(2), 205–237 (2002)
17. Ha, V., Haddawy, P.: Problem-focused incremental elicitation of multi-attribute utility models. In: *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, (UAI-97), pp. 215–222. Morgan Kaufmann (1997). URL citeseer.ist.psu.edu/ha97problemfocused.html. San Francisco
18. Hadzic, T., O'Sullivan, B.: Critique graphs for catalogue navigation. In: *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pp. 115–122. ACM, New York, NY, USA (2008)
19. Hammond, K., Burke, R., Schmitt, K.: A case-based approach to knowledge navigation. In: D. Leake (ed.) *Case-Based Reasoning Experiences, Lessons and Future Directions.*, pp. 125–136. AAAI Press (1996)
20. Jameson, A.: More than the sum of its members: Challenges for group recommender systems. In: *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pp. 48–54. ACM, New York, NY, USA (2004)
21. Lee, K., Joshi, K., McIvor, R.: Understanding multicultural differences in online satisfaction. In: *SIGMIS-CPR '07: Proceedings of the 2007 ACM SIGMIS CPR conference on Computer personnel research*, pp. 209–212. ACM, New York, NY, USA (2007)
22. Linden, G., Hanks, S., Lesh, N.: Interactive assessment of user preference models: The automated travel assistant. In: C.P.A. Jameson, C. Tasso (eds.) *User Modeling: Proceedings of the Sixth International Conference*, pp. 67–78. Springer Wien (1997)
23. Lodge, C.: The impact of culture on usability: Designing usable products for the international user. In: N.M. Aykin (ed.) *Usability and Internationalization. HCI and Culture, Second International Conference on Usability and Internationalization, UI-HCII 2007, Held as Part of HCI International 2007, Beijing, China, July 22-27, 2007, Proceedings, Part I*, pp. 365–368. Springer (2007)
24. Masthoff, J., Gatt, A.: In pursuit of satisfaction and the prevention of embarrassment: Affective state in group recommender systems. *User Modeling and User-Adapted Interaction* **16**(3-4), 281–319 (2006)
25. McCarthy, J., Anagnost, T.: Musicfx: An arbiter of group preferences for computer supported collaborative workouts. In: *Proc. of Conference on Computer Supported Cooperative Work*, pp. 363–372 (1998)
26. McCarthy, K., Reilly, J., McGinty, L., Smyth, B.: On the dynamic generation of compound critiques in conversational recommender systems. In: *Adaptive Hypermedia and Adaptive Web-Based Systems, Third International Conference, AH 2004, Eindhoven, The Netherlands, August 23-26, 2004, Proceedings, Lecture Notes in Computer Science*, vol. 3137, pp. 176–184. Springer (2004)
27. McCarthy, K., Reilly, J., McGinty, L., Smyth, B.: Experiments in dynamic critiquing. In: *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pp. 175–182. ACM, New York, NY, USA (2005)
28. McCarthy, K., Reilly, J., Smyth, B., McGinty, L.: Generating diverse compound critiques. *Artif. Intell. Rev.* **24**(3-4), 339–357 (2005)
29. McCarthy, K., Salamó, M., Coyle, L., McGinty, L., Smyth, B., Nixon, P.: Cats: A synchronous approach to collaborative group recommendation. In: G. Sutcliffe, R. Goebel (eds.) *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, Melbourne Beach, Florida, USA, May 11-13, 2006*, pp. 86–91. AAAI Press (2006)
30. McCarthy, K., Salamó, M., Coyle, L., McGinty, L., Smyth, B., Nixon, P.: Group recommender systems: A critiquing based approach. In: *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pp. 267–269. ACM, New York, NY, USA (2006)

31. McCarthy, K., Salamó, M., McGinty, L., Smyth, B.: The needs of the many: A case-based group recommender system. In: ICCBR '06: Proceedings of the 11th international conference on Intelligent user interfaces, pp. 196–210. Springer LNCS (2006)
32. McGinty, L., Smyth, B.: The role of diversity in conversational systems. In: D. Bridge, K. Ashley (eds.) Proceedings of the Fifth International Conference on Case-Based Reasoning (ICCB-03). Springer (2003). Troindheim, Norway.
33. McGinty, L., Smyth, B.: Tweaking critiquing. In: Proceedings of the Workshop on Personalization and Web Techniques at the International Joint Conference on Artificial Intelligence (IJCAI-03). Morgan-Kaufmann (2003). Acapulco, Mexico
34. McGinty, L., Smyth, B.: Adaptive selection: An analysis of critiquing and preference-based feedback in conversational recommender systems. *Int. J. Electron. Commerce* **11**(2), 35–57 (2006)
35. McSherry, D.: Similarity and compromise. In: K. Ashley, D. Bridge (eds.) Case-Based Reasoning Research and Development. LNAI Vol.2689, pp. 291–305. Springer (2003)
36. McSherry, D.: Explaining the pros and cons of conclusions in cbr. In: P. Funk, P.A. González-Calero (eds.) Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings, pp. 317–330. Springer (2004)
37. McSherry, D.: Retrieval failure and recovery in recommender systems. *Artif. Intell. Rev.* **24**(3-4), 319–338 (2005)
38. McSherry, D., Aha, D.: Mixed-initiative relaxation of constraints in critiquing dialogues. In: ICCBR '07: Proceedings of the 7th international conference on Case-Based Reasoning, pp. 107–121. Springer-Verlag, Berlin, Heidelberg (2007)
39. McSherry, D., Aha, D.W.: Avoiding long and fruitless dialogues in critiquing. In: M. Bramer, F. Coenen, A. Tuson (eds.) Research and Development in Intelligent Systems XXIII. BCS Conference Series, pp. 173–186. Springer, London (2006)
40. McSherry, D., Aha, D.W.: The ins and outs of critiquing. In: M.M. Veloso (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, January 6-12, 2007, pp. 962–967 (2007)
41. Nguyen, Q., Ricci, F.: User preferences initialization and integration in critique-based mobile recommender systems. In: In Proc. 5th Int'l Workshop Artificial Intelligence in Mobile Systems (AIMS 04), p. pp. 7178 (2004)
42. Nguyen, Q., Ricci, F., Cavada, D.: Critique-based recommendations for mobile users: Gui design and evaluation. In: In Proceedings of the 3rd International Workshop on HCI in Mobile Guides, in conjunction with the 6th International Conference on Mobile Human-Computer Interaction (2004). Glasgow, Scotland.
43. Nguyen, Q.N., Ricci, F.: Replaying live-user interactions in the off-line evaluation of critique-based mobile recommendations. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 81–88. ACM, New York, NY, USA (2007)
44. Nguyen, Q.N., Ricci, F.: Long-term and session-specific user preferences in a mobile recommender system. In: IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces, pp. 381–384. ACM, New York, NY, USA (2008)
45. Noiwan, J., Norcio, A.F.: Cultural differences on attention and perceived usability: Investigating color combinations of animated graphics. *Int. J. Hum.-Comput. Stud.* **64**(2), 103–122 (2006)
46. O'Connor, M., Cosley, D., Konstan, J., Riedl, J.: PolyLens: A recommender system for groups of users. In: Proc. of European Conference on Computer-Supported Cooperative Work, pp. 199–218 (2001)
47. Park, Y.J., Chang, K.N.: Individual and group behavior-based customer profile model for personalized product recommendation. *Expert Syst. Appl.* **36**(2), 1932–1939 (2009)
48. Payne, J., Bettman, J., Johnson, E.: *The Adaptive Decision Maker*. Cambridge University Press (1993)
49. Plua, C., Jameson, A.: Collaborative preference elicitation in a group travel recommender system. In: Proceedings of the AH 2002 Workshop on Recommendation and Personalization in eCommerce, pp. 148–154. Malaga, Spain (2002)

50. Prada, R., Paiva, A.: Believable groups of synthetic characters. In: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems, pp. 37–43. The Netherlands (2005)
51. Pu, P., Chen, L.: Trust building with explanation interfaces. In: IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces, pp. 93–100. ACM, New York, NY, USA (2006)
52. Pu, P., Chen, L.: Trust-inspiring explanation interfaces for recommender systems. *Know.-Based Syst.* **20**(6), 542–556 (2007)
53. Pu, P., Faltings, B.: Personalized navigation of heterogeneous product spaces using smart-client. In: IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces, pp. 212–213. ACM, New York, NY, USA (2002)
54. Pu, P., Faltings, B.: Decision tradeoff using example-critiquing and constraint programming. *Constraints* **9**(4), 289–310 (2004)
55. Pu, P., Z., H., Kumar, P.: Evaluating example-based search tools. In: EC '04: Proceedings of the 5th ACM conference on Electronic commerce, pp. 208–217. ACM, New York, NY, USA (2004)
56. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Dynamic critiquing. In: P. Funk, P.A. González-Calero (eds.) *Advances in Case-Based Reasoning, 7th European Conference, EC-CBR 2004*, Madrid, Spain, August 30 - September 2, 2004, Proceedings, *Lecture Notes in Computer Science*, vol. 3155, pp. 763–777. Springer (2004)
57. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Explaining compound critiques. *Artif. Intell. Rev.* **24**(2), 199–220 (2005)
58. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Incremental critiquing. *Knowl.-Based Syst.* **18**(4-5), 143–151 (2005)
59. Reilly, J., Smyth, B., McGinty, L., McCarthy, K.: Critiquing with confidence. In: H. Muñoz-Avila, F. Ricci (eds.) *Case-Based Reasoning, Research and Development, 6th International Conference, on Case-Based Reasoning, ICCBR 2005*, Chicago, IL, USA, August 23-26, 2005, Proceedings, *Lecture Notes in Computer Science*, vol. 3620, pp. 436–450. Springer (2005)
60. Reilly, J., Zhang, J., McGinty, L., Pu, P., Smyth, B.: A comparison of two compound critiquing systems. In: IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces, pp. 317–320. ACM, New York, NY, USA (2007)
61. Ricci, F., Nguyen, Q.N.: Critique-based mobile recommender systems. *GAI Journal*, GAI Press **Volume 24, Number 4** (2004)
62. Ricci, F., Nguyen, Q.N.: Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intelligent Systems* **22**(3), 22–29 (2007)
63. Salamo, M., Reilly, J., McGinty, L., Smyth, B.: Improving incremental critiquing. In: In Proceedings of the 16th Conference on Artificial Intelligence and Cognitive Science (AICS05), pp. 379–388 (2005)
64. Shearin, S., Lieberman, H.: Intelligent profiling by example. In: IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces, pp. 145–151. ACM, New York, NY, USA (2001)
65. Smyth, B., McGinty, L.: Improving the Performance of Recommender Systems that Use Critiquing, chap. *Intelligent Techniques for Web Personalization*, pp. 114–132. 978-3-540-29846-5. Springer (2005)
66. Sørmo, F., Cassens, J., Aamodt, A.: Explanation in case-based reasoning-perspectives and goals. *Artificial Intelligence Review* **24**(2), 109–143 (2005)
67. Spiekermann, S., Paraschiv, C.: Motivating Human-Agent Interaction: Transferring Insights from Behavioral Marketing to Agent Design, pp. 255–285. Kluwer Academic Publishers (2002)
68. Viappiani, P., Faltings, B., Pu, P.: Preference-based search using example-critiquing with suggestions. *Journal of Artificial Intelligence Research* **27**, 2006 (2006)
69. Weld, D., Anderson, C., Domingos, P., Etzioni, O., Lau, T., Gajos, K., Wolfman, S.: Automatically personalizing user interfaces. In: Proceedings of the 18th International Joint Confer-

- ence Artificial Intelligence (IJCAI-03), pp. 1613–1619. Morgan Kaufman (2003). Acapulco, Mexico
70. Williams, M.: What makes rabbit run? *International Journal of Man-Machine Studies* **21**, 333–352 (1984)
 71. Williams, M., Tou, F.: Rabbit: An interface for database access. In: *Proceedings of the ACM'82 Conference*, pp. 83–87. ACM, New York, USA (1982)
 72. Zhang, J., Jones, N., Pu, P.: A visual interface for critiquing-based recommender systems. In: *EC '08: Proceedings of the 9th ACM conference on Electronic commerce*, pp. 230–239. ACM, New York, NY, USA (2008)
 73. Zhang, J., Pu, P.: A comparative study of compound critique generation in conversational recommender systems. In: V.P. Wade, H. Ashman, B. Smyth (eds.) *Adaptive Hypermedia and Adaptive Web-Based Systems, 4th International Conference, AH 2006, Dublin, Ireland, June 21-23, 2006, Proceedings, Lecture Notes in Computer Science*, vol. 4018, pp. 234–243. Springer (2006)

Chapter 14

Creating More Credible and Persuasive Recommender Systems: The Influence of Source Characteristics on Recommender System Evaluations

Kyung-Hyan Yoo and Ulrike Gretzel

Abstract Whether users are likely to accept the recommendations provided by a recommender system is of utmost importance to system designers and the marketers who implement them. By conceptualizing the advice seeking and giving relationship as a fundamentally social process, important avenues for understanding the persuasiveness of recommender systems open up. Specifically, research regarding the influence of source characteristics, which is abundant in the context of human-human relationships, can provide an important framework for identifying potential influence factors. This chapter reviews the existing literature on source characteristics in the context of human-human, human-computer, and human-recommender system interactions. It concludes that many social cues that have been identified as influential in other contexts have yet to be implemented and tested with respect to recommender systems. Implications for recommender system research and design are discussed.

14.1 Introduction

Recommender systems are taking on an important role in supporting online users during complex decision-making processes by providing personalized advice [7]. Yet, although recommender systems make recommendations based on often sophisticated data mining and analysis techniques, it cannot be automatically implied that the advice provided by a system will always be accepted by its users. Whether a recommendation is seen as credible advice and actually taken into account not only depends on users' perceptions of the recommendation but also of the sys-

Kyung-Hyan Yoo

William Paterson University, Communication department, 300 Pompton Road, Wayne, NJ, USA,
e-mail: toinette75@gmail.com

Ulrike Gretzel

Texas A&M University, 2261 TAMU, College Station, TX, USA, e-mail: ugretzel@tamu.edu

tem as the advice-giver. The traditional persuasion literature suggests that people are more likely to accept recommendations from credible sources [99]. Accordingly, it has recently been argued that considering the credibility of recommender systems is important in increasing the likelihood of recommendation acceptance [139, 36, 28, 56, 94]. The question is how the credibility of recommender systems can be enhanced.

Recent research regarding the persuasiveness of technology suggests that technologies can be more credible and persuasive when leveraging social aspects that elicit social responses from their human users [88, 36]. This notion emphasizes the role of recommender systems as quasi-social actors, and thus, persuasive sources of advice whose characteristics influence the perceptions of their users. Various influential source characteristics have been investigated in the traditional persuasion literature based on human-human communication. Most importantly, recent research in the context of human-computer interaction found that these source characteristics are also important when humans interact with technologies [36, 37, 91, 108]. With regards to recommender systems, some studies exist that have investigated the various influences of system characteristics when users evaluate systems as well as recommendations [106, 76, 94, 24, 105]. While these findings provide good examples of source characteristics that help to develop more credible recommender systems, still many possibly influential source characteristics have not been examined.

Consequently, this chapter seeks to provide a synopsis of credibility-related research to draw attention to source characteristics which likely play a role in recommender system credibility evaluations. For that purpose, this chapter will first give an overview of the source characteristics found influential in traditional interpersonal advice seeking relationships. Then, source characteristics which have been studied in the context of human and computer interaction and, in particular, in the recommender systems realm will be discussed. Finally, the chapter identifies research gaps in terms of source characteristics that have yet to be examined in the context of recommender systems. Overall, by exploring existing findings and identifying important knowledge gaps, this chapter seeks to provide insights for recommender system researchers as far as future research needs are concerned. It also aims at providing practical implications for recommender system designers who seek to enhance the credibility of the recommender systems they build. Note that this chapter focuses on perceptions of the recommender system as a whole. The issue of trust in those who provide the ratings used to derive recommendations is dealt with in Chapter 20 Trust and Recommendations.

14.2 Recommender Systems as Social Actors

Most existing recommender system studies have viewed recommender systems as software tools and have largely neglected their social role in the interaction with users. More recent studies, however, argue that computer applications like recommender systems need to be understood as “social actors” [108]. Nass and Moon [91]

urged that people construct social relationships with machines including computers, and apply social rules in their interactions with technology. Indeed, a good number of past empirical studies have shown that individuals form social relationships with technology and that these social relationships form the basis for interactions with the technology [38, 81, 89, 92, 100, 107].

Recent recommender system studies also support this “Computers as Social Actors” paradigm. Wang and Benbasat [134], for instance, found that users not only perceive recommender systems as having human characteristics and, consequently, treat systems as social actors, but that such social perceptions influence system evaluations. Specifically, their experiment demonstrated that recommender system users perceived human characteristics such as benevolence and integrity when they interacted with online recommendation agents. Benevolence and integrity are important dimensions of trust and the users’ trust in agents was found to significantly affect perceived usefulness of agents as well as intentions to adopt the agents. Al-Natour, Benbasat and Cenfetelli [3] also argued that online shopping assistants are perceived as social actors, and that users attribute personality and behavioral traits to them. Similarly, Bonhard and Sasse [12] insisted that understanding the social embedding of a recommendation can be a key to generating more useful, trustworthy and understandable recommendations. In addition, the findings by Aksoy et al. [2] suggest that the similarity rule is also applied when humans interact with recommender systems. The study found that a user is more likely to use a recommender agent when it generates recommendations in a way similar to the user’s decision-making process. These studies all support the need for a social focus in recommender system research. Recommender systems need to be understood as communication sources to which theories developed for human-human communication apply. One set of such theories relates to the impact of source characteristics on persuasion likelihood and outcomes. The fundamental assumption of these theories is that credible sources are more effective persuaders.

14.3 Source Credibility

Credibility is not an intrinsic characteristic of a source; rather, the decision regarding a communicator’s credibility depends on how the message recipient perceives the source [99, 77, 117]. Thus, source credibility can be defined as judgments made by a message receiver concerning the believability of a communicator [37]. Reviews of source credibility studies by McGuire [75] concluded that a more credible source is preferred and also more persuasive. A good number of past studies confirm that source credibility is positively correlated with influence on message recipients’ attitudes and behavioral intentions as well as behaviors [42, 46, 64, 118, 119].

Credibility is generally described as comprising multiple dimensions [13, 39, 102, 117]. Although the literature suggests various dimensions of credibility, most researchers agree that it is comprised of two key elements: trustworthiness and expertise [99, 37, 36, 109]. Both trustworthiness and expertise have been studied

extensively and have also been addressed in the context of recommender systems [140].

14.3.1 Trustworthiness

Trustworthiness of a source refers to aspects such as character or personal integrity [99]. Intentions are also seen as instrumental in determining the trustworthiness of a source. A source whose intent it is to persuade is perceived as less trustworthy than one without persuasive intent [102]. Consequently, trustworthiness is often described by terms such as well-intentioned, truthful, and unbiased [37]. Mayer, Davis, and Schoorman [70] conceptualized benevolence and integrity as dimensions of trustworthiness. Delgado-Ballester [26] identified reliability and intentions as important trustworthiness dimensions. Fogg [36] identified key points that affect the perceptions of trustworthiness: 1) a source is fair and unbiased; 2) a source would argue against their own interest; and 3) a source has perceived similarity. In the context of recommender systems, Xiao and Benbasat [138] propose to test benevolence and integrity of recommender systems, with benevolence being defined as the recommender system's caring about the user and acting in the user's interest, and integrity being described as the recommender system's adherence to a set of principles (e.g. honesty) that the user finds acceptable.

14.3.2 Expertise

Mayer et al. [70] describe expertise as the ability of a source to have influence in a certain domain. Fogg et al. [37] conceptualize it using terms such as knowledgeable, experienced, and competent; thus, this dimension seems to capture the perceived knowledge and skill of the source. Similarly, O'Keefe [99] referred to expertise as competence, expertness, or qualification. Fogg [36] provides many examples for cues that lead to perceptions of expertise such as labels that proclaim one as an expert, appearance cues, and documentation of accomplishments. Xiao and Benbasat [138] describe the competence of a recommender system as the system's ability, skills, and expertise to perform effectively.

14.3.3 Influences on Source Credibility

Whether a source is perceived as having expertise and being trustworthy depends to a great extent on its characteristics. Thus, source characteristics serve as important cues in human judgment. Humans are often not aware of the influence of such cues,

as they are typically processed through the peripheral rather than the central route of cognitive processing and are, therefore, not elaborated on [103].

14.4 Source Characteristics Studied in Human-Human Interactions

Hovland and his colleagues [55] argued that one of the main classes of stimuli that determine the success of persuasive attempts can be summarized as the observable characteristics of the perceived message source. They specifically identified perceptions of source credibility as a direct result of the observations of particular source cues. Not surprisingly, many researchers have since investigated various communicator characteristics which influence source credibility judgments in human-human interactions.

14.4.1 Similarity

It is unquestionably the case that perceived similarities or dissimilarities between source and audience can influence the audience's judgment of source credibility [99]. In general, homophily theory [65] states that humans like similar others. However, the relation between similarity and the dimensions of credibility appears to be complex.

14.4.1.1 Expertise Judgments

Past empirical studies show contradicting results with respect to similarity and source expertise judgments. For example, Mills and Kimble [78] found that similar others are seen as having greater expertise than dissimilar others. However, Delia [27] observed that similarity between the source and the message receiver makes the receiver see the source less as an expert. In contrast, some studies found that similarity does not make any difference in source expertise judgments [126, 6].

14.4.1.2 Trustworthiness Judgments

The perceived similarity of the message source also has varying effects on perceived trustworthiness of the communicator. O'Keefe [99] suggested that perceived attitudinal similarities can influence the receiver's liking for the source, and enhanced liking for the source is commonly accompanied by enhanced judgments of the communicator's trustworthiness. However, Atkinson et al. [6] found that ethnic similarity and dissimilarity did not influence the perceived trustworthiness of the source,

while Delia [27] observed that similarity sometimes diminished trustworthiness perceptions.

Reflecting on the complex nature of the relationship between similarity and judgments of the communicator's credibility, O'Keefe [99] noted that the effects of perceived similarities on judgments of communicator credibility depend on whether, and how, the receiver perceives these as relevant to the issue at hand. Thus, different types of similarity likely have different effects in different communication contexts.

14.4.2 Likeability

People mindlessly tend to agree with those who are seen as likable [15]. Liking refers to the affective bond that an individual may feel toward another person [122]. Research generally supports the assumption that liked communicators are more effective influence agents than are disliked communicators [30, 40, 111] and likeability has been labeled a persuasion tactic and a scheme of self-presentation [21]. O'Keefe [99] stressed enhanced liking for the source is commonly accompanied by enhanced judgments of the communicator's trustworthiness. Further, a number of studies found that similarity increases likeability [18, 19, 52].

There is also some evidence indicating that the receiver's liking of the communicator can influence judgments of the communicator's trustworthiness, although not judgments of the communicator's expertise [99, 66].

14.4.3 Symbols of Authority

Evidence presented in the persuasion literature indicates that we often embrace the mental shortcut of assuming that people who simply display symbols of authority such as titles, tailors and tone should be listened to [109, 11, 51, 41, 104]. Hofling et al. [51] found that something simple as the title "Dr." made subjects perceive a source as credible and was surprisingly effective as a compliance-gaining device. Similarly, a number of studies reported that cues like the communicator's education, occupation, training, and amount of experience influence a message receiver's perceptions of source credibility. For example, Hewgill & Miller [50] manipulated the occupations of the communicator (Professor vs. High school sophomore) for the same message and found that those subjects who were informed that the message had been written by a professor evaluated both the source and the message as significantly more credible.

Uniforms and well-tailored business suits are another recognized symbol of authority that can influence credibility judgment and bring about mindless compliance [109, 21]. The findings of Bickman [11] indicate that a person wearing a security guard's uniform who asks strangers to do things could produce significantly more compliance than a person wearing street clothes. Sebastian and Bristow [116] re-

vealed that formally dressed individuals achieved greater credibility ratings than individuals who dressed informally.

14.4.4 Styles of Speech

A number of studies exist which suggest that the style of speech can influence speaker credibility judgments. For instance, several studies have demonstrated that communicators can enhance their trustworthiness when they provide both sides of the argument - the pros and the cons - rather than arguing only in their own favor [31, 121]. Cooper, Bennett and Sukel [23] suggest that people evaluate the speaker's expertise higher when he/she spoke in complex, difficult-to-understand terms. This indicates that experts may be most persuasive when nonexperts cannot understand the details of what they are saying [109]. Several investigators have found that with increasing numbers of nonfluencies in a speech, speakers are rated significantly lower on expertise judgments [73, 14, 32, 115] and the speaking rate can also influence credibility judgments, although the evidence for this effect is not as clear as for others [98, 1, 45, 68, 63]. Also, citing sources of evidence appears to enhance perceptions of the communicator's expertise and trustworthiness [35, 72].

14.4.5 Physical Attractiveness

A number of studies have found that physically attractive communicators are more persuasive [54, 123]. Eagly et al. [29] explained that there appears to be a positive reaction to good physical appearance that generalizes to favorable trait perceptions such as a talent, kindness, honesty and intelligence. The effects of physical attractiveness are seen as influencing indirectly, especially by means of influence on the receiver's liking for the communicator [99].

14.4.6 Humor

Previous studies found effects of humor when message receivers evaluate the communicator's credibility. However, the specific effects varied across different studies. A number of studies found positive effects of humor on communicator trustworthiness judgments but rarely on judgments of expertise [20, 44, 129]. When positive effects of humor were found, the effects tended to enhance the audience's liking of the communicator and this liking helped increase perceptions of trustworthiness. In contrast, some researchers found that the use of humor can decrease the audience's liking for the communicator, the perceived trustworthiness, and even the perceived

expertise of the source when the use of humor is perceived as excessive or inappropriate for the context [17, 86, 130].

14.5 Source Characteristics in Human-Computer Interactions

It seems obvious that a computer is a tool or medium and not an actor in social life. However, media equation theory suggests that individuals' interactions with computers, television sets, and new media are fundamentally social and natural, just like interactions in real life [108]. This theory thus argues that the technologies should be understood as social actors not just tools or media. Based on this new paradigm, a growing number of studies have investigated how certain social characteristics of the technologies influence their users' perceptions and behaviors.

Similarity between a computer and its users was found to be important when computer users evaluated the computer and its contents [91, 36]. For example, Nass and Moon [91] report that computers that convey similar personality types are more persuasive. In their study, dominant participants were more attracted to, assigned greater intelligence to, and conformed more with a dominant computer compared to a submissive computer. Submissive participants reacted the same way to the submissive computer as opposed to the dominant computer, despite the essentially identical content. Nass, Isbister and Lee [90] also revealed the effects of demographic similarity. Their study found that computer users perceived computer agents as more attractive, trustworthy, persuasive and intelligent when same-ethnicity agents were presented.

Presenting authority symbols has also been identified as an influential factor when people interact with technology. Nass and Moon [91] found that a television set labeled as a specialist was perceived as providing better content than a television set labeled as a generalist. Fogg [36] also posited that computing technology that assumes roles of authority is more persuasive. He argued that websites displaying awards or third-party endorsements such as seals of approval will be perceived as more credible.

A number of studies [90, 93] argue that the demographic characteristics of computer agents influence users' perceptions. Nass et al. [93] illustrated that people apply gender and ethnicity stereotypes to computers. Specifically, their study found that people evaluated the tutor computer as significantly more competent and likeable when it was equipped with a male voice than a female voice. They also found that the female-voiced computer was perceived as a better teacher of love and relationships and a worse teacher of computing than a male-voiced computer, even though they performed identically.

In addition, the use of language such as flattery [38], apology [132] and politeness [71] has been identified as factors which make a difference in computer users' perceptions and behaviors. Further, the physical attractiveness of computer agents was found to matter. The findings by Nass, Isbister and Lee [90] indicate that computer users prefer to look at and interact with computer agents that are more attractive.

Finally, humor has also been tested in the human-computer interaction context. Morkes, Kernal and Nass [83] found that computers which display humor are rated as more likeable. Yet, findings related to greater perceptions of similarity based on humor and greater length of interaction that were found for human-human interactions could not be replicated in the human-computer context.

14.6 Source Characteristics in Human-Recommender System Interactions

If computers are seen as social actors, interactions with recommender systems should also be conceptualized as interactions that are fundamentally social. Especially systems that provide direct feedback based on explicit user inputs exhibit qualities that are generally associated with social exchanges.

In the existing recommender system literature, a number of previous studies has investigated how specific characteristics of recommender systems influence users' system evaluations. Xiao and Benbasat [138] classified the various characteristics that have been studied as being associated with either recommender system type, input, process or output design. See Chapters 8, 15 and 16.

Also with the increasing interest in and use of embodied agents in recommender systems, a growing number of studies have investigated the effects of embodied agents' characteristics. Thus, in the following subsections, these previously identified influential source characteristics will be reviewed.

14.6.1 Recommender system type

Recommender systems come in different shapes and forms and can be classified based on filtering methods, decision strategies or amount of support provided by the recommender systems [138]. A number of previous studies have discussed the advantages and disadvantages of these different types of recommender systems [5, 69, 16]. Different filtering methods were compared and it was found that hybrid recommender systems that combine collaborative filtering and content filtering are evaluated as more helpful than traditional systems that use a pure collaborative filtering technique [113, 114]. Burke [16] also confirmed that hybrid recommender systems provide more accurate predictions of users' preferences. Regarding the different decision strategies used in recommender systems, compensatory recommender systems have been suggested to lead to greater trust, perceived usefulness and satisfaction than non-compensatory recommender systems [138]. They have also been found to increase users' confidence in their product choices [33]. As far as the amount of support provided by recommender systems is concerned, Xiao and Benbasat [138] argued that needs-based systems rather than feature-based systems help users better recognize their needs and more accurately answer the preference-

elicitation questions, thus resulting in better decision quality. Needs-based systems are therefore recommended for novice users [34].

14.6.2 Input characteristics

Input characteristics of recommender systems include those cues that are related with the preference elicitation method, ease of generating new/additional recommendations and the amount of control users have when interacting with the recommender systems' preference elicitation interface [138]. A number of previous findings suggest that characteristics associated with recommender system input design influence system users' evaluations. Xiao and Benbasat [138] specifically argued that the preference elicitation method (implicit vs. explicit) influences users' evaluation of the system. They proposed that an implicit preference elicitation method leads to greater perceived ease of use of and satisfaction with the recommender system while explicit elicitation is considered to be more transparent by users and leads to better decision quality.

Allowing users more control was also found to be an influential factor when evaluating systems. West et al. [137] posited that giving more control to system users will increase their trust and satisfaction with the system. Indeed, a recent study [76] found that users who used user-controlled interfaces reported higher user satisfaction than users who interacted with system-controlled and mixed-initiative recommender systems. In addition, users of user-controlled interfaces felt that the recommender systems more accurately represented their tastes and showed the greatest loyalty to the systems. Similarly, Pereira [101] demonstrated that users showed more positive affective reactions to recommender systems when they had increased control over the interaction with the recommender system. Komiak et al. [62] also found that control over the process was one of the top contributors to users' trust in a virtual agent. Supporting the importance of user control, Wang [133] noted that more restrictive recommender systems were considered as less trustworthy and useful by their users.

In addition to control, the structural characteristics of the preference elicitation process (relevance, transparency and effort) have also been found to influence users' perceptions of the recommender system [43]. The specific study by Gretzel and Fesenmaier found that topic relevance, transparency in the elicitation process and the effort required by users to provide inputs positively influence users' perceptions of the value of the elicitation process. The findings suggest that by asking questions, the system takes on a social role and communicates interest in the user's preferences, which is seen as valuable. The more questions it asks, the greater its potential to provide valuable feedback. Also, making intentions explicit in this interaction is important. Although trust was not specifically measured, benevolence and intentions are important drivers of trust and can be implied from the importance based on transparency. Further, McGinty and Smyth [74] suggested that the conversation style of recommender systems during the input process matters. In contrast to [43], they

argued that the comparison-based recommendation approach which asks users to choose a preferred item from a list of recommended items instead of a current deep dialogue approach that asks users a series of direct questions about the importance of product features would minimize the cost to the user and maintain recommendation quality.

14.6.3 Process characteristics

Characteristics of recommender systems displayed during the recommendation calculation process appear to influence users' perceptions of the systems [138]. Such process factors include information about the search process and about the system response time. Mohr and Bitner [79] noted that system users use various cues or indicators to assess the amount of effort saved by decision aids. Indicators that inform users about the search progress help users become aware of the efforts saved by the system. The higher users' perceptions of the effort saved by decision aids the greater their satisfaction with the decision process [9]. Sutcliffe et al. [125] found that users reported usability/comprehension problems with information retrieval systems that did not provide a search progress indicator.

Influences of system response time, i.e. the time between the user's input and the system's response, have also been identified as important in a number of studies. Basartan [8] varied the response time from a simulated shopbot and found that users prefer those shopbots less that make them wait a long time before receiving recommendations. In contrast, Swearingen and Sinha [128] found that the time taken by users to register and to receive recommendations from recommender systems did not have a significant effect on users' perceptions of the system. The lengthier sign up process increased users' satisfaction with and loyalty toward the system [76]. Xiao and Benbasat [138] explained that the contradicting findings of previous studies regarding response time may depend on users cost-benefit assessments. They suggest that users do not form negative evaluations of the recommender systems when they perceive the benefits of waiting as leading to high quality recommendations. The findings of [43] regarding the relationship between elicitation effort and the perceived value of the elicitation process support this assumption.

14.6.4 Output characteristics

Recommender system characteristics portrayed in the output stage of the recommendation process are related to the content and the format of the recommendations presented to users. Previous findings indicate that the content and the format of recommendations can have significant impact on users' evaluations of recommender systems [135, 138, 120, 24]. Xiao and Benbasat [138] noted that three aspects of recommendation contents - the familiarity of the recommended option, the amount

of information on recommended products, and the explanation on how the recommendation was generated - are especially relevant when users evaluate recommender systems.

Some studies found that more familiar recommendations increase users' trust in the recommender system. Sinha and Swearingen [120] found that recommended products that were familiar to users were helpful in establishing users' trust in recommender systems. A study by Cooke et al. [22] also observed that unfamiliar recommendations lowered users' evaluations of recommender systems. Further, the availability of product information appeared to positively influence users' perceptions of recommender systems. Sinha and Swearingen [120] suggest that detailed product information available on the recommendation page enhances users' trust in the recommender system. Cooke et al. [22] also explained that the attractiveness of unfamiliar recommendations can be increased if recommender systems provide detailed information about the new product.

The impacts of explanations on users' evaluations of recommender systems have been investigated in a considerable number of studies. Wang and Benbasat [135] found that explanations of the recommender system's reasoning logic strengthened users' beliefs in the recommender system's competence and benevolence. Herlocker et al. [48] also reported that explanations were important in establishing trust in systems since users were less likely to trust recommendations when they did not understand why certain items were recommended to them. Recommender systems must establish a connection between the advice seeker and the system through explanation interfaces in order to enhance the user's level of trust in the system. Similarly, other studies [105, 131] showed that system users exhibited more trust in the case of explanation interfaces.

The format in which recommendations are presented to the user also appears to influence users' evaluation of recommender systems. Sinha and Swearingen [120] found that navigation and layout of recommendation presentation interfaces significantly influence users' satisfaction with the systems. Swearingen and Sinha [128] further found that interface navigation and layout influenced users' overall rating of the systems. Consistent with these findings, Yoon and Lee [142] showed that interface design and display format influenced system users' behaviors. However, a study conducted by Bharti and Chaudhury [10] did not find any significant influence of navigational efficiency on users' satisfaction.

In addition, Schafer [112] suggested that merging the preferences interface and the recommendation elicitation interface within a single interface can make the recommender system be seen as more helpful since this new "dynamic query" interface can provide immediate feedback regarding the effect caused by individual's preference changes. Since this merges the input with the output interface, this suggestion touches upon cues such as transparency already discussed in the context of input characteristics.

14.6.5 Characteristics of embodied agents

Recommender systems often include virtual personas guiding the user through the process. It can be assumed that social responses are even more prevalent if the system is personified. Indeed, the important role and impacts of embodied interface agents in the context of recommender systems have recently been emphasized in a number of studies. For example, the presence of a humanoid virtual agent in the system interface was found to increase system credibility [85], to augment social interactions [106], to enhance the online shopping experience [53], as well as to induce trust [136]. With growing interests in such interface agents, a number of studies has started investigating if and how certain characteristics of the interface agent influence recommender system users' perceptions and evaluations.

One of the important identified characteristics of agents is anthropomorphism. Anthropomorphism is defined as the extent to which a character has either the appearance or behavioral attributes of a human being [96, 97, 60, 95]. Many researchers have found that anthropomorphism of embodied agents influences people's interactions with computers (e.g. [96, 60, 95]) and specifically with recommender systems [106]. Yet, the benefits and costs of anthropomorphic agents are debatable. For example, more anthropomorphic interface agents were rated as being more credible, engaging, attractive and likeable than less anthropomorphic agents in some studies [97, 60] while other studies found contrasting results [96, 95, 87]. The social cues communicated by the inclusion of such agents might create expectations in the users that cannot be met by the actual system functionalities.

Human voice is a very strong social cue that has been found to profoundly shape human-technology interactions [88]. However, findings in the context of embodied interface agents are not widely available and are currently inconclusive. The voice output of interface agents was found to be helpful in inducing social and affective responses from users in some studies [106, 82] but other studies found that sociability is higher when the system avatar only communicated with text [124]. The demographic characteristics of interface agents have also been found to influence system users' perceptions and behaviors. Qiu [106] reports that system users evaluated the system as more sociable, competent, and enjoyable when the agents were matched with them in terms of ethnicity and gender, thus supporting the homophily hypothesis. Cowell and Stanny [25] also observed that system users prefer to interact with interface characters that matched their ethnicity and were young looking. A study by Nowak and Rauh [97] indicated that people showed a clear preference for characters that matched their gender.

In addition to similarity cues, other source characteristics have also been investigated in the context of embodied interface agents. The effects of attractiveness and expertise of interface agents were tested by Holzwarth et al. [53]. They found that an attractive avatar is a more effective sales agent at moderate levels of product involvement while an expert agent is a more effective persuader at high levels of product involvement. Further, the potential impacts of non verbal behavior cues including facial expression, eye contact, gestures, para-language and posture of in-

terface agents were emphasized by Cowell and Stanney [25]. However, research in this area is currently very limited.

14.7 Discussion

Swearingen and Sinha [128] noted that the ultimate effectiveness of a recommender system depends on factors that go beyond the quality of the algorithm. Nevertheless, recommender system features are oftentimes implemented because they can be implemented. They might be tested in the course of overall system evaluations or usability studies but are rarely assessed in terms of their persuasiveness. Häubl and Murray [47] demonstrated that recommender systems can indeed have profound impacts on consumer preferences and choice beyond the immediate recommendation. Thus, conceptualizing recommender systems not only as social but also as persuasive actors is crucial in understanding their potential impacts. The above review of the literature suggests a wide array of recommender system characteristics which could be influential. Following the paradigm of “Computers as Social Actors” [108, 36], recent recommender system studies have started emphasizing the social aspects of recommender systems and stress the importance of integrating social cues to create more credible and persuasive systems [134, 106, 3]. This recognition of recommender systems as social actors has important implications for recommender systems research and design. Most importantly, conceptualizing human-recommender system interactions as social exchanges means that important source characteristics identified as influential in traditional advice seeking relationships can also be seen as potentially influential in human-recommender system interactions.

14.8 Implications

Understanding the influence of source characteristics when evaluating recommender systems has many implications of theoretical and practical importance. From a theoretical perspective, the classic interpersonal communication theories need to be expanded in scope and applied to understand human-recommender system relationships. By applying classic theories, researchers can test and examine various aspects of human-recommender system interactions. However, the unique qualities of human-recommender interactions should be considered when applying these theories and when developing methodologies to test them. Further, while some recommender system-related research exists with respect to source characteristics, the efforts are currently not very systematic and sometimes inconclusive. Clearly, more research is needed in this area so that a strong theoretical framework can be built.

From the practical perspective, understanding recommender systems as social actors whose characteristics influence user perceptions helps system developers and designers to better understand user interactions with systems. Social interactions

thrive on trust and are also subject to persuasion. The way in which preferences are elicited, the way recommendations are derived, and the more insight users have in these processes, the greater perceptions of credibility and the greater the likelihood for a recommendation to be accepted [43]. Hybrid systems, explicit elicitation and generally giving users control over the process seem to be highly effective strategies [113, 114, 138, 16, 137, 101]. The dynamic query interface suggested by Schafer [112], which merges the preferences interface and the recommendation elicitation interface within a single interface, may be one way to help users feel that they have control over the system since the interface can provide immediate feedback regarding the effect caused by individuals' preference changes. During interaction with recommender systems, response times needs to be kept short [8] and the specifics of the search process should be communicated to users [79, 9, 125] to demonstrate the system's efforts as this will influence credibility perceptions. When generating recommendations, more familiar recommendations with detailed product descriptions [22] and explanations regarding the underlying logic of how the recommendation was generated [133, 48] would increase users' perceived credibility of the system. A good understanding of users' system use history and patterns using a sophisticated data mining technique would help the systems generate more familiar recommendations to users. Along with the text descriptions of recommended products, recommender system designers may consider providing virtual product experiences. Jiang and Benbasat [56] noted that a virtual product experience enhances consumers' product understanding, brand attitude, purchase intention as well as decreases the perceived risks. Adding virtual experiences of products enables the users not only to have a better understanding of the recommended products but also to inspire greater attention, interest and enjoyment. Recommender system designers should also pay attention to the display format of the recommendations [128, 142]. Navigational efficacy and design familiarity and attractiveness need to be considered when the recommendations are presented to users.

Most importantly, research regarding source characteristics in the context of recommender systems provides implications regarding the design of credible and persuasive recommender systems. The challenge for design is to find ways in which source characteristics such as similarity, likeability and authority can be manipulated and translated into concrete design features that fit within the context of recommender systems. For instance, presenting third party seals signaling the authority of the system can increase the overall credibility of systems. Similarity between recommender systems and users can be implemented by the use of needs-based questions that elicit users' product preferences and their choices of the decision strategies the users prefer [138]. Manipulating personalities (e.g. extraversion or introversion) of recommender systems to match with users' by varying communication style and voice characteristics was also suggested in [49, 80]. One way in which some characteristics can be more easily implemented is by adding an embodied agent to the system interface. The embodied agent serves as the representative of the system and, thus, emphasizes the social role of the system as the advice giver [141]. Voice interfaces can be another way to translate source characteristics into credibility-evoking recommender system design.

From the marketing point of view, creating credible and persuasive recommender systems is important since the recommender systems play similar roles as human salespersons in physical stores who interact with consumers and advise consumers in terms of what to buy [134, 62]. Thus creating more sociable and credible recommender systems will help marketers to enhance their e-services.

14.9 Directions for future research

While existing studies have identified and tested a number of influential source characteristics in human-recommender system advice seeking relationships, many potential characteristics suggested by general communication theories such as authority, caring, non verbal behaviors like facial expression and gestures, and humor have not been examined. Those unexamined characteristics need to be successfully implemented and also empirically tested in future recommender system studies.

The identified and tested source characteristics also need to be more precisely examined. The effects of source characteristics on judgments of source credibility are often found to be complex rather than linear in previous studies conducted in human-human advice seeking contexts [99]. Since situational factors, individual differences and product type can also play a significant role in determining the recommender system credibility, relationships will have to be specifically tested for specific recommender systems to provide accurate input for design considerations.

In addition, there can be additional source characteristics that might not be prominent in influencing advice seeking relationships among human actors but are important aspects to be considered in the realm of recommender systems. For instance, anthropomorphism of the technology has been identified as an important characteristic that influences interactions with technologies [60, 96] while it is of course not a critical characteristic in interactions among human actors. The realness of interface agents can also be considered as a potentially influential source cue. There is some evidence that users are less likely to respond socially to a poor implementation of a human-like software character than to a good implementation of a dog-like character [59]. In future research, such additional source cues need to be identified and tested.

Some of the source characteristics have been tested in isolation from another. In order to investigate interaction effects, different source cues should be tested simultaneously if it is possible. This will help with understanding the relationships among various source factors.

Overall, the literature presented in this chapter suggests that there is a great need for research in this area. It also suggests that new methodologies might have to be developed to investigate influences that happen at a subconscious level. Especially a greater emphasis on behavioral measures of recommendation acceptance seems to be warranted if the persuasiveness of recommender systems is to be evaluated.

References

1. Addington, D.W. (1971). The effect of vocal variations on ratings of source credibility. *Speech Monographs*, 38, 242-247.
2. Aksoy, L., Bloom, P. N., Lurie, N. H., & Cooil, B. (2006). recommendation agents think like people? *Journal of Service Research*, 8(4), 297-315.
3. Al-Natour, S., Benbasat, I., & Cenfetelli, R. T. (2006). The role of design characteristics in shaping perceptions of similarity: The case of online shopping assistants. *Journal of Association for Information Systems*, 7 (12), 821-861.
4. Andersen, K.E., & Clevenger, T., Jr. (1963). A summary of experimental research in ethos. *Speech Monographs*, 30, 59-78.
5. Ansari, A., Essegaier, S. and Kohli, R. (2000). Internet Recommendation Systems. *Journal of Marketing Research*, 37(3), 363-375
6. Atkinson, D.R., Winzelberg, A., & Holland, A. (1985). Ethnicity, locus of control for family planning, and pregnancy counselor credibility. *Journal of Counseling Psychology*, 32, 417-421.
7. Barwise, P., Elberse, A. and Hammond, K. (2002). Marketing and the internet: a research review. In B. Weitz, & R. Wensley (Eds), *Handbook of Marketing* (pp. 3-7), New York, NY, Russell Sage.
8. Basartan, Y. (2001). *Amazon Versus the Shopbot: An Experiment About How to Improve the Shopbots*. Ph.D. Summer Paper, Carnegie Mellon University, Pittsburgh, PA.
9. Bechwati, N. N., & Xia, L. (2003). Do computers sweat? The impact of perceived effort of on-line decision aids on consumers' satisfaction with the decision process. *Journal of Consumer Psychology*, 13, 1-2, 139-148.
10. Bharti, P., & Chaudhury, A. (2004). An Empirical Investigation of Decision-Making Satisfaction in Web-Based Decision Support Systems. *Decision Support Systems*, 37 (2), 187-197.
11. Bickman, L. (1974). The social power of a uniform. *Journal of Applied Social Psychology*, 4, 47-61.
12. Bonhard P., & Sasse M. A. (2005). I thought it was terrible and everyone else loved it - A New Perspective for Effective Recommender System Design, in *Proceedings of the 19th British HCI Group Annual Conference* (pp. 251-261), Napier University, Edinburgh, UK 5-9 September 2005.
13. Buller, D. B., & Burgoon, J. K. (1996). Interpersonal deception theory. *Communication Theory*, 6, 203-242.
14. Burgoon, J. K., Birk, T., & Pfau, M. (1990). Nonverbal behaviors, persuasion, and credibility. *Human Communication Research*, 17, 140-169.
15. Burgoon, J. K., Dunbar, N. E. & Segring, C. (2002). Nonverbal Influence. In J. P. Dillard, & M. Pfau (Eds), *Persuasion Handbook: Developments in Theory and Practice* (pp.445-473). Thousand Oaks, CA: Sage Publications.
16. Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4), 331-370.
17. Bryant, J., Brown, D., Silberberg, A.R., & Elliott, S.M. (1981). Effects of humorous illustrations in college textbooks. *Human Communication Research*, 8, 43-57.
18. Byrne, D. (1971). *The attraction paradigm*. New York: Academic Press.
19. Carli, L. L., Ganley, R., & Pierce-Otay, A. (1991). Similarity and satisfaction in roommate relationships. *Personality and Social Psychology Bulletin*, 17 (4), 419-426.
20. Chang, K.-J., & Gruner, C. R. (1981). Audience reaction to self-disparaging humor. *Southern Speech Communication Journal*, 46, 419-426.
21. Cialdini, R. B. (1994). Interpersonal Influence. In S. Shavitt, & T. C. Brock (Eds). *Persuasion: Psychological Insights and Perspective* (pp.195-217). Needhan Heights, Massachusetts, Allyn and Bacon

22. Cooke, A. D. J., Sujan, H., Sujan, M., & Weitz, B. A. (2002). Marketing the Unfamiliar: The Role of Context and Item-Specific Information in Electronic Agent Recommendations, *Journal of Marketing Research*, 39 (4), 488-497.
23. Cooper, J., Bennett, E.A., & Sukel, H. L. (1996). Complex scientific testimony: How do jurors make decisions? *Law and Human Behavior*, 20, 379-394.
24. Cosley, D., Lam, S. K., Albert, I., Konstan, J., & Riedl, J. (2003). Is seeing believing? How recommender systems influence users' opinions. In *Proceedings of CHI 2003: Human Factors in Computing Systems* (pp. 585-592). New York: ACM Press.
25. Cowell, A. J. & Stanney, K. M. (2005). Manipulation of non-verbal interaction style and demographic embodiment to increase anthropomorphic computer character credibility. *International Journal of Human-Computer Studies*, 62, 281-306.
26. Delgado-Ballester, E. (2004). Applicability of a brand trust scale across product categories: A multigroup invariance analysis. *European Journal of Marketing*, 38(5/6), 573-592.
27. Delia, J. G. (1975). Regional dialect, message acceptance, and perceptions of the speaker. *Central States Speech Journal*, 26, 188-194.
28. Dijkstra, J. J., Liebrand, W. B. G., & Timminga, E. (1998). Persuasiveness of Expert Systems. *Behaviour & Information Technology*, 17(3), 155-163.
29. Eagly, A. H., Ashmore, R. D., Makhijani, M. G., & Longo, L. C. (1991). What is beautiful is good, but: A meta-analytic review of research on the physical attractiveness stereotype. *Psychological Bulletin*, 110, 109-128.
30. Eagly, A. H. & Chaiken, S. (1975). An attribution analysis of the effect of communicator characteristics on opinion change: The case of communicator attractiveness, *Journal of Personality and Social Psychology*, 32(1), 136-144.
31. Eagly, A.H., Wood, W., & Chaiken, S. (1978). Causal inferences about communicators and their effect on opinion change. *Journal of Personality and Social Psychology*, 36, 424, 435.
32. Engstrom, E. (1994). Effects of nonfluencies on speakers' credibility in newscast settings. *Perceptual and Motor Skills*, 78, 739-743.
33. Fasolo, B., McClelland, G. H., & Lange, K. A. (2005). The Effect of Site Design and Inter-attribute Correlations on Interactive Web-Based Decisions. In C. P. Haughvedt, K. Machleit, and R. Yalch (Eds.) *Online Consumer Psychology: Understanding and Influencing Behavior in the Virtual World* (pp. 325-344). Mahwah, NJ: Lawrence Erlbaum Associates.
34. Felix, D., Niederberger, C., Steiger, P., and Stolze, M. (2001). Featur-Oriented vs. Needs-Oriented Product Access for Non-expert Online Shoppers. In B. Schmid, K. Stanoevska-Slabeva, and V. Tschammer-Zurich (Eds.) *Towards the E-Society: E-Commerce, E-Business, and E-Government* (pp. 399-406). New York: Springer.
35. Fleshler, H., Ilardo, J., & Demoretcky, J. (1974). The influence of field dependence, speaker credibility set, and message documentation on evaluations of speaker and message credibility. *Southern Speech Communication Journal*, 39, 389-402.
36. Fogg, B. J. (2003). *Persuasive technology: Using computers to change what we think and do*. San Francisco: Morgan Kaufmann.
37. Fogg, B. J., Lee, E., & Marshall, J. (2002). Interactive technology and Persuasion. In J. P. Dillard, & M. Pfau (Eds). *Persuasion handbook: Developments in theory and practice* (pp.765-797). London, United Kingdom: Sage
38. Fogg, B.J., & Nass, C. (1997). Silicon sycophants: Effects of computers that flatter, *International Journal of Human-Computer Studies*, 46(5), 551-561.
39. Gatignon, H., & Robertson, T. S. (1991). *Innovative decision processes*. Englewood Cliffs, NJ: Prentice Hall.
40. Giffen, K., & Ehrlich, L. (1963). Attitudinal effects of a group discussion on a proposed change in company policy. *Speech Monographs*, 30, 377-379.
41. Giles, H., & Coupland, N. (1991). *Language: Contexts and consequences*. Pacific Grove, CA: Brooks/Cole.
42. Gilly, M. C., Graham, J. L., Wolfenbarger, M. F., & Yale, L. J. (1998). A dyadic study of personal information search. *Journal of the Academy of Marketing Science*, 26(2), 83-100.

43. Gretzel, U., & Fesenmaier, D. R. (2007). Persuasion in Recommender Systems. *International Journal of Electronic Commerce*, 11(2), 81-100.
44. Gruner, C. R., & Lampton, W. E. (1972). Effects of including humorous material in a persuasive sermon. *Southern Speech Communication Journal*, 38, 188-196.
45. Gundersen, D.F., & Hopper, R. (1976). Relationships between speech delivery and speech effectiveness. *Communication Monographs*, 43, 158-165.
46. Harmon, R. R., & Coney, K. A. (1982). The persuasive effects of source credibility in buy and lease situations. *Journal of Marketing Research*, 19(2), 255-260.
47. Häubl and Murray (2003). Preference Construction and Persistence in Digital Marketplaces: The Role of Electronic Recommendation Agents. *Journal of Consumer Psychology*, 13 (1&2), 75-91.
48. Herlocker, J., Konstan, J. A., & Riedl, J. (2000). Explaining Collaborative Filtering Recommendations. *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, Philadelphia, PA, 241-250.
49. Hess, T. J., Fuller, M. A. & Mathew, J. (2005). Involvement and Decision-Making Performance with a Decision Aid: The Influence of Social Multimedia, Gender, and Playfulness. *Journal of Management Information Systems*, 22(3), 15-54.
50. Hewgill, M. A., & Miller, G. R. (1965). Source credibility and response to fear-arousing communications. *Speech Monographs*, 32, 95-101.
51. Hofling, C. K., Brozman, E., Dalrymple, S., Graves, N., & Pierce, C. M. (1966). An experimental study of nurse-physician relationships. *Journal of Nervous and Mental Disease*, 143, 171-180.
52. Hogg, M. A., CooperShaw, L., & Holzworth, D. W. (1993). Group prototypically and depersonalized attraction in small interactive groups. *Personality and Social Psychology Bulletin*, 19 (4), 452-465.
53. Holzwarth, M., Janiszewski, C. & Neumann, M.M. (2006). The influence of avatars on online consumer shopping behavior. *Journal of Marketing*, 70(October), 19-36.
54. Horai, J., Naccari, N., & Fatoullah, E. (1974). The effects of expertise and physical attractiveness upon opinion agreement and liking. *Sociometry*, 37, 601-606.
55. Hovland, C.I., Janis, I.L., & Kelley, H.H. (1953). *Communication and Persuasion*. New Haven, CT: Yale University Press.
56. Jiang, J. J., Klein, G., & Vedder, R.G. (2000). Persuasive expert systems: The influence of confidence and discrepancy. *Computers in Human Behavior*, 16, 99-109.
57. Jiang, Z. & Benbasat, I. (2005). Virtual Product Experience: Effects of Visual and Functional Control of Products on Perceived Diagnosticity and Flow in Electronic Shopping. *Journal of Management Information Systems*, 21 (3), 111-148.
58. Kim, B.-D., & Kim, S.-O. (2001). A new recommender system to combine content-based and collaborative filtering systems. *Journal of Database Marketing*, 8(3), 244-252.
59. Kiesler, S., Sproull, L., & Waters, K. (1996). A prisoner's dilemma experiment on cooperation with people and human-like computers. *Journal of Personality and Social Psychology*, 70(1), 47-65.
60. Koda, T. (1996). *Agents with faces: A study on the effects of personification of software agents*. Master's Thesis, Massachusetts Institute of Technology, Boston, MA, USA.
61. Komiak, S. X. and Benbasat, I. (2004). Understanding customer trust in agent-mediated electronic commerce, web-mediated electronic commerce and traditional commerce. *Information Technology and Management*, 5 (1&2), 181-207
62. Komiak, S. Y. X., Wang, W., & Benbasat, I. (2005). Trust Building in Virtual Salespersons Versus in Human Salespersons: Similarities and Differences. *e-Service Journal*, 3(3), 49-63.
63. Lautman, M. R., & Dean, K. J. (1983). Time compression of television advertising. In I. Percy & A. G. Woodside (Eds.), *Advertising and consumer psychology* (pp.219-236). Lexington, Ma: Lexington Books.
64. Lascu, D.-N., Bearden, W. O., & Rose, R. L. (1995). Norm extremity and personal influence on consumer conformity. *Journal of Business Research*, 32(3), 201-213.

65. Lazarsfeld, P., and R. K. Merton. (1954). Friendship as a Social Process: A Substantive and Methodological Analysis. In M. Berger, T. Abel, and C. H. Page (Eds), *Freedom and Control in Modern Society* (pp. 18-66). New York: Van Nostrand.
66. Levine, R. V. (2003). Whom Do We Trust? Experts, Honesty, and Likability. In R. V. Levine (Ed.), *The Power of Persuasion*, pp.29-63. Hoboken, NJ: John Wiley & Sons.
67. Liao, Q. (2005). *Empirical findings on persuasiveness of recommender systems for customer decision support in electronic commerce*. Doctoral dissertation, Mississippi State University.
68. MacLachlan, J. (1982). Listener perception of time-compressed spokespersons. *Journal of Advertising Research*, 22(2), 47-51.
69. Maes, P., Guttman, R. H., & Moukas, A.G. (1999). *Agents that Buy and Sell*, Communication of the ACM, 42(3), 81-91.
70. Mayer, R. C., Davis, J. H. & Schoorman, F. D. (1995). An integrative model of organizational trust. *Academy of Management Review*, 20, 709-734.
71. Mayer, R. E., Johnson, W. L., Shaw, E. & Sandhu, S. (2006). Constructing computer-based tutors that are socially sensitive: Politeness in educational software. *International Journal of Human-Computer Studies*, 64(1), 36-42.
72. McCroskey, J. C. (1970). The effects of evidence as an inhibitor of counter-persuasion. *Speech Monographs*, 37, 188-194.
73. McCroskey, J. C., & Mehrley, R. S. (1969). The effects of disorganization and nonfluency on attitude change and source credibility. *Speech Monographs*, 36, 13-21.
74. McGinty, L. & Smyth. B. (2002). Deep Dialogue vs Casual Conversation in Recommender Systems. In F. Ricci and B. Smyth (Eds), *Proceedings of the Workshop on Personalization in eCommerce at the Second International Conference on Adaptive Hypermedia and Web-Based Systems* (AH 2002), pp. 80-89. Universidad de Malaga, Malaga, Spain, Springer.
75. McGuire, W. J. (1968). The Nature of Attitudes and Attitude Change. In G. Lindzey and E. Aronson (Eds.). *Handbook of Social Psychology*. Reading, MA: Addison-Wesley.
76. McNee, S. M.; Lam, S. K.; Konstan, J. A.; and Riedl, J. (2003). *Interfaces for eliciting new user preferences in recommender systems*. In User Modeling 2003, LNCS 2702, 178-187. Springer.
77. Michener, H. A., DeLamater, J. D., & Myers, D. J. (2004). *Social Psychology*. Wadsworth: Thomson Learning, Inc.
78. Mills, J., & Kimble, C. E. (1973). Opinion change as a function of perceived similarity of the communicator and subjectivity of the issue. *Bulletin of the psychonomic society*, 2, 35-36.
79. Mohr, L. A., & Bitner, M. J. (1995). The Role of Employee Effort in Satisfaction with Service Transactions. *Journal of Business Research*, 32(3), 239-252.
80. Moon, Y. (2002). Personalization and Personality: Some Effects of Customizing Message Style Based on Consumer Personality. *Journal of Consumer Psychology*, 12(4), 313-326.
81. Moon, Y., & Nass, C. (1996). How "real" are computer personalities? Psychological responses to personality types in human-computer interaction. *Communication Research*, 23(6), 651-674.
82. Moreno, R., Mayer, R. E., Spires, H. A., & Lester, J. C. (2001). The case for social agency in computer-based teaching: Do students learn more deeply when they interact with animated pedagogical agents? *Cognition and Instruction*, 19(2), 177-213.
83. Morkes, J., Kernal, H. K., & Nass, C. (1999). Effects of humor in task-oriented human-computer interaction and computer-mediated communication: A direct test of SRCT theory. *Human-Computer Interaction*, 14(4),395-435.
84. Moulin, B., Irandoust, H., Belanger, M., & Desbordes, G. (2002). Explanation and argumentation capabilities: Towards the creation of more persuasive agents. *Artificial Intelligence Review*, 17, 169-222.
85. Moundridou, M., & Virvou, M. (2002). Evaluation the persona effect of an interface agent in a tutoring system. *Journal of Computer Assisted Learning*, 18(3), 253-261.
86. Munn, W. C., & Gruner, C. R. (1981). "Sick" Jokes, speaker sex, and informative speech. *Southern Speech Communication Journal*, 46, 411-418.

87. Murano, P. (2003). Anthropomorphic vs. Non-Anthropomorphic Software Interface Feedback for Online Factual Delivery. *Proceedings of the Seventh International Conference on Information Visualization*. Retrieved October, 1, 2008 from <http://portal.acm.org/citation.cfm?id=939634&dl=ACM&coll=portal>
88. Nass, C., & Brave, S. (2005). *Wired for Speech: How Voice Activates and Advances the Human-Computer Relationship*. Cambridge, MA: MIT Press.
89. Nass, C., Fogg, B.J., & Moon, Y. (1996). Can computers be teammates? *International Journal of Human-Computer Studies*, 45(6), 669-678.
90. Nass, C., Isbister, K., & Lee, E. -J. (2000). Truth is beauty: Researching embodied conversational agents. In J Cassell, J. Sullivan, S. Prevost, & E. Churchill (Eds), *Embodied conversational agents* (pp. 374-402). Cambridge, MA: MIT Press
91. Nass, C. & Moon, Y. (2000). Machines and Mindlessness: Social Responses to Computers. *Journal of Social Issues*, 56 (1), 81-103.
92. Nass, C., & Moon, Y., & Carney, P. (1999). Are respondents polite to computers? Social desirability and direct responses to computers. *Journal of Applied Social Psychology*, 29 (5), 1093-1110.
93. Nass, C., Moon, Y., & Green, N. (1997). Are computers gender-neutral? Gender stereotypic responses to computers. *Journal of Applied Social Psychology*, 27(10), 864-876.
94. Nguyen, H., Masthoff, J. & Edwards. P. (2007). Persuasive Effects of Embodied Conversational Agent Teams. *Proceedings of 12th International Conference on Human-Computer Interaction*, Beijing, China, Springer-Verlag, Berlin, 176-185
95. Nowak, K. (2004). The influence of anthropomorphism and agency on social judgment in virtual environments. *Journal of Computer-Mediated Communication*, 9 (2)
96. Nowak, K. L., & Biocca, F. (2003). The effect of the agency and anthropomorphism on user's sense of telepresence, copresence, and social presence in virtual environments. *Presence: Teleoperators and Virtual Environments*, 12(5), 481-494.
97. Nowak, K., & Rauh, C. (2005). The influence of the avatar on online perceptions of anthropomorphism, androgyny, credibility, homophily, and attraction. *Journal of Computer-Mediated Communication*, 11 (1)
98. O'Keefe, D. J. (1998). Justification explicitness and persuasive effect: A meta-analytic review of the effects of varying support articulation in persuasive messages. *Argumentation and advocacy*, 35, 61-75.
99. O'Keefe, D. J. (2002). *Persuasion: Theory & Research*. Thousand Oaks, CA: Sage Publications.
100. Parise S, Kiesler S, Sproull L, Waters K. (1999). Cooperating with life-like interface agents. *Computers in Human Behavior*, 15, 123-142.
101. Pereira, R. E. (2000). Optimizing Human-Computer Interaction for the Electronic Commerce Environment. *Journal of Electronic Commerce Research*, 1 (1), 2000, 23-44.
102. Petty, R. E., & Cacioppo, J. T. (1981). *Attitudes and persuasion: Classic and contemporary approaches*. Dubuque, IA: William C. Brown.
103. Petty, R. E., & Cacioppo, J. T. (1986). *Communication and Persuasion: Central and Peripheral Routes to Attitude Change*. New York: Springer-Verlag.
104. Pittam, J. (1994). *Voice in social interaction: An interdisciplinary approach*. Thousand Oaks, CA: Sage.
105. Pu, P. & Chen, L. (2007). Trust-inspiring explanation interfaces for recommender systems. *Knowledge-Based Systems*, 20, 542-556
106. Qiu, L. (2006). *Designing social interaction with animated avatars and speech output for product recommendation agents in electronic commerce*. Doctoral Thesis, University of British Columbia, Vancouver.
107. Quintanar, L. R., Crowell, C. R., Pryor, J. B., & Adamopoulos, J. (1982). Human-computer interaction: A preliminary social psychological analysis. *Behavior Research Methods & Instrumentation*, 14 (2), 210-220.
108. Reeves, B. & Nass, C. (1996). *The media equation: how people treat computers, television, and new media like real people and places*. New York, NY: CSLI.

109. Rhoads, K. V., & Cialdini, R. B. (2002). The Business of Influence. J. P. Dillard & M. Pfau (Eds). *Persuasion handbook: Developments in theory and practice* . (pp.513-542). London, United Kingdom: Sage
110. Rokach, L. (2008), Mining manufacturing data using genetic algorithm-based feature set decomposition, *Int. J. Intelligent Systems Technologies and Applications*, 4(1):57-78.
111. Sampson, E. E., & Insko, C. A. (1964). Cognitive consistency and performance in the autokinetic situation. *Journal of Abnormal and Social Psychology*, 68, 184-192.
112. Schafer, J. B. (2005). DynamicLens: A Dynamic User-Interface for a Meta-Recommendation System. *Workshop: Beyond Personalization 2005, IUI'05*, San Diego, CA.
113. Schafer, J. B., Konstan, J.A., & Riedl, J. (2002). Meta-Recommendation Systems: User-Controlled Integration of Diverse Recommendations. Paper presented at the 11th international Conference on Information and Knowledge Management, McLean, VA, November 2002.
114. Schafer, J.B., Konstan, J.A., & Reidl, J. (2004). The View through MetaLens: Usage Patterns for a Meta Recommendation System. *IEE Proceedings Software*.
115. Schliesser, H. F. (1968). Information transmission and ethos of a speaker using normal and defective speech. *Central States Speech Journal*, 19, 169-174.
116. Sebastian, R. J., & Bristow, D. (2008). Formal or Informal? The Impact of Style of Dress and Forms of Address on Business Students' Perceptions of Professors. *Journal of Education for Business*, 83(4), 196-201.
117. Self, C. S. (1996). Credibility. In M. B. Salwen & D. W. Stacks (Eds.), *An integrated approach to communication theory and research* (pp. 421-441). Mahwah, NJ: Lawrence Erlbaum.
118. Sénécal, S., & Nantel, J. (2003). Online influence of relevant others: A framework. *Proceedings of the Sixth International Conference on Electronic Commerce Research (ICECR-6)*, Dallas, Texas.
119. Sénécal, S., & Nantel, J. (2004). The influence of online product recommendations on consumers' online choices. *Journal of Retailing*, 80(2), 159-169.
120. Sinha, R. & Swearingen, K. (2001). Comparing Recommendations Made by Online Systems and Friends. *Proceedings of the 2nd DELOS Network of Excellence Workshop on Personalization and Recommender Systems in Digital Libraries*, Dublin Ireland, June 18-20.
121. Smith, R. E., & Hunt, S. D. (1978). Attributional processes and effects in promotional situations. *Journal of Consumer Research*, 5, 149-158.
122. Smith, D., Menon, S., & Sivakumar, K. (2005). Online peer and editorial recommendations, trust, and choice in virtual markets. *Journal of Interactive Marketing*, 19(3), 15-37.
123. Snyder, M., & Rothbart, M. (1971). Communicator attractiveness and opinion change. *Canadian Journal of Behavioural Science*, 3, 377-387.
124. Sproull, L., Subramani, M., Kiesler, S., Walker, J. H., & Waters, K. (1996). When the interface is a face. *Human-Computer Interaction*, 11(1), 97-124.
125. Sutcliffe, A.G., Ennis, M., Hu, J. (2000). Evaluating the Effectiveness of Visual User Interfaces for Information Retrieval. *International Journal of Human-Computer Studies*, 53, 741-763
126. Swartz, T. A. (1984). Relationship between source expertise and source similarity in an advertising context. *Journal of Advertising*, 13(2), 49-55.
127. Swearingen, K., & Sinha, R. (2001). Beyond Algorithms: An HCI Perspective on Recommender Systems. *Proceedings of the ACM SIGIR 2001 Workshop on Recommender Systems*, New Orleans, Louisiana
128. Swearingen, K. & Sinha, R. (2002). Interaction design for recommender systems. *Designing Interactive Systems 2002*. ACM, 2002.
129. Tamborini, R., & Zillmann, D. (1981). College students' perceptions of lecturers using humor. *Perceptual and Motor Skills*, 52, 427-432.
130. Taylor, P.M. (1974). An experimental study of humor and ethos. *Southern Speech Communication Journal*, 39, 359-366.

131. Tintarev, N. and Masthoff, J. (2007). Effective explanations of recommendations: User-centered design. *Proceedings of the ACM Conference on Recommender Systems*, Minneapolis, US, 153-156.
132. Tzeng, J. -Y. (2004). Toward a more civilized design: Studying the effects of computers that apologize. *International Journal of Human-Computer Studies*, 61(3), 319-345.
133. Wang, W. (2005). *Design of Trustworthy Online Recommendation Agents: Explanation Facilities and Decision Strategy Support*. Doctoral Thesis, University of British Columbia, Vancouver.
134. Wang, W., & Benbasat, I. (2005). Trust in and adoption of online recommendation agents. *Journal of the Association for Information Systems*, 6(3), 72-101.
135. Wang, W., & Benbasat, I. (2007). Recommendation Agents for Electronic Commerce: Effects of Explanation Facilities on Trusting Beliefs, *Journal of Management Information Systems*, 23 (4), 217-246.
136. Wang, Y. D. & Emurian, H. H. (2005). An overview of online trust: Concepts, Elements and Implications. *Computers in Human Behavior*, 21(1), 105-125.
137. West, P. M., Ariely, D., Bellman, S., Bradlow, E., Huber, J., Johnson, E., Kahn, B., Little, J., & Schkade, D. (1999). Agents to the rescue? *Marketing Letters*, 10 (3), 285-300
138. Xiao, B., & Benbasat, I. (2007). E-Commerce product recommendation agents: Use, characteristics, and impact. *MIS Quarterly*, 31(1), 137-209.
139. Yoo, K. -H. (2008). *Creating more credible and likable recommender systems*. Dissertation Proposal. Texas A&M University.
140. Yoo, K. -H. & Gretzel, U. (2008). The influence of perceived credibility on preferences for recommender systems as sources of advice. *Information Technology & Tourism*, 10(2), 133-146.
141. Yoo, K. -H, & Gretzel, U. (2009). The Influence of Virtual Representatives on Recommender System Evaluation, *Proceedings of the 15th Americas Conference on Information Systems*, San Francisco, California
142. Yoon, S. N., & Lee, Z. (2004). The impact of the Web-based Product Recommendation Systems from Previous Buyers on Consumers' Purchasing Behavior. Paper presented at the tenth Americas Conference on Information Systems, New York, New York, August, 2004.

Chapter 15

Designing and Evaluating Explanations for Recommender Systems

Nava Tintarev and Judith Masthoff

Abstract This chapter gives an overview of the area of explanations in recommender systems. We approach the literature from the angle of evaluation: that is, we are interested in what makes an explanation “good”, and suggest guidelines as how to best evaluate this. We identify seven benefits that explanations may contribute to a recommender system, and relate them to criteria used in evaluations of explanations in existing systems, and how these relate to evaluations with live recommender systems. We also discuss how explanations can be affected by how recommendations are presented, and the role the interaction with the recommender system plays w.r.t. explanations. Finally, we describe a number of explanation styles, and how they may be related to the underlying algorithms. Examples of explanations in existing systems are mentioned throughout.

15.1 Introduction

In recent years, there has been an increased interest in more user-centered evaluation metrics for recommender systems such as those mentioned in [42]. It has also been recognized that many recommender systems functioned as black boxes, providing no transparency into the working of the recommendation process, nor offering any additional information to accompany the recommendations beyond the recommendations themselves [29].

Explanations can provide that transparency, exposing the reasoning and data behind a recommendation. This is the case with some of the explanations hosted on Amazon, such as: “*Customers Who Bought This Item Also Bought ...*”. Expla-

Nava Tintarev
University of Aberdeen, Aberdeen, U.K. e-mail: n.tintarev@abdn.ac.uk

Judith Masthoff
University of Aberdeen, Aberdeen, U.K. e-mail: j.masthoff@abdn.ac.uk

nations can also serve other aims such as helping to inspire user trust and loyalty, increase satisfaction, make it quicker and easier for users to find what they want, and persuade them to try or purchase a recommended item. In this way, we distinguish between different explanation such as e.g. explaining the way the recommendation engine works (transparency), and explaining why the user may or may not want to try an item (effectiveness). An effective explanation may be formulated along the lines of “*You might (not) like Item A because...*”. In contrast to the Amazon example above, this explanation does not *necessarily* describe how the recommendation was selected - in which case it is not transparent.

This chapter offers guidelines for designing and evaluating explanations in recommender systems as summarized in Section 15.2. Expert systems can be said to be the predecessors of recommender systems. In Section 15.3 we therefore briefly relate research on evaluating explanations in expert systems to evaluations of explanations in recommender systems. We also identify the developments in recommender systems which may have caused a revived interest in explanation research since the days of expert systems.

Up until now there has been little consensus as to how to evaluate explanations, or why to explain at all. In Section 15.4, we list seven explanatory criteria, and describe how these have been measured in previous systems. These criteria can also be understood as advantages that explanations may offer to recommender systems, answering the question of *why* to explain. In the examples for effective and transparent explanations above, we saw that the two evaluation criteria could be mutually exclusive.

In Section 15.5, we consider that the underlying recommender system affects the evaluation of explanations, and discuss this in terms of the evaluation metrics normally used for recommender systems (e.g. accuracy and coverage). We mention and illustrate examples of explanations throughout the chapter, and offer an aggregated list of examples in commercial and academic recommender systems in Table 15.6. We will see that explanations have been presented in various forms, using both text and graphics.

Additionally, explanations are not decoupled from recommendations themselves or the way in which users can interact with the recommender system: both factors influence each other and the explanations that can be generated, which in turn affects the degree to which explanatory goals are achieved. We discuss these types of design choices in Section 15.6 – in Section 15.6.1 we mention different ways of presenting recommendations, and Section 15.6.2 how users can interact and give input to a recommender system.

Moreover, the underlying algorithm of a recommender engine may influence the types of explanations that can be generated, although it is also possible that the explanations selected by the system developer do *not* reflect the underlying algorithm. This is particularly the case for computationally complex algorithms for which explanations may be more difficult to generate, such as collaborative filtering [29, 31]. In this case, the developer must consider the trade-offs between different explanatory goals such as satisfaction (as an extension of understandability) and transparency. In Section 15.7, we relate the most common explanation styles and

how they may relate to the underlying algorithms. Finally, we conclude with a summary and future directions in Section 15.8.

15.2 Guidelines

The content of this chapter is divided into sections which each elaborate on the following design guidelines for explanations in recommender systems.

- Consider the benefit(s) you would like to obtain from the explanations, and the best metric to evaluate on the associated criteria (Section 15.4).
- Be aware that the evaluation of explanations is related to, and may be confounded with, the functioning of the underlying recommendation engine, as measured by criteria commonly used for evaluating recommender systems (Section 15.5).
- Think about how the way that you present the recommendations themselves, and the the interaction model, affect each other and the explanations (Section 15.6). These factors in turn affect the degree to which different explanatory goals can be achieved.
- Last, but certainly not least, consider the relationship between the underlying algorithm and the type of explanations you choose to generate (Section 15.7). Do the explanations that you generate help you achieve your explanatory goals?

15.3 Explanations in Expert Systems

Explanations in intelligent systems are not a new idea: explanations have often been considered as part of the research in the area of expert systems [8, 32, 38, 27, 66]. This research has largely been focused on what kind of explanations can be generated and how these have been implemented in real world systems [8, 32, 38, 66]. Overall, *there are few evaluations of the explanations in these systems*. When they did occur evaluations of explanations have largely focused on *user acceptance* of the system such as [15] or acceptance of the systems' conclusions [67]. An exception is an evaluation in MYCIN which considered the decision support of the system as a whole [27]. In contrast, the commercial intent behind recommender systems targeting a wide user base was previously unseen in expert systems, has extended the evaluation goals for explanations beyond acceptance.

Also, developments in recommender systems have revived explanation research, after a decline of studies in expert systems in the 90's. One such development is the

increase in data: due to the growth of the Web, there are now more users using the average (recommender) system. Systems are also no longer developed in isolation of each other, making the best possible reuse of code (open source projects) and datasets such as the MovieLens [2] and Netflix dataset [3]. In addition, new algorithms, in particular in the domain of collaborative filtering, have been adapted and developed (see also Chapter 4 on neighborhood based approaches, and Chapter 5 on advances in collaborative filtering). These approaches mitigate domain dependence, and allow for greater generalizability, and are more suitable for large and often sparse datasets. One sign of the revived interest in explanation research is the success of a recent series of workshops on explanation aware computing (see e.g. [53, 54]).

For further reading, see the following reviews on expert systems with explanatory capabilities for three of the most common inference methods: heuristic-based methods [36], Bayesian networks [35], and case-based reasoning [22].

15.4 Defining Goals

Guideline 1: Consider the benefit(s) you would like to obtain from the explanations, and the best metric to evaluate on the associated criteria.

Surveying the literature on explanations in recommender systems, we see that recommender systems with explanatory capabilities have been evaluated according to different criteria, and identify seven different goals for explanations. Here we mention goals that are applicable to single item recommendations, i.e. when a single recommendation is being offered. When recommendations are made for multiple items, such as in a list, the criteria may be different and consider other factors such as diversity (e.g. are the items in the list sufficiently varied).

Table 15.1 states these goals, which are similar to those desired (but not evaluated on) in expert systems, c.f. MYCIN [10]. In Table 15.2, we summarize previous evaluations of explanations in recommender systems, and the criteria by which they have been evaluated. Works that have no clear criteria stated, or have not evaluated the system on the explanation criteria which they state, are omitted from this table.

For example, in Section 15.3 we mentioned that expert systems were commonly evaluated in terms of user acceptance and the decision support of the system as a whole. User acceptance can be defined in terms of our goals of satisfaction or persuasion. If the evaluation measures acceptance with the system as whole, such as [15] who asked questions such as “*Did you like the program?*”, then this reflects user satisfaction. If rather the evaluation measures user acceptance of advice or explanations, as in [67], the criterion can be said to be persuasion.

It is important to identify these goals as distinct, even if they may interact, or require certain trade-offs. Indeed, it would be hard to create explanations that do well

on all criteria, in reality it is a trade-off. For instance, in our work we have found that while personalized explanations may lead to greater user satisfaction, they do not necessarily increase effectiveness [61]. Other times, goals that seem to be inherently related are not necessarily so, for example it has been found that transparency does not necessarily aid trust [20]. For these reasons, while an explanation in Table 15.2 may have been evaluated for several criteria, it may not have achieved them all.

The type of explanation that is given to a user is likely to depend on the criteria of the designer of a recommender system. For instance, when building a system that sells books one might decide that user trust is the most important aspect, as it leads to user loyalty and increases sales. For selecting tv-shows, user satisfaction could be more important than effectiveness. That is, it is more important that a user enjoys the service, than that they are presented the best available shows.

In addition, some attributes of explanations may contribute toward achieving multiple goals. For instance, one can measure how *understandable* an explanation is, which can contribute to e.g. user trust, as well as satisfaction.

In this section we describe seven criteria for explanations, and suggest evaluation metrics based on previous evaluations of explanation facilities, or offer suggestions of how existing measures could be adapted to evaluate the explanation facility in a recommender system.

Table 15.1: Explanatory criteria and their definitions

Aim	Definition
Transparency (Tra.)	Explain how the system works
Scrutability (Scr.)	Allow users to tell the system it is wrong
Trust	Increase users' confidence in the system
Effectiveness (Efk.)	Help users make good decisions
Persuasiveness (Pers.)	Convince users to try or buy
Efficiency (Efc.)	Help users make decisions faster
Satisfaction (Sat.)	Increase the ease of use or enjoyment

15.4.1 Explain How the System Works: Transparency

An anecdotal article in the Wall Street Journal titled “*If TiVo Thinks You Are Gay, Here’s How to Set It Straight*” describes users’ frustration with irrelevant choices made by a video recorder that records programs it assumes its owner will like, based on shows the viewer has recorded in the past[69]. For example, one user, Mr. Iwanyk, suspected that his TiVo thought he was gay since it inexplicably kept recording programs with gay themes. This user clearly deserved an explanation.

An explanation may clarify *how* a recommendation was chosen. In expert systems, such as in the domain of medical decision making, the importance of trans-

Table 15.2: The criteria by which explanations in recommender systems have been evaluated. System names are mentioned if given, otherwise we only note the type of recommended items. Works that have no clear criteria stated, or have not *evaluated* the system on the explanation criteria which they state, are omitted from this table. Note that while a system may have been evaluated for several criteria, it may not have achieved all of them. Also, for the sake of completeness we have distinguished between multiple studies using the same system.

System (type of items)	Tra.	Scr.	Trust	Efk.	Per.	Efc.	Sat.
(Internet providers) [23]			X		X		X
(Digital cameras, notebooks computers) [49]			X				
(Digital cameras, notebooks computers) [50]			X	X			
(Music) [55]			X				
(Movies) [61]				X	X		X
<i>Adaptive Place Advisor</i> (restaurants) [59]				X		X	
<i>ACORN</i> (movies) [65]							X
<i>CHIP</i> (cultural heritage artifacts) [19]	X		X	X			
<i>CHIP</i> (cultural heritage artifacts) [20]	X		X				X
<i>iSuggest-Usability</i> (music) [30]	X			X			
<i>LIBRA</i> (books) [11]				X			
<i>MovieLens</i> (movies) [29]					X		X
<i>Movixplain</i> (movies) [58]				X			X
<i>myCameraAdvisor</i> [63]			X				
<i>Qwikshop</i> (digital cameras) [39]				X		X	
<i>SASY</i> (e.g. holidays) [21]	X	X					X
<i>Tagsplanations</i> (movies) [62]	X			X			

parency has also been recognized [10]. Transparency or the heuristic of “Visibility of System Status” is also an established usability principle [44], and its importance has also been highlighted in user studies of recommender systems [55].

Vig et al. differentiate between transparency and justification [62]. While transparency should give an honest account of how the recommendations are selected and how the system works, justification can be descriptive and decoupled from the recommendation algorithm. The authors cite several reasons for opting for justification rather than genuine transparency. For example some algorithms that are difficult to explain (e.g. latent semantic analysis where the distinguishing factors are latent and may not have a clear interpretation), protection of trade secrets by system designers, and the desire for greater freedom in designing the explanations.

Cramer et al. have investigated the effects of transparency on other evaluation criteria such as trust, persuasion (acceptance of items) and satisfaction (acceptance) in an art recommender [19, 20]. Transparency itself was evaluated in terms of its effect on actual and perceived understanding of how the system works [20]. While actual understanding was based on user answers to interview questions, perceived understanding was extracted from self-reports in questionnaires and interviews.

The evaluation of transparency has also been coupled with scrutability (Section 15.4.2) and trust (Section 15.4.3), but we will see in these sections that these criteria can be distinct from each other.

15.4.2 Allow Users to Tell the System it is Wrong: Scrutability

Explanations may help isolate and correct misguided assumptions or steps. When the system collects and interprets information in the background, as is the case with TiVo, it becomes all the more important to make the reasoning available to the user. Following transparency, a second step is to allow a user to correct reasoning, or make the system *scrutable* [21]. Explanations should be part of a cycle, where the user understands what is going on in the system and exerts control over the type of recommendations made, by correcting system assumptions where needed [56]. Scrutability is related to the established usability principle of User Control [44]. See Figure 15.1 for an example of a scrutable holiday recommender.

While scrutability is very closely tied to the criteria of transparency, it deserves to be uniquely identified. The explanations in Table 15.4 are scrutable, but not (fully) transparent even if they offer some form of justification. For example, there is nothing in this Table that suggests that the underlying recommendations are based on a Bayesian classifier. In such a case, we can imagine that a user attempts to scrutinize a recommender system, and manages to change their recommendations by modifying their ratings, but still does not understand exactly what happens within the system.

Czarkowski found that users were not likely to scrutinize on their own, and that extra effort was needed to make the scrutability tool more visible [21]. In addition, it was easier to get users perform a given scrutinization task such as changing the personalization (e.g. “Change the personalisation so that only Current Affairs programs are included in your 4:30-5:30 schedule.”) Their evaluation included metrics such as task correctness, and if users could express an understanding of what information was used to make recommendations for them. They understood that adaptation in the system was based on their personal attributes stored in their profile, that their profile contained information they volunteered about themselves, and that they could change their profile to control the personalization [21].

15.4.3 Increase Users’ Confidence in the System: Trust

Trust is sometimes linked with transparency: previous studies indicate that transparency and the possibility of interaction with recommender systems increases user trust [23, 55]. A user may also be more forgiving, and more confident in recommendations, if they understand why a bad recommendation has been made. Trust in the recommender system could also be dependent on the accuracy of the recommen-

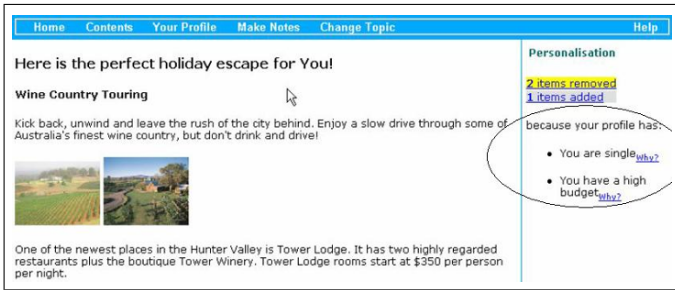


Fig. 15.1: Scrutable holiday recommender [21]. The explanation is in the circled area, and the user profile can be accessed via the “why” links.

ation algorithm [41]. A study of users’ trust (defined as perceived confidence in a recommender system’s *competence*) suggests that users intend to return to recommender systems which they find trustworthy [16]. We note however, that there is a case where transparency and trust were not found to be related [20].

We do not claim that explanations can fully compensate for poor recommendations, but good explanations may help users make better decisions (see Section 15.4.5 on effectiveness). A user may also appreciate when a system is “frank” and admits that it is not confident about a particular recommendation.

In addition, the interface design of a recommender system may affect its credibility. In a study of factors determining web page credibility, the largest proportion of users’ comments (46.1%) referred to the appeal of the overall visual design of a site, including layout, typography, font size and color schemes [25]. Likewise the perceived credibility of a Web article was significantly affected by the presence of a photograph of the author [24]. So, while recommendation accuracy, and the criteria of transparency are often linked to the evaluation of trust, design is also a factor that needs to be considered as part of the evaluation.

Questionnaires can be used to determine the degree of trust a user places in a system. An overview of trust questionnaires can be found in [45] which also suggests and validates a five dimensional scale of trust. Note that this validation was done with the aim of using celebrities to endorse products, but was not conducted for a particular domain. Additional validation may be required to adapt this scale to a particular recommendation domain.

A model of trust in recommender systems is proposed in [16, 50], and the questionnaires in these studies consider factors such as intent to return to the system, and intent to save effort. Also [63] query users about trust, but focus on trust related beliefs such as the perceived competence, benevolence and integrity of a virtual adviser. Although questionnaires can be very focused, they suffer from the fact that self-reports may not be consistent with user behavior. In these cases, implicit measures (although less focused) may reveal factors that explicit measures do not.

One such implicit measure could be loyalty, a desirable bi-product of trust. One study compared different interfaces for eliciting user preferences in terms of how

they affected factors such as loyalty [41]. Loyalty was measured in terms of the number of logins and interactions with the system. Among other things, the study found that allowing users to independently choose which items to rate affected user loyalty. It has also been thought that Amazon's conservative use of recommendations, mainly recommending familiar items, enhances user trust and has led to increased sales [57]. We encourage readers who would like to learn more about trust in recommender systems to read Chapter 20 which is dedicated to this topic.

15.4.4 Convince Users to Try or Buy: Persuasiveness

Explanations may increase user acceptance of the system or the given recommendations [29]. Both definitions qualify as persuasion, as they are attempts to gain benefit for the system rather than for the user.

[20] evaluated the acceptance of recommended items in terms of how many recommended items were present in a final selection of six favorites. In a study of a collaborative filtering- and rating-based recommender system for movies, participants were given different explanation interfaces (e.g. Figure 15.2)[29]. This study directly inquired how likely users were to see a movie (with identifying features such as title omitted) for 21 different explanation interfaces. Persuasion was thus a numerical rating on a 7-point Likert scale.

In addition, it is possible to measure if the evaluation of an item has changed, i.e. if the user rates an item differently after receiving an explanation. Indeed, it has been shown that users can be manipulated to give a rating closer to the system's prediction [18]. This study was in the low investment domain of movie rental, and it is possible that users may be less influenced by incorrect predictions in high(er) cost domains such as cameras¹. It is also important to consider that too much persuasion may backfire once users realize that they have tried or bought items that they do not really want.

Persuasiveness can be measured in a number of ways. For example, it can be measured as the difference between two ratings: the first being a previous rating, and the second a re-rating for the same item but with an explanation interface [18]. Another possibility would be to measure how much users actually try or buy items compared to users in a system without an explanation facility. These metrics can also be understood in terms of the concept of "conversion rate" commonly used in e-Commerce, operationally defined as the percentage of visitors who take a desired action. For a more in-depth discussion of persuasion in recommender systems the reader may consult Chapter 14.

¹ In [60] participants reported that they found incorrect overestimation less useful in high cost domains compared to low cost domains.

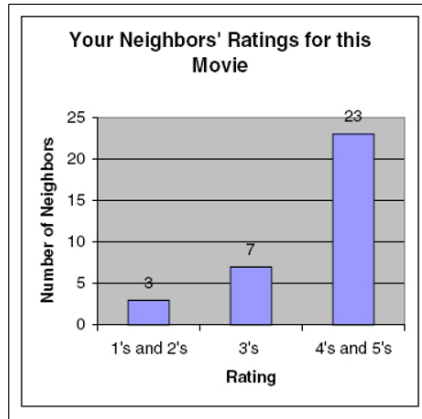


Fig. 15.2: One out of twenty-one interfaces evaluated for persuasiveness - a histogram summarizing the ratings of similar users (neighbors) for the recommended item grouped by good (5's and 4's), neutral (3's), and bad (2's and 1's), on a scale from 1 to 5 [29].

15.4.5 Help Users Make Good Decisions: Effectiveness

Rather than simply persuading users to try or buy an item, an explanation may also assist users to make *better* decisions. Effectiveness is by definition highly dependent on the accuracy of the recommendation algorithm. An effective explanation would help the user evaluate the quality of suggested items according to their own preferences. This would increase the likelihood that the user discards irrelevant options while helping them to recognize useful ones. For example, a book recommender system with effective explanations would help a user to buy books they actually end up liking. Bilgic and Mooney emphasize the importance of measuring the ability of a system to assist the user in making accurate decisions about recommendations based on explanations such as those in Figure 15.3 and Tables 15.3, 15.4 and 15.5 [11]. Effective explanations could also serve the purpose of introducing a new domain, or the range of products, to a novice user, thereby helping them to understand the full range of options [23, 49].

Vig et al. measure perceived effectiveness: “*This explanation helps me determine how well I will like this movie.*” [62]. Effectiveness of explanations can also be calculated as the *absence of a difference* between the liking of the recommended item prior to, and after, consumption. For example, in a previous study, users rated a book twice, once after receiving an explanation, and a second time after reading the book [11]. If their opinion on the book did not change much, the system was considered effective. This study explored the effect of the whole recommendation process, explanation inclusive, on effectiveness. The same metric was also used to evaluate whether personalization of explanations (in isolation of a recommender system) increased their effectiveness in the movie domain [61].

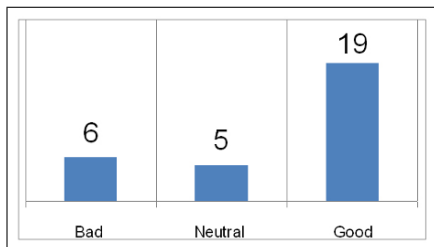


Fig. 15.3: The Neighbor Style Explanation - a histogram summarizing the ratings of similar users (neighbors) for the recommended item grouped by good (5’s and 4’s), neutral (3’s), and bad (2’s and 1’s), on a scale from 1 to 5. The similarity to Figure 15.2 in this study was intentional, and was used to highlight the difference between persuasive and effective explanations [11].

Table 15.3: The keyword style explanation by [11]. This recommendation is explained in terms of keywords that were used in the description of the item, and that have previously been associated with highly rated items. “Count” identifies the number of times the keyword occurs in the item’s description, and “strength” identifies how influential this keyword is for predicting liking of an item.

Word	Count	Strength	Explain
HEART	2	96.14	<i>Explain</i>
BEAUTIFUL	1	17.07	<i>Explain</i>
MOTHER	3	11.55	<i>Explain</i>
READ	14	10.63	<i>Explain</i>
STORY	16	9.12	<i>Explain</i>

While this metric considers the difference between the before and after ratings, it does not discuss the effects of over- contra underestimation. If a user’s evaluation of an item decreases after exposure to an item, their initial rating was an overestimation. Likewise, if their evaluation increases after exposure to the item, the initial rating was an underestimation. In our work we found that users considered overestimation to be less effective than underestimation, and that this varied between domains. Specifically, overestimation was considered more severely in high investment domains compared to low investment domains. In addition, the strength of the effect on perceived effectiveness varied depending on where on the scale the prediction error occurred [60].

Another way of measuring the effectiveness of explanations has been to test the same system with and without an explanation facility, and evaluate if subjects who receive explanations end up with items more suited to their personal tastes [19].

Other work evaluated explanation effectiveness using a metric from marketing [28], with the aim of finding the single *best* possible item (rather than “good enough items” as above) [17]. Participants interacted with the system until they found the

Table 15.4: A more detailed explanation for the “strength” of a keyword which shows after clicking on “*Explain*” in Table 15.3. In practice “strength” probabilistically measures how much more likely a keyword is to appear in a positively rated item than a negatively rated one. It is based on the user’s previous positive ratings of items (“rating”), and the number of times the keyword occurs in the description of these items (“count”) [11].

Title	Author	Rating	Count
Hunchback of Notre Dame	Victor Hugo, Walter J. Cobb	10	11
Till We Have Faces: A Myth Retold	C.S. Lewis, Fritz Eichenberg	10	10
The Picture of Dorian Gray	Oscar Wilde, Isobel Murray	8	5

item they would buy. They were then given the opportunity to survey the entire catalog and to change their choice of item. Effectiveness was then measured by the fraction of participants who found a better item when comparing with the complete selection of alternatives in the database. So, using this metric, a low fraction represents high effectiveness.

Effectiveness is the criterion that is most closely related to accuracy measures such as precision and recall [19, 58, 59]. In systems where items are easily consumed, e.g. internet news, these can be translated into recognizing relevant items and discarding irrelevant options respectively. For example, there have been suggestions for an alternative metric of “precision” based on the number of profile concepts matching with user interests, divided by the number of concepts in their profile [19].

15.4.6 Help Users Make Decisions Faster: Efficiency

Explanations may make it *faster* for users to decide which recommended item is best for them. Efficiency is another established usability principle, i.e. how quickly a task can be performed [44]. This criterion is one of the most commonly addressed in the recommender systems literature given that the task of recommender systems is to find needles in haystacks of information.

Efficiency may be improved by allowing the user to understand the relation between competing options. [39, 43, 49] use so called critiquing, a sub-class of knowledge-based algorithms based on trade-offs between item properties, which lends itself well to the generation of explanations. In the domain of digital cameras, competing options may for example be viewed by selecting “*Less Memory and Lower Resolution and Cheaper*” [39]. This way users are quickly able to use this query revision to find a cheaper camera if they are willing to settle for less memory and lower resolution. More details on critiquing-based recommender systems can also be found in Chapter 13 of this handbook.

Efficiency is often used in the evaluation of so-called conversational recommender systems, where users continually interact with a recommender system, refining their preferences (see also Section 15.6.2). In these systems, the explanations can be seen to be implicit in the dialog. Efficiency in these systems can be measured by the total amount of interaction time, and number of interactions needed to find a satisfactory item [59]. Evaluations of explanations based on improvements in efficiency are not limited to conversational systems however. Pu and Chen for example, compared completion time for two explanatory interfaces, and measured completion time as the amount of time it took a participant to locate a desired product in the interface [49].

Other metrics for efficiency also include the number of inspected explanations, and number of activations of repair actions when no satisfactory items are found [23, 52]. Normally, it is not sensible to expose users to all possible recommendations and their explanations, and so users can choose to inspect (or scrutinize) a given recommendation by asking for an explanation. In a more efficient system, the users would need to inspect *fewer* explanations. Repair actions consist of feedback from the user which changes the type of recommendation they receive, as outlined in the sections on scrutability (Section 15.4.2). Examples of user feedback/repair actions can be found in Section 15.6.2.

15.4.7 Make the use of the system enjoyable: Satisfaction

Explanations have been found to increase user satisfaction with, or acceptance of, the overall recommender system [23, 29, 55]. The presence of longer descriptions of individual items has been found to be positively correlated with both the *perceived* usefulness [60], and ease of use of the recommender system [55]. Also, many commercial recommender systems such as those seen in Table 15.6 are primarily sources of entertainment. In these cases, any extra facility should take notice of the effect on user satisfaction. Figure 15.4 gives an example of an explanation evaluated on the criterion of satisfaction.

When measuring satisfaction, one can directly ask users whether the system is enjoyable to use [15], or if users like the explanations themselves [60]. Satisfaction can also be measured indirectly by measuring user loyalty [41, 23] (see also Section 15.4.3), and likelihood of using the system for a search task [20].

In measuring explanation satisfaction, it is important to differentiate between satisfaction with the recommendation process², and the recommended products (persuasion) [20, 23]. One (qualitative) way to measure satisfaction with the process would be to conduct usability testing methods such as record a think-aloud protocol for a user conducting a task [37].

² Here we mean the entire recommendation process, inclusive of the explanations. However, in Section 15.5 we highlight that evaluation of explanations in recommender systems are seldom fully independent of the underlying recommendation process.

In this case, the participants describe their entire experience using the system: what they are looking at, thinking, doing and feeling, as they go about a task such as finding a satisfactory item. Objective notes of everything that users say are taken, without interpretation or influencing the users in any way. Video and voice recordings can also be used to revisit the session and to serve as a memory aid. In such a case, it is possible to identify usability issues and even apply quantitative metrics such as the ratio of positive to negative comments; the number of times the evaluator was frustrated; the number of times the evaluator was delighted; the number of times and where the evaluator worked around a usability problem etc.

It is also arguable that users would be satisfied with a system that offers effective explanations, confounding the two criteria. However, a system that aids users in making good decisions, may have other disadvantages that decrease the overall satisfaction (e.g. requiring a large cognitive effort on the part of the user). Fortunately, these two criteria can be measured by distinct metrics.

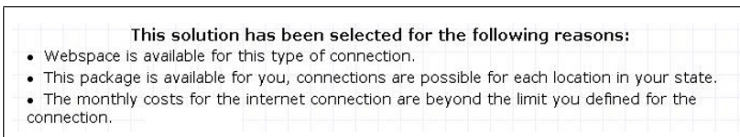


Fig. 15.4: An explanations for an internet provider, describing the provider in terms of user requirements: “This solution has been selected for the following reasons . . .” [23].

15.5 Evaluating the Impact of Explanations on the Recommender System

Guideline 2: Be aware that the evaluation of explanations is related to, and may be confounded with, the functioning of the underlying recommendation engine, as measured by criteria commonly used for evaluating recommender systems.

We have now identified seven criteria by which explanations in recommender systems can be evaluated, and given suggestions of how such evaluations can be performed. To some extent, these criteria assume that we are evaluating only the explanation component. It also seems reasonable to evaluate the system as a *whole*. In that case we might measure the general system usability and accuracy, which will depend on both the recommendation algorithm as well as the impact of the explana-

Prediction for <i>Sixth Sense, The (1999)</i>	
In Depth Prediction	MovieLens Confidence
★★★★★	★★★★★

Fig. 15.5: Confidence display for a recommendation, [29] - the movie is strongly recommended (5/5), and there is a large amount of information to support the recommendation (4.5/5).

tion component. Therefore, in this section, we describe the interaction between the recommender engine and our explanation criteria, organized by the evaluation metrics commonly used in recommender system evaluations: accuracy, learning rate, coverage, novelty/serendipity and acceptance.

15.5.1 Accuracy Metrics

Accuracy metrics regard the ability of the recommendation engine to predict correctly, but accuracy is likely to interact with explanations too. For example, with respect to the relationship between transparency and accuracy: Cramer et al. found that transparency led to changes in user behavior that ultimately decreased recommendation accuracy [19].

The system's own confidence in its recommendations is also related to accuracy and can be reflected in explanations. An example of an explanation aimed to help users understand (lack of) accuracy, can be found in confidence displays such as Figure 15.5. These can be used to explain e.g. poor recommendations in terms of insufficient information used for forming the recommendation. For further work on confidence displays see also [40].

Explanations can also help users understand how they would relate to a particular item, possibly supplying additional information that helps the user make more informed decisions (effectiveness). In the case of poor accuracy, the risk of missing good items, or trying bad ones increases while explanations can help decrease this risk. By helping users to correctly identify items as good or bad, the accuracy of the recommender system as a whole may also increase.

15.5.2 Learning Rate

The learning rate represents how quickly a recommender system learns a user's preferences, and how sensitive it is to changes in preferences. Learning rate is likely to affect user satisfaction as users would like a recommender system to quickly learn

their preferences, and be sensitive to short term as well as long term interests. Explanations can increase satisfaction by clarifying or hinting that the system considers changes in the user's preferences. For example, the system can flag that the value for a given variable is getting close to its threshold for incurring a change, but that it has not reached it yet. A system can also go a step further, and allow the user to see just how it is learning and changing preferences (transparency), or make it possible for a user to delete old preferences (scrutability). For example, the explanation facility can request information that would help it learn/change quicker, such as asking if a user's favorite movie genre has changed from action to comedy.

15.5.3 Coverage

Coverage regards the range of items which the recommender system is able to recommend. Explanations can help users understand where they are in the search space. By directing the user to rate informative items in under-explored parts of the search space, explanations may increase the overlap between certain items or features (compared to sparsity). Ultimately, this may increase the overall coverage for potential recommendations. Understanding the remaining search options is related to the criterion of transparency: a recommender system can explain why certain items are not recommended. It may be impossible or difficult to retrieve an item (e.g. for items that have a very particular set of properties in a knowledge-based system, or the item does not have many ratings in a collaborative-filtering system). Alternatively, the recommender system may function under the assumption that the user is not interested in the item (e.g. if their requirements are too narrow in a knowledge-based system, or if they belong to a very small niche in a collaborative-based system). An explanation can explain why an item is not available for recommendation, and even how to remedy this and allow the user to change their preferences (scrutability).

Coverage may also affect evaluations of the explanatory criteria of effectiveness. For example, if a user's task is not only to find a "good enough" item, but the best item for them, then the coverage needs to be sufficient to ensure that "best" items are included in the recommendations. Depending on how much time retrieving these items takes, coverage may also affect efficiency.

15.5.4 Acceptance

It is possible to confound acceptance, or satisfaction with a system with other types of satisfaction. If users are satisfied with a system with an explanation component, it remains unclear whether this is due to: satisfaction with the *explanation component*, satisfaction with *recommendations*, or general design and visual appeal. Satisfaction with the system due to the recommendations is connected to accuracy metrics,

or even novelty and diversity, in the sense that sufficiently good recommendations need to be given to a user in order to keep them satisfied. Although explanations may help increase satisfaction, or tolerance toward the system, they cannot function as a substitute for e.g. good accuracy. Indeed, this is true for all the mentioned explanatory criteria. An example of an explanation striving toward the criterion of satisfaction may be: *“Please bare with me, I still need to learn more about your preferences before I can make an accurate recommendation.”*

15.6 Designing the Presentation and Interaction with Recommendations

Guideline 3: Think about how the way that you present the recommendations themselves, and the the interaction model, affect each other and the explanations. These factors affect the degree to which explanatory goals are achieved.

The way recommendations are presented are likely to affect the interaction model that can be used for eliciting users preferences. Likewise, both factors can affect the types of explanations that can be generated. In turn, some of the explanations that can be generated may be more suitable for particular explanatory criteria. Chapter 16 of this handbook, also discusses a complementary evaluation framework for preference-based (such as critiquing which is described in Chapter 13) recommender systems and focuses on the design of both presentation of recommendations and interaction model. For example one guideline states: *“Showing one search result or recommending one item at a time allows for a simple display strategy which can be easily adapted to small display devices; however, it is likely to engage users in longer interaction sessions or only allow them to achieve relatively low decision accuracy.”* (Guideline 9).

15.6.1 Presenting Recommendations

We summarize the ways of presenting recommendations that we have seen for the systems summarized in this paper. While there are a number of possibilities for the *appearance* of the graphical user interface, the actual *structure* of offering recommendations can also vary. We identify the following categories for structuring the presentation of recommendations:

- **Top item.** Perhaps the simplest way to present a recommendation is by offering the user the best item for them. E.g. *“You have been watching a lot of sports, and football in particular. This is the most popular and recent item from the world cup.”*

- **Top N-items.** The system may also present several items at once. “*You have watched a lot of football and technology items. You might like to see the local football results and the gadget of the day.*” Note that while this system could be able to explain the relation between chosen items, it could also explain the rationale behind each single item.
- **Similar to top item(s).** Once a user shows a preference for one or more items, the recommender system can offer *similar* items. E.g. “*You might also like... Oliver Twist by Charles Dickens*”.
- **Predicted ratings for all items.** Rather than forcing selections on the user, a system may allow its users to browse all the available options. Recommendations are then presented as predicted ratings on a scale (say from 0 to 5) for each item. A user might query why a certain item, for example local hockey results, is predicted to have a low rating. The recommender system might then generate an explanation like: “*While this is a sports it is about hockey, which you do not seem to like!*”.
- **Structured overview.** The recommender system can give a structure which displays trade-offs between items [49, 68]. The advantage of a structured overview is that the user can see how items compare, and what other items are still available if the current recommendation should not meet their requirements.

15.6.2 Interacting with the Recommender System

There are different ways in which a user can give input to the recommender system. This interaction is what distinguishes conversational systems from “single-shot” recommendations. They allow users to elaborate their requirements over the course of an extended dialog [51] rather than each user interaction being treated independently of previous history.

We expand on the four ways suggested by [26], supplying examples of current applications³. Note that although there are more unobtrusive ways to elicit user preferences, e.g. via usage data [46] or demographics [6], this section focuses on *explicit* feedback from users.

- **The user specifies their requirements.** The user can specify their requirements through a dialog about their preferences in plain English [43, 64]. Such a dialog does not make use of the user’s previous interests, nor does it explain *directly*. That is, there is no sentence that claims to be a justification of the recommendation. It does however do so indirectly, by reiterating (and satisfying) the user’s *requirements*.
- **The user asks for an alternation.** A more direct approach is to allow users to explicitly critique recommended items (see also Chapter 13 on the evolution of critiquing), for instance using a structured overview (see Section 15.6.1).

³ A fifth section on mixed interaction interfaces is appended to the end of this original list.

One such system explains the difference between a selected item and remaining items [39].

- **The user rates items.** To change the type of recommendations they receive, the user may want to correct predicted ratings, or modify a rating they made in the past. The *influence based explanation* in Table 15.5 shows which rated titles influenced the recommended book the most [11].
- **The user gives their opinion.** A common usability principle is that it is easier for humans to recognize items, than to draw them from memory. For example, a user could specify whether they think an item is interesting or not, if they would like to see more similar items, or if they have already seen the item previously [12, 57].
- **Mixed interaction interfaces.** Recommender systems can also combine different types of interactions [17, 41].

Table 15.5: The *influence based explanation* showed which rated titles influenced the recommended book the most. Although this particular system did not allow the user to modify previous ratings, or degree of influence, in the explanation interface, it can be imagined that users could directly change their rating here. Note however, that it would be much harder to modify the degree of influence, as it is computed: any modification is likely to interfere with the regular functioning of the recommendation algorithm [11].

BOOK	YOUR RATING Out of 5	INFLUENCE Out of 100
Of Mice and Men	4	54
1984	4	50
Till We Have Faces: A Myth Retold	5	50
Crime and Punishment	4	46
The Gambler	5	11

15.7 Explanation Styles

Guideline 4: Consider the relationship between the underlying algorithm and the type of explanations you choose to generate. Do the explanations that you generate help you achieve your explanatory goals?

In this section we describe explanations inspired by a particular underlying algorithm, or different “explanation styles”. We caution that explanations may follow the “style” of a particular algorithm irrespective of whether or not this is how the

recommendations have been retrieved or computed. In other words, the explanation style for a given explanation *may, or may not*, reflect the underlying algorithm by which the recommendations are computed. There often is a divergence between how the recommendations are retrieved and the style of the given explanations. Consequently, this type of explanation would not be consistent with the goal of transparency, but may support other explanatory goals.

Table 15.6: Examples of explanations in commercial and academic systems, ordered by explanation style (case, collaborative, content, conversational, demographic and knowledge/utility-based).

System	Example explanation	Explanation style
<i>iSuggest-Usability</i> [30]	See e.g. Figure 15.8	Case-based
<i>LoveFilm.com</i>	<i>“Because you have selected or highly rated: Movie A”</i>	Case-based
<i>LibraryThing.com</i>	“Recommended By User X for Book A”	Case-based
<i>Netflix.com</i>	A list of similar movies the user has rated highly in the past	Case-based
<i>Amazon.com</i>	<i>“Customers Who Bought This Item Also Bought ...”</i>	Collaborative
<i>LIBRA</i> [11]	Keyword style (Tables 15.3 and 15.4); Neighbor style (Figure 15.3); Influence style (Figure 15.5)	Collaborative
<i>MovieLens</i> [29]	Histogram of neighbors (Figure 15.2) and Confidence display (Figure 15.5)	Collaborative
<i>Amazon.com</i>	<i>“Recommended because you said you owned Book A”</i>	Content-based
<i>CHIP</i> [20]	<i>“Why is ‘The Tailor’s Workshop recommended to you’? Because it has the following themes in common with artworks that you like: * Everyday Life * Clothes ...”</i>	Content-based
<i>Movielxplain</i> [58]	See Table 15.7	Content-based
<i>MovieLens: “Tags-explanations”</i> [62]	Tags ordered by relevance or preference (see Figure 15.7)	Content-based
<i>News Dude</i> [12]	<i>“This story received a [high/low] relevance score, because it contains the words f1, f2, and f3.”</i>	Content-based
Continued on next page		

Table 15.6 – continued from previous page

System	Example explanation	Explanation style
<i>OkCupid.com</i>	Graphs comparing two users according to dimensions such as “more introverted”; comparison of how users have answered different questions	Content-based
<i>Pandora.com</i>	<i>“Based on what you’ve told us so far, we’re playing this track because it features a leisurely tempo . . .”</i>	Content-based
<i>Adaptive place Advisor</i> [59]	Dialog e.g. “Where would you like to eat?” “Oh, maybe a cheap Indian place.”	Conversational
<i>ACORN</i> [65]	Dialog e.g. “What kind of movie do you feel like?” “I feel like watching a thriller.”	Conversational
INTRIGUE [6]	<i>“For children it is much eye-catching, it requires low background knowledge, it requires a few seriousness and the visit is quite short. For yourself it is much eye-catching and it has high historical value. For impaired it is much eye-catching and it has high historical value.”</i>	Demographic
<i>Qwikshop</i> [39]	<i>“Less Memory and Lower Resolution and Cheaper”</i>	Knowledge/utility-based
<i>SASY</i> [21]	<i>“... because your profile has: *You are single; *You have a high budget”</i> (Figure 15.1)	Knowledge/utility-based
<i>Top Case</i> [43]	“Case 574 differs from your query only in price and is the best case no matter what transport, duration, or accommodation you prefer”	Knowledge/utility-based
<i>(Internet Provider)</i> [23]	<i>“This solution has been selected for the following reasons: *Web space is available for this type of connection . . .”</i> (Figure 15.4)	Knowledge/utility-based
<i>“Organizational Structure”</i> [49]	Structured overview: <i>“We also recommend the following products because: *they are cheaper and lighter, but have lower processor speed.”</i> (Figure ??)	Knowledge/utility-based
Continued on next page		

Table 15.6 – continued from previous page

System	Example explanation	Explanation style
<i>myCameraAdvisor</i> [63]	e.g “...cameras capable of taking pictures from very far away will be more expensive ...”	Knowledge/utility-based

Transparency is not the only explanatory goal to consider when deciding upon explanation style. For example, for a given system one might find that users are more satisfied with content-based style explanations even though critique-based style explanations are more efficient. As of yet, there is little comparison between explanation styles with regard to their performance on explanatory goals. Only Hingston [30] has compared the understandability and scrutability of different explanation styles inspired by algorithm, although in these cases, the explanations were directly influenced by different underlying algorithms as well. Other studies have however considered the effects of different explanation interfaces on different explanatory goals [20, 29, 61].

Notwithstanding, the underlying algorithm of a recommender engine will to a certain degree influence the types of explanations that can be generated. Table 15.6 summarizes the most commonly used explanation styles (case-based, content-based, collaborative-based, demographic-based, knowledge and utility-based) with examples of each. In this section we describe each style: their *assumed* inputs, processes and generated explanations. For commercial systems where this information is not public, we offer educated guesses. While conversational systems are included in the Table, we consider conversational systems as more of an interaction style than a specific algorithm.

In the following sections we will give further examples of how explanation styles can be inspired by common algorithms as classified by Burke [13]. For each example we also mention how the recommendations are presented, and the interaction model that was chosen.

For describing the interface between the recommender system and explanation component we use the notation used in [13]: U is the set of users whose preferences are known, and $u \in U$ is the user for whom recommendations need to be generated. I is the set of items that can be recommended, and $i \in I$ is an item for which we would like to predict u 's preferences.

15.7.1 Collaborative-Based Style Explanations

For collaborative-based style explanations the assumed input to the recommender engine are user u 's ratings of items in I . These ratings are used to identify users that are similar in ratings to u . These similar users are often called “neighbors” as nearest-neighbors approaches are commonly used to compute similarity. Then, a

prediction for the recommended item is extrapolated from the neighbors' ratings of \mathbf{i} .

Commercially, the most well known usage of collaborative-style explanations are the ones used by Amazon.com: "*Customers Who Bought This Item Also Bought ...*". This explanation assumes that the user is viewing an item which they are already interested in. It implies that the system finds similar users (who bought this item), and retrieves and recommends items that these similar users bought. The recommendations are presented in the format of similar to top item. In addition, this explanation assumes an interaction model, whereby ratings are implicitly inferred through purchase behavior.

Herlocker et al. suggested 21 explanation interfaces using text as well as graphics [29]. These interfaces varied with regard to content and style, but a number of these explanations directly referred to the concept of neighbors. Figure 15.2 for example, shows how neighbors rated a given (recommended) movie, a bar chart with "good", "ok" and "bad" ratings clustered into distinct columns. Again, we see that this explanation is given for a specific way of recommending items, and a particular interaction model: this is a single recommendation (either top item or one item out of a top-N list), and assumes that the users are supplying rating information for items.

15.7.2 Content-Based Style Explanation

For content-based style explanations the assumed input to the recommender engine are user \mathbf{u} 's ratings (for a sub-set) of items in \mathbf{I} . These ratings are then used to generate a classifier that fits \mathbf{u} 's rating behavior and use it on \mathbf{i} . A prediction for the recommended item is based on how well it fits into this classifier. E.g. if it is similar to other highly rated items.

If we simplify this further, we could say that content-based algorithms consider similarity between items, based on user ratings but considering item properties. In the same spirit, content-based style explanations are based on the items' properties. For example, [58] justifies a movie recommendation according to what they infer is the user's favorite actor (see Table 15.7). While the underlying approach is in fact a hybrid of collaborative and content-based approaches, the explanation style suggests that they compute the similarity between movies according to the presence of features in highly rated movies. They elected to present users with several recommendations and explanations (top-N) which may be more suitable if the user would like to make a selection between movies depending on the information given in the explanations (e.g. feeling more like watching a movie with Harrison Ford over one starring Bruce Willis). The interaction model is based on ratings of items.

A more domain independent approach is suggested by [62] who suggest a similarity measure based on user specified keywords, or tags. The explanations used in this study use the relationship between keywords and items (tag relevance), and the relationship between tags and users (tag preference) to make recommendations

(see Figure 15.7). Tag preference, or how relevant a tag is for a given user, can be seen as a form of content-based explanation, as it is a weighted average of a given user’s ratings of movies with that tag. Tag relevance, or how relevant a keyword is for recommending an item, on the other hand is the correlation between (aggregate) users’ preference for the tag, and their preference for a movie with which the tag is associated. In this example, showing recommendations as a single top item allows the user to view many of the tags that are related to the item. The interaction model is again based on numerical ratings.

The commercial system Pandora, explains its recommendations of songs according to musical properties such as tempo and tonality. These features are inferred from users ratings of songs. Figure 15.6 shows an example of this [1]. Here, the user is offered one song at a time (top item) and gives their opinion as “thumbs-up” or “thumbs-down” which also can be considered as numerical ratings.

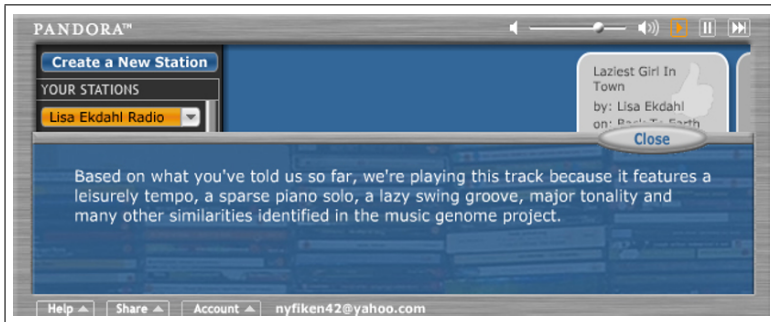


Fig. 15.6: Pandora explanation: “Based on what you’ve told us so far, we’re playing this track because it features a leisurely tempo . . .”

Table 15.7: Example of an explanation in Movieweave, using features such as actors, which occur for movies previously rated highly by this user, to justify a recommendation [58].

Recommended movie title	The reason is the participant	who appears in
Indiana Jones and the Last Crusade (1989)	Ford, Harrison	5 movies you have rated
Die Hard 2 (1990)	Willis, Bruce	2 movies you have rated

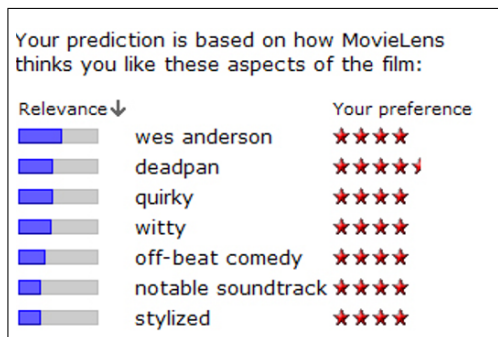


Fig. 15.7: Tagsplanation with both tag preference and relevance, but sorted by tag relevance

15.7.3 Case-Based Reasoning (CBR) Style Explanations

Explanations can also omit mention of significant properties and focus primarily on the similar items used to make the recommendation. The items used are thus considered cases for comparison, resulting in case-based style explanations. We note that CBR systems greatly vary with regard to the recommendation algorithm. For example, the FINDME recommender [14] is based on critiquing, and the ranking of items in [5] is based on their presence in travel plans of users who expressed similar interests. While these CBR systems have also used different methods to present their explanations, we recall that this section, and the sections describing the other explanation styles, are focused on the *style* of the explanation rather than the actual underlying algorithm. As such, each of these systems could in theory have had a case-based style explanation.

In fact, in this chapter we have already seen a type of case-based style explanation, the “influence based style explanation” of [11] in Figure 15.5. Here, the influence of an item on the recommendation is computed by looking at the difference in the score of the recommendation with and without that item. In this case, recommendations were presented as top item, assuming a rating based interaction. Another study computed the similarity between recommended items⁴, and used these similar items as justification for a top item recommendation in the “learn by example” explanations (see Figure 15.8) [30].

⁴ The author does not specify which similarity metric was used, though it is likely to be a form of rating based similarity measure such as cosine similarity.

15.7.4 Knowledge and Utility-Based Style Explanations

For knowledge and utility-based style explanations the assumed input to the recommender engine are description of user **u**'s needs or interests. The recommender engine then infers a match between the item **i** and **u**'s needs. One knowledge-based recommender system takes into consideration how camera properties such as memory, resolution and price reflect the available options as well as a user's preferences [39]. Their system may explain a camera recommendation in the following manner: "*Less Memory and Lower Resolution and Cheaper*". Here recommendations are presented as a form of structured overview describing the competing options, and the interaction model assumes that users ask for alterations in the recommended items.

Similarly, in the system described in [43] users gradually specify (and modify) their preferences until a top recommendation is reached. This system can generate explanations such as the following for a recommended holiday titled "Case 574": "*Top Case: Case 574 differs from your query only in price and is the best case no matter what transport, duration, or accommodation you prefer*".

It is arguable that there is a certain degree of overlap between knowledge-based, content-based style (Section 15.7.2) and case-based style explanations (Section 15.7.3) which can be derived from either type of algorithm depending on the details of the implementation.

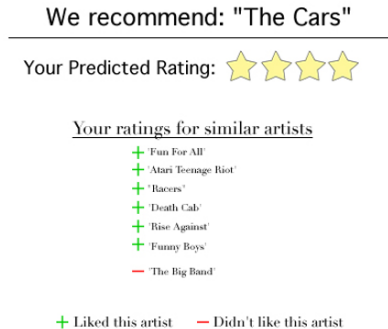


Fig. 15.8: Learn by example, or case based reasoning [30].

15.7.5 Demographic Style Explanations

For demographic-based style explanations, the assumed input to the recommender engine is demographic information about user u . From this, the recommendation algorithm identifies users that are demographically similar to u . A prediction for the recommended item i is extrapolated from how the similar users rated this item, and how similar they are to u .

Surveying a number of systems which use a demographic-based filter e.g. [6, 34, 48], we could only find one which offers an explanation facility: “*For children it is much eye-catching, it requires low background knowledge, it requires a few seriousness and the visit is quite short. For yourself it is much eye-catching and it has high historical value. For impaired it is much eye-catching and it has high historical value.*”[6]. In this system recommendations were offered as a structured overview, categorizing places to visit according to their suitability to different types of travelers (e.g. children, impaired). Users can then add these items to their itinerary, but there is no interaction model that modifies subsequent recommendations

To our knowledge, there are no other systems that make use of demographic style explanations. It is possible that this is due to the sensitivity of demographic information; anecdotally we can imagine that many users would not want to be recommended an item based on their gender, age or ethnicity (e.g. “*We recommend you the movie Sex in the City because you are a female aged 20-40.*”).

15.8 Summary and future directions

In this chapter, we offer guidelines for the designers of explanations in recommender systems. Firstly, the designer should consider what benefit the explanations offer, and thus which criteria they are evaluating the explanations for (e.g. *transparency, scrutability, trust, efficiency, effectiveness, persuasion or satisfaction*). The developer may select several criteria which may be related to each other, but may also be conflicting. In the latter case, it is particularly important to distinguish between these evaluation criteria. It is only in more recent work that these trade-offs are being shown and becoming more apparent [20, 61].

In addition, the system designer should consider the *metrics* they are going to use when evaluating the explanations, and the dependencies the explanations may have with different parts of the system, such as the way recommendations are presented (e.g. top item, top N-items, similar to top item(s), predicted ratings for all items, structured overview), the way users interact with the explanations (e.g. the user specifies their requirements, asks for an alteration, rates items, gives their opinion, or uses a hybrid interaction interface) and the underlying recommender engine.

To offer a single example of the relation between explanations and other recommender system factors, we can imagine a recommender engine with low recommendation accuracy. This may affect all measurements of effectiveness in the system, as users do not really like the items they end up being recommend. These measure-

ments do not however reflect the effectiveness of the *explanations* themselves. In this case, a layered approach to evaluation [47], where explanations are considered in isolation from the recommendation algorithm as seen in [61], may be warranted. Similarly, thought should be given to how the method of presenting recommendations, and the method of interaction may affect the (evaluation of) explanations.

We offered examples of explanation styles influenced by the most common algorithms (e.g. content-based, collaborative, demographic, or knowledge/utility-based), and how they have been used in existing systems. To a certain extent these types of explanations can be reused (likely at the cost of transparency) for hybrid recommendations, and other complex recommendation methods such as latent semantic analysis, but these areas of research remain largely open. Preliminary works for some of these areas can be found in e.g. [33] (explaining Markov decision processes) and [31] (explaining latent semantic analysis models).



Fig. 15.9: Newsmap - a treemap visualization of news. Different colors represent topic areas, square and font size to represent importance to the current user, and shades of each topic color to represent recency.

As of yet, there has been little comparison between explanation styles with regard to their performance on explanatory goals. This is an avenue of research in which we hope to see further progress in the near future. Also, future work will likely involve more advanced interfaces for explanations. For example, the “treemap” structure (see Figure 15.9 [4]) offers an overview of the search space [9]. This type of overview may also be used for explanation. Assume for example, that a user is being recommended the piece “The Votes Obama Truly Needs”, and that this rectangle is highlighted. This interface “explains” that this item is being recommended because the user is interested in current US news (orange color), it is popular (big square), and that it is recent (bright color).

Last, but certainly not least, researchers are starting to find that explanations are part of a cyclical process. The explanations affect a user’s mental model of the rec-

ommender system, and in turn the way they interact with the explanations. In fact this may also impact the recommendation accuracy negatively [7, 20]. For example [7] saw that recommendation accuracy decreased as users removed keywords from their profile for a news recommender system. Understanding this cycle will likely be one of the future strands of research.

References

1. Pandora (2006). <http://www.pandora.com>
2. Movielens dataset (2009). <http://www.grouplens.org/node/73>
3. Netflix dataset (2009). <http://www.netflixprize.com/>
4. Newsmap (2009). <http://www.marumushi.com/apps/newsmap/index.cfm>
5. Nutking (2010). <http://nutking.ectrldev.com/nutking/jsp/language.do?action=english>
6. Adrissono, L., Goy, A., Petrone, G., Segnan, M., Torasso, P.: Intrigue: Personalized recommendation of tourist attractions for desktop and handheld devices. *Applied Artificial Intelligence* **17**, 687–714 (2003)
7. Ahn, J.W., Brusilovsky, P., Grady, J., He, D., Syn, S.Y.: Open user profiles for adaptive news systems: help or harm? In: *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pp. 11–20. ACM Press, New York, NY, USA (2007).
8. Andersen, S.K., Olesen, K.G., Jensen, F.V.: HUGIN—a shell for building Bayesian belief universes for expert systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1990)
9. Bederson, B., Shneiderman, B., Wattenberg, M.: Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Transactions on Graphics* **21**(4), 833–854. (2002)
10. Bennett, S.W., Scott, A.C.: *The Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, chap. 19 - Specialized Explanations for Dosage Selection, pp. 363–370. Addison-Wesley Publishing Company (1985)
11. Bilgic, M., Mooney, R.J.: Explaining recommendations: Satisfaction vs. promotion. In: *Proceedings of the Workshop Beyond Personalization, in conjunction with the International Conference on Intelligent User Interfaces*, pp. 13–18 (2005)
12. Billsus, D., Pazzani, M.J.: A personal news agent that talks, learns, and explains. In: *Proceedings of the Third International Conference on Autonomous Agents*, pp. 268–275 (1999)
13. Burke, R.: Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction* **12**(4), 331–370 (2002)
14. Burke, R.D., Hammond, K.J., Young, B.C.: Knowledge-based navigation of complex information spaces. In: *AAAI/IAAI*, Vol. 1, pp. 462–468 (1996)
15. Carenini, G., Mittal, V., Moore, J.: Generating patient-specific interactive natural language explanations. *Proc Annu Symp Comput Appl Med Care* pp. 5–9 (1994)
16. Chen, L., Pu, P.: Trust building in recommender agents. In: *WPRSIUI in conjunction with Intelligent User Interfaces*, pp. 93–100 (2002)
17. Chen, L., Pu, P.: Hybrid critiquing-based recommender systems. In: *Intelligent User Interfaces*, pp. 22–31 (2007)
18. Cosley, D., Lam, S.K., Albert, I., Konstan, J.A., Riedl, J.: Is seeing believing?: how recommender system interfaces affect users' opinions. In: *CHI, Recommender systems and social computing*, vol. 1, pp. 585–592 (2003).
19. Cramer, H., Evers, V., Someren, M.V., Ramlal, S., Rutledge, L., Stash, N., Aroyo, L., Wielinga, B.: The effects of transparency on perceived and actual competence of a content-based recommender. In: *Semantic Web User Interaction Workshop, CHI (2008)*

20. Cramer, H.S.M., Evers, V., Ramlal, S., van Someren, M., Rutledge, L., Stash, N., Aroyo, L., Wielinga, B.J.: The effects of transparency on trust in and acceptance of a content-based art recommender. *User Model. User-Adapt. Interact* **18**(5), 455–496 (2008).
21. Czarkowski, M.: A scrutable adaptive hypertext. Ph.D. thesis, University of Sydney (2006)
22. Doyle, D., Tsybal, A., Cunningham, P.: A review of explanation and explanation in case-based reasoning. Tech. rep., Department of Computer Science, Trinity College, Dublin (2003)
23. Felfernig, A., Gula, B.: Consumer behavior in the interaction with knowledge-based recommender applications. In: *ECAI 2006 Workshop on Recommender Systems*, pp. 37–41 (2006)
24. Fogg, B., Marshall, J., Kameda, T., Solomon, J., Rangekar, A., Boyd, J., Brown, B.: Web credibility research: A method for online experiments and early study results. In: *CHI 2001*, pp. 295–296 (2001)
25. Fogg, B.J., Soohoo, C., Danielson, D.R., Marable, L., Stanford, J., Tauber, E.R.: How do users evaluate the credibility of web sites?: a study with over 2,500 participants. In: *Proceedings of DUX'03: Designing for User Experiences*, no. 15 in *Focusing on user-to-product relationships*, pp. 1–15 (2003). URL <http://doi.acm.org/10.1145/997078.997097>
26. Ginty, L.M., Smyth, B.: Comparison-based recommendation. *Lecture Notes in Computer Science* **2416**, 731–737 (2002).
27. Hance, E., Buchanan, B.: Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project. Addison-Wesley (1984)
28. Häubl, G., Trifts, V.: Consumer decision making in online shopping environments: The effects of interactive decision aids. *Marketing Science* **19**, 4–21 (2000)
29. Herlocker, J.L., Konstan, J.A., Riedl, J.: Explaining collaborative filtering recommendations. In: *ACM conference on Computer supported cooperative work*, pp. 241–250 (2000)
30. Hingston, M.: User friendly recommender systems. Master's thesis, Sydney University (2006)
31. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: *ICDM* (2008)
32. Hunt, J.E., Price, C.J.: Explaining qualitative diagnosis. *Engineering Applications of Artificial Intelligence* **1**(3), Pages 161–169 (1988)
33. Khan, O.Z., Poupart, P., Black, J.P.: Minimal sufficient explanations for mdps. In: *Workshop on Explanation-Aware Computing associated with IJCAI* (2009)
34. Krulwich, B.: The infofinder agent: Learning user interests through heuristic phrase extraction. *IEEE Intelligent Systems* **12**, 22–27 (1997)
35. Lacave, C., Diéz, F.J.: A review of explanation methods for bayesian networks. *The Knowledge Engineering Review* **17**:2, 107–127 (2002)
36. Lacave, C., Diéz, F.J.: A review of explanation methods for heuristic expert systems. *The Knowledge Engineering Review* **17**:2, 107–127 (2004)
37. Lewis, C., Rieman, J.: Task-centered user interface design: a practical introduction. University of Colorado (1994)
38. Lopez-Suarez, A., Kamel, M.: Dykor: a method for generating the content of explanations in knowledge systems. *Knowledge-based Systems* **7**(3), 177–188 (1994)
39. McCarthy, K., Reilly, J., McGinty, L., Smyth, B.: Thinking positively - explanatory feedback for conversational recommender systems. In: *Proceedings of the European Conference on Case-Based Reasoning (ECCBR-04) Explanation Workshop.*, pp. 115–124 (2004)
40. McNee, S., Lam S.K. and Guetzlaff, C., Konstan J.A. and Riedl, J.: Confidence displays and training in recommender systems. In: *INTERACT IFIP TC13 International Conference on Human-Computer Interaction*, pp. 176–183 (2003)
41. McNee, S.M., Lam, S.K., Konstan, J.A., Riedl, J.: Interfaces for eliciting new user preferences in recommender systems. *User Modeling* pp. pp. 178–187 (2003)
42. McNee, S.M., Riedl, J., Konstan, J.A.: Being accurate is not enough: How accuracy metrics have hurt recommender systems. In: *Extended Abstracts of the 2006 ACM Conference on Human Factors in Computing Systems (CHI 2006)* (2006)
43. McSherry, D.: Explanation in recommender systems. *Artificial Intelligence Review* **24**(2), 179 – 197 (2005)

44. Nielsen, J., Molich, R.: Heuristic evaluation of user interfaces. In: ACM CHI'90, pp. 249–256 (1990)
45. Ohanian, R.: Construction and validation of a scale to measure celebrity endorsers' perceived expertise, trustworthiness, and attractiveness. *Journal of Advertising* **19**:3, 39–52 (1990)
46. O'Sullivan, D., Smyth, B., Wilson, D.C., McDonald, K., Smeaton, A.: Improving the quality of the personalized electronic program guide. *User Modeling and User-Adapted Interaction* **14**, pp. 5–36 (2004)
47. Paramythis, A., Totter, A., Stephanidis, C.: A modular approach to the evaluation of adaptive user interfaces. In: S. Weibelzahl, D.N. Chin, G. Weber (eds.) *Evaluation of Adaptive Systems in conjunction with UM'01*, pp. 9–24 (2001)
48. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review* **13**, 393–408 (1999)
49. Pu, P., Chen, L.: Trust building with explanation interfaces. In: IUI'06, Recommendations I, pp. 93–100 (2006).
50. Pu, P., Chen, L.: Trust-inspiring explanation interfaces for recommender systems. *Knowledge-based Systems* **20**, 542–556 (2007)
51. Rafter, R., Smyth, B.: Conversational collaborative recommendation - an experimental analysis. *Artif. Intell. Rev* **24**(3-4), 301–318 (2005). URL <http://dx.doi.org/10.1007/s10462-005-9004-8>
52. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Dynamic critiquing. In: P. Funk, P.A. González-Calero (eds.) *ECCBR, Lecture Notes in Computer Science*, vol. 3155, pp. 763–777. Springer (2004)
53. Roth-Berghofer, T., Schulz, S., Leake, D.B., Bahls, D.: Workshop on explanation-aware computing. In: *ECAI* (2008)
54. Roth-Berghofer, T., Tintarev, N., Leake, D.B.: Workshop on explanation-aware computing. In: *IJCAI* (2009)
55. Sinha, R., Swearingen, K.: The role of transparency in recommender systems. In: *Conference on Human Factors in Computing Systems*, pp. 830–831 (2002)
56. Sørmo, F., Cassens, J., Aamodt, A.: Explanation in case-based reasoning perspectives and goals. *Artificial Intelligence Review* **24**(2), 109 – 143 (2005)
57. Swearingen, K., Sinha, R.: Interaction design for recommender systems. In: *Designing Interactive Systems*, pp. 25–28 (2002).
58. Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: Justified recommendations based on content and rating data. In: *WebKDD Workshop on Web Mining and Web Usage Analysis* (2008)
59. Thompson, C.A., Göker, M.H., Langley, P.: A personalized system for conversational recommendations. *J. Artif. Intell. Res. (JAIR)* **21**, 393–428 (2004).
60. Tintarev, N., Masthoff, J.: Over- and underestimation in different product domains. In: *Workshop on Recommender Systems associated with ECAI* (2008)
61. Tintarev, N., Masthoff, J.: Personalizing movie explanations using commercial meta-data. In: *Adaptive Hypermedia* (2008)
62. Vig, J., Sen, S., Riedl, J.: Tagsplanations: Explaining recommendations using tags. In: *Intelligent User Interfaces* (2009)
63. Wang, W., Benbasat, I.: Recommendation agents for electronic commerce: Effects of explanation facilities on trusting beliefs. *Journal of Management Information Systems* **23**, 217–246 (2007)
64. Wärnestål, P.: Modeling a dialogue strategy for personalized movie recommendations. In: *Beyond Personalization Workshop*, pp. 77–82 (2005)
65. Wärnestål, P.: User evaluation of a conversational recommender system. In: *Proceedings of the 4th Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pp. 32–39 (2005)
66. Wick, M.R., Thompson, W.B.: Reconstructive expert system explanation. *Artif. Intell.* **54**(1-2), 33–70 (1992).
67. Ye, L., Johnson, P., Ye, L.R., Johnson, P.E.: The impact of explanation facilities on user acceptance of expert systems advice. *MIS Quarterly* **19**(2), 157–172 (1995).

68. Yee, K.P., Swearingen, K., Li, K., Hearst, M.: Faceted metadata for image search and browsing. In: ACM Conference on Computer-Human Interaction (2003)
69. Zaslow, J.: Oh no! My TiVo thinks I'm gay (2002).

Chapter 16

Usability Guidelines for Product Recommenders Based on Example Critiquing Research

Pearl Pu, Boi Faltings, Li Chen, Jiyong Zhang and Paolo Viappiani

Abstract Over the past decade, our group has developed a suite of decision tools based on example critiquing to help users find their preferred products in e-commerce environments. In this chapter, we survey important usability research work relative to example critiquing and summarize the major results by deriving a set of usability guidelines. Our survey is focused on three key interaction activities between the user and the system: the initial preference elicitation process, the preference revision process, and the presentation of the systems recommendation results. To provide a basis for the derivation of the guidelines, we developed a multi-objective framework of three interacting criteria: accuracy, confidence, and effort (ACE). We use this framework to analyze our past work and provide a specific context for each guideline: when the system should maximize its ability to increase users' decision *accuracy*, when to increase user *confidence*, and when to minimize the interaction *effort* for the users. Due to the general nature of this multi-criteria model, the set of guidelines that we propose can be used to ease the usability engineering process of other recommender systems, especially those used in e-commerce environments. The ACE framework presented here is also the first in the field to evaluate the performance of preference-based recommenders from a user-centric point of view.

Designers can use these guidelines for the implementation of an effective and successful product recommender.

Pearl Pu · Li Chen · Jiyong Zhang
Human Computer Interaction Group, School of Computer and Communication Sciences,
Swiss Federal Institute of Technology in Lausanne (EPFL), CH-1015, Lausanne, Switzerland
e-mail: \{pearl.pu, li.chen, jiyong.zhang\}@epfl.ch

Boi Faltings
Artificial Intelligence Laboratory, School of Computer and Communication Sciences
Swiss Federal Institute of Technology in Lausanne (EPFL), CH-1015, Lausanne, Switzerland
e-mail: boi.faltings@epfl.ch

Paolo Viappiani
Department of Computer Science, University of Toronto, 6 King's College Road, M5S3G4,
Toronto, ON, CANADA
e-mail: paolo.viappiani@gmail.com

16.1 Introduction

According to Jacob Nielsen, a well known usability researcher, the first law of e-commerce is that if users cannot find the product, they cannot buy it either.¹ The word “find” indeed defines a challenging task that e-commerce systems must support. It refers to the online retailer’s ability to help users identify an ideal product that satisfies the user’s needs (sometimes even unknown to him/herself) and inspire users to select the items recommended to them. This implies that the system must assist users to carry out not only a search but also a decision making task.

How do users actually face such tasks in the online environments? With increased competition, online retailers are offering a progressively large collection of available products. New items are added to their catalogs regularly, often to ensure that all of their direct competitors’ products are included in their inventory as well. The task of locating a desired choice is directly dependent on the number of available options. Indeed with this “infinite” shelf space the user task is becoming daunting, if not impossible, for the average user. Under such circumstances, users are likely to employ one of two decision approaches. In the first case, they try to achieve high decision accuracy, but face the time-consuming task of sifting through all options and trading off the pros and cons between the various aspects of the products. Alternatively, they can adopt heuristic decision strategies and process information more selectively. Although they expend less effort in this case, these heuristic strategies can lead to decision errors and are likely to cause decision regret. Clearly, neither approach is ideal, since adopting one or the other implies a compromise on either decision accuracy or effort. According to [40], the tradeoff between accuracy and effort is an inherent dilemma in decision-making that cannot be easily reconciled.

Significant research has been performed to develop highly interactive and intelligent tools to assist users. As a result, preference-based recommenders have emerged and are broadly recognized as effective search and navigation mechanisms guiding users to find their preferred products in e-commerce and other demanding decision environments. In the past decade, we have developed the example critiquing method and a suite of decision tools based on this method to provide personalization and recommendation to the product search problem [43, 45, 51, 52]. More than a dozen user studies were carried out and published, which validated the method in various data domains: travel planning, apartment search, and product search for laptops, Tablet PCs, and digital cameras. However, the wide adoption of example critiquing in electronic commerce remains limited.

Our goal is to analyze and survey our past work related to example critiquing and synthesize the major results by deriving a set of usability guidelines. More specifically, we focus on three key interaction activities: the initial preference elicitation process, the preference revision process, and the presentation of the system’s recommendation results. To provide a basis for the derivation of the guidelines, we investigate the objectives a product recommender must achieve in order to maximize

¹ Nielsen stated this law in his Alertbox in 2003. For details, please visit <http://www.useit.com/alertbox/20030825.html>.

user satisfaction and their willingness to use the system. A recommender's accuracy, i.e., how the system finds items that users truly want, has traditionally been an important and central aspect of recommenders. However, accuracy alone does not entirely capture user benefit without the consideration of ease of use (usability) [35]. People are known to have very limited cognitive resources and are not likely to achieve a high level of accuracy if the required effort is excessive. Finally, since product search is a decision process, the recommender must also help users achieve confidence that the products recommended to them are what they truly want. Instead of accuracy alone, we therefore propose a multi-objective framework, called ACE, for the derivation of our guidelines: 1) the system's ability to help users find their most preferred item (accuracy), 2) its ability to inspire users' confidence in selecting the items that were recommended to them (confidence), and 3) the amount of user effort it requires for achieving the relative accuracy (effort).

Notice that some users may be ready to spend a lot of effort to obtain a very accurate recommendation, while others may be ready to accept a lower accuracy to obtain a result quickly. While the design of a recommender clearly involves a tradeoff between accuracy and effort, what actually matters to the user is the tradeoff between *confidence* and effort: I am willing to put in more interaction effort only if I am increasingly convinced of the products that are recommended to me. The main challenge for recommender system design is to ensure that user confidence is sufficiently high to make them spend enough effort to reach an acceptable decision outcome.

This set of three requirements for deriving the guidelines can also be used as an evaluation framework to measure the usability of a product recommender. This chapter therefore contributes to the field in two main areas. From an academic point of view, the article establishes a novel set of user-centric criteria to evaluate the performance of preference-based recommenders. From a practical point of view, the chapter surveys the state of the art of example critiquing on three key interaction activities and derives a set of 11 usability guidelines that can be applied in a wider and more scalable way. Since these guidelines are derived from methods that have been validated in usability studies, practitioners can use them with confidence to enhance the usability engineering process, including design and testing, of a product recommender.

16.2 Preliminaries

16.2.1 Interaction Model

Preference-based recommenders suggest items to users based on their *explicitly* stated preferences over the attributes of the items. The various systems described in this chapter were designed using different architectures and evaluated using different data domains. However, they share some overall characteristics, especially

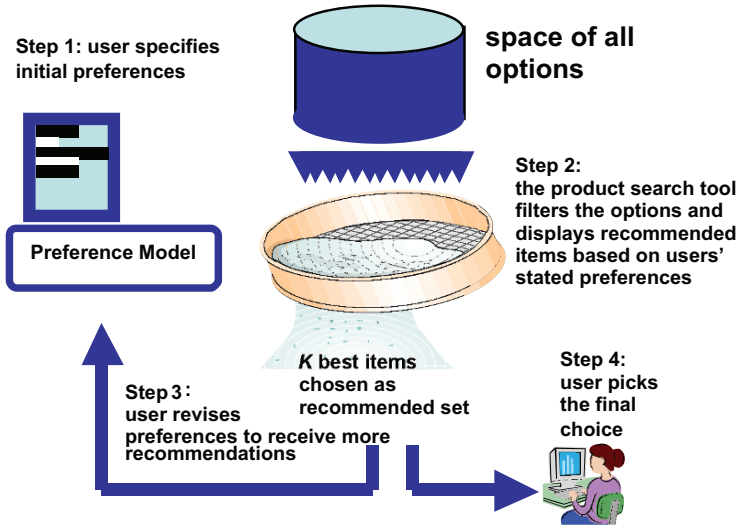


Fig. 16.1: The generic system-user interaction model of a preference-based recommender system.

concerning the interaction model. To form a nomenclature throughout the survey and the guideline derivation process, we present a generic model of interaction summarizing the steps of a recommender in Figure 16.1. A user starts his/her interaction process by stating a set of initial preferences via, for example, a graphical user interface. After obtaining that information, the system filters the space of options and selects the items to be recommended to users based on their stated preferences. This set is called the *recommendation set*. At that point, either the user finds her most preferred item in the recommendation set and thus terminates her interaction with the system, or she revises the preference model, using critiques such as “I would like a cheaper item”, in order to obtain more accurate recommendations. This last user feedback step is called preference revision. As a result of the process, the user can either pick a single item, or construct a list of items known as the consideration set that might then be compared in further detail.

The recommender can fail if the user is not sufficiently confident that the process is indeed finding a suitable product, and therefore does not undertake sufficient preference revision cycles to give the system an accurate preference model. Our guidelines are designed to avoid this situation, and at the end of this chapter we provide a model that shows their rationale with respect to the user’s decision process.

Once a recommendation set has been determined, a system may use various display strategies to show the results. A typical tool presents the user with a set of k items ($1 \leq k \leq n$, where n is the total number of products) in each of a total of m interactions. In each display of these k items, a user is identifying her target choice to be included in the consideration set. The more options are displayed, the more effort the user must expend to examine them. On the other hand, in a small display

set users could easily overlook their target choice and engage in more interaction cycles. This tradeoff of effort versus accuracy will be discussed in further detail when individual systems are presented.

The presented interactive components may not be simultaneously included in the same system by other tools. For example, the initial preference elicitation is an optional step when users are presented with a set of recommendations (e.g., best sellers in different categories) as soon as they visit a site. Other systems, on the other hand, may elicit users' initial preferences but do not provide the option to allow users to revise them.

16.2.2 Utility-Based Recommenders

Since example critiquing tools are based on multi-attribute utility theory, we provide an overview of the underlying recommendation algorithm.

The fundamental assumption underlying these recommenders is that people prefer items because of their attributes. Different values of an attribute correspond to different degrees of preference depending on the situation: for example, a large apartment is useful when there are guests, but not useful when it has to be cleaned. A user will determine preferences for an item by weighing the advantages and disadvantages of each feature according to how often it is beneficial and how often it is not. Thus, preference is a weighted function of attributes.

Formally, the preference-based recommendation problem can be formulated as a Multi-Attribute Decision Problem (MADP) $\Psi = \langle \mathbf{X}, \mathbf{D}, \mathbf{O}, \mathbf{P} \rangle$, where $\mathbf{X} = \{X_1, \dots, X_n\}$ is a finite set of attributes that the product catalog has, $\mathbf{D} = D_1 \times \dots \times D_n$ indicates the product domain space (each $D_i (1 \leq i \leq n)$ is a set of domain values for attribute X_i), $\mathbf{O} = \{O_1, \dots, O_m\}$ is a finite set of available products that the system may provide, and $\mathbf{P} = \{P_1, \dots, P_t\}$ denotes a set of preferences that the user may have. The objective of a MADP is to find a product (or products) that is (or are) most preferred by the user. A MADP can be solved by constraint-based approaches or utility-based approaches. Below we introduce the approach based on the multi-attribute utility theory (MAUT) to solve a given MADP. Please see [45] for the constraint-based approach.

Multi-Attribute Utility Theory (MAUT)

The origin of utility theory dates back to 1738 when Bernoulli proposed his explanation to the St. Petersburg paradox in terms of the utility of monetary value [4]. Two centuries later it was von Neumann and Morgenstern (1944) who revived this method to solve problems they encountered in economics [71]. Later, in the early 1950s, in the hands of Marschak [29] and of Herstein and Milnor [21], the Expected Utility Theory was established on the basis of a set of axioms that form the basis of the von Neumann Morgenstern theorem (VNM Theorem) [39, 59].

In the 1970s Keeney and Raiffa [24] extended utility theory to the case of multiple attributes. The main idea of multi-attribute utility theory is to represent user preferences as utility functions of the attribute values.

Let the symbol \succeq denote the user’s preference order, e.g. $A \succeq B$ means “A is preferred or indifferent to B”. According to Utility Theory, for a given MADP, there exists a utility function $U : O \rightarrow \mathfrak{R}$, such that for any two possible products O and $\bar{O} \in O$,

$$O \succeq \bar{O} \iff U(O) \geq U(\bar{O}) \tag{16.1}$$

More specifically, a product O can be represented by a set of attribute values $\langle X_1 = x_1, \dots, X_n = x_n \rangle$ (in short as $\langle x_1, \dots, x_n \rangle$), thus the above formula can be rewritten as

$$\langle x_1, \dots, x_n \rangle \succeq \langle \bar{x}_1, \dots, \bar{x}_n \rangle \iff U(\langle x_1, \dots, x_n \rangle) \geq U(\langle \bar{x}_1, \dots, \bar{x}_n \rangle) \tag{16.2}$$

If the utility function is given, the degree of preference for each product is characterized as a numerical utility and the preference order of all products is given by these utility values.

Finding the proper utility function U to represent users’ preferences precisely is a challenging task. While in theory the utility function can be used in any style to represent user preferences, a special case is commonly used to reduce computation effort. If the attributes are mutually preferentially independent² based on the utility theory, the utility function has the additive form as follows:

$$U(\langle x_1, \dots, x_n \rangle) = \sum_{i=1}^n w_i v_i(x_i) \tag{16.3}$$

where v_i is a value function of attribute X_i with range $[0, 1]$, and w_i is the weight value of X_i satisfying $\sum_{i=1}^n w_i = 1$. In other words, the utility function for a product O is the weighted sum of the utility functions for each of its attributes. The weight value for each attribute can be given as default value $1/n$, and we can allow the user to specify the weight values of some attributes. The value function v_i can be determined to satisfy the user’s preferences related to the attribute X_i . Usually a linear function with the form $v_i = ax_i + b$ is enough to represent the user’s preference on each attribute.

Once the utility function for each product is determined, we are able to rank all the products based on their overall utilities and select the top K products with highest utilities as the recommendation set. In practice, we assume that the attributes of any product are mutually preferentially independent, so the additive form of the utility function can always be applied.

² An attribute X is said to be preferentially independent of another attribute Y if preferences for levels of attribute X do not depend on the level of attribute Y . If Y is also preferentially independent of X , then the two attributes are said to be mutually preferentially independent. See more details in [22].

16.2.3 *The Accuracy, Confidence, Effort Framework*

As mentioned in the introduction, our search for a multi-objective requirement framework is to find a basis for the derivation of the design guidelines. We look for criteria that a product recommender must satisfy in order to achieve maximum user satisfaction and their willingness to use the system. We give a more precise definition of the ACE (Accuracy, Confidence, Effort) framework as well as ways of measuring these variables.

Accuracy refers to the objective accuracy of a recommender. For rating-based systems, the most often used measure is the mean absolute error (or MAE) [1]. It is measured by an offline procedure known as leave-one-out on a previously acquired dataset. Leave-one-out involves leaving one rating out and then trying to predict it with the recommender algorithm being evaluated. The predicted rating is then compared with the real rating and the difference in absolute value is computed. The procedure is repeated for all the ratings and an average of all the errors is called the Mean Absolute Error.

Recommendations generated by utility-based systems are based on users' preference profiles. Since such profiles cannot be simulated, offline methods to measure accuracy are not possible. One method commonly used in this field is the switching task as defined in [48]. It measures how often users' truly preferred items are selected from the ones recommended to them. For example, if 70 out of 100 users found their preferred items during the recommendation process without changing their decisions after the experiment administrator presented all of the available options to them, then we say the accuracy of the system is 70%. This implies an experimental procedure where each user first interacts with a recommender to pick items for her consideration set. In a second phase of this procedure, the experiment administrator will show all of the available items and then ask her whether she finds the items in the consideration set still attractive. If she switches to other items, the system has failed to help her make an accurate decision. Such procedures were already employed in consumer decision research to measure decision quality, known as the switching task [20]. The fraction of users that switch to other items, called the switching rate, gives the only precise account of decision accuracy for a personalized recommender tool. However, as they are very time consuming, switching tasks are only performed in carefully defined empirical studies.

User confidence, the second evaluation criterion in our ACE framework, is the system's ability to inspire users to select the items recommended to them. It is a subjective variable and can only be assessed using a post-study questionnaire that asks users to indicate their agreement with statements such as: *I am confident that the items recommended to me are the ones I look for*. When the objective accuracy cannot be feasibly obtained, user confidence can be used to assess the perceived accuracy of a system. Perceived accuracy is strongly correlated to actual accuracy, but also influenced by other aspects of the recommender: whether the user is given enough possibilities and sufficiently involved in preference elicitation and revision, and whether the display of results convinces the user that the best results are being found.

By *user effort*, we refer to the actual task time users take to finish an instructed action, such as the preference elicitation and preference revision procedures and the time they take to construct the consideration set. As an alternative to elapsed time, we can also measure the number of interaction cycles for any of the interaction activities since this measure is independent of the working habits of individual users.

Throughout this chapter, we will analyze the 3 major components of example-based recommender systems: preference elicitation, revision and result display. We use accuracy, confidence and effort to evaluate different techniques and derive a set of 11 guidelines for the design of successful example-based recommender systems, and compare them against a model of user behavior that provides a unified motivation.

We believe that confidence is a major factor for the success of any recommender system, and therefore suggest that the ACE framework could be useful to evaluate other types of recommender systems as well.

16.2.4 Organization of this Chapter

We structure the guidelines according to the generic components of the model presented in Figure 16.1: initial preference elicitation (step 1), preference revision (step 4), and display strategy (step 2). The rest of this article is organized as follows: section 16.3 (Related Work) reviews design guidelines for preference elicitation and personalized product recommender tools in other fields; Section 16.4 (Initial Preference Elicitation) presents guidelines for motivating users to state their initial preferences as accurately as possible using principles from behavioral decision research; Section 16.5 (Stimulating Preference Elicitation with Examples) continues the discussion on preference elicitation and identifies concrete methods to help users state complete and sound preferences; Section 16.6 (Preference Revision) describes strategies to help users resolve conflicting preferences and perform tradeoff decisions; Section 16.7 (Display Strategies) presents guidelines for device display strategies that achieve a good balance between minimizing user's information processing effort and maximizing decision accuracy and user confidence; Section 16.8 presents a model of user behavior that provides a rationale for the guidelines; Section 16.9 concludes this article.

16.3 Related Work

16.3.1 Types of Recommenders

Two types of recommender systems have been broadly recognized in the field relative to the way systems gather and build user preference profiles: those based on

users' explicitly stated preferences (called preference-based recommenders) and those based on users navigation or purchase behaviors (called behavior-based recommenders). For the treatment of behavior-based recommenders, which generate recommendations based on users' accumulated interaction behaviors such as the items users have examined and purchased, please refer to [75], and to [26, 55] for demographic-based recommenders. Four types of preference-based recommenders exist: rating-based, case-based, utility-based and critiquing-based. Please see other ways to classify recommender systems ([1, 7]).

16.3.2 Rating-based Systems

Users explicitly express their preferences (even though they may not know it) by giving either binary or multi-scale scores to items that they experienced. Either the system proposes a user to rate a set of items or users will select on their own a set of items to rate. These initial ratings constitute the user profile. Systems that fall into this category are most commonly known as collaborative recommenders due to the fact that the user is recommended items that people with similar tastes and preferences liked in the past. For this reason, this type of system is also called social recommender. The details of how the collaborative algorithms work can be found in [1]. Lately some websites, such as tripadvisor.com, started collecting users' ratings on multiple attributes of an item to obtain a more refined preference profile.

16.3.3 Case-based Systems

This type of system recommends items that are similar to what users have indicated as interesting. A product is treated as a case having multiple attributes. Content-based [1] and case-based technologies [7] are used to analyze the attribute values of available products and the stated preferences of a user, and then identify one or several best-ranked options according to a ranking scheme.

16.3.4 Utility-based Systems

Utility-based recommenders, such as example critiquing recommenders, propose items based on users' stated preferences on multi-attribute products. Multi-attribute products refer to the encoding scheme used to represent all available data with the same set of attributes $\{a_1, \dots, a_k\}$ where each attribute a_i can take any value v , from a domain of values $d(a_i)$. For example, a data set comprising all digital cameras in an e-store can be represented by the same set of attributes: manufacturer, price, resolution, optical zoom, memory, screen size, thickness, weight, etc. The list of at-

tributes as well as the domain range varies among product domains. We assume that users' preferences depend entirely on the values of these attributes so that two items that are identical in all attributes would be equally preferred. Furthermore, products considered here, such as digital cameras, portable PCs, or apartments, demand a significant financial commitment. They are called high involvement products because users are expected to possess a reasonable amount of willingness to interact with the system, participate in the selection process and expend a certain amount of effort to process information [62]. Users are also expected to exhibit slightly more complex decision behaviors in such environments than they would in selecting a simpler item, such as a book, a DVD, or a news article.

Tools using these technologies have also been referred to as knowledge-based recommenders [7] and utility-based decision support interface systems (DSIS) [62]. Utility refers to multi-attribute utility theory that such technologies use to calculate a product's suitability to a user's stated preferences. A related technology, specializing in searching configurable products, uses constraint satisfaction technology [45]. The difference between utility- and case-based systems lies in the notion of utility. While the weight of user preference is important for UBR, it is ignored in CBR. Further, the notion of value tradeoff is an essential part of decision making for UBRs. Essentially UBRs recommend decisions, rather than just similar products. See more details about this topic in the section on tradeoff reasoning.

16.3.5 Critiquing-based Systems

Both case- and utility-based recommenders can be improved by adding the additional interaction step of critiquing. A critiquing-based product recommender simulates an artificial salesperson that recommends options based on users' current preferences and then elicits their feedback in the form of critiques such as "I would like something cheaper" or "with faster processor speed." These critiques help the agent improve its accuracy in predicting users' needs in the next recommendation cycle. For a user to finally identify her ideal product, a number of such cycles are often required. Since users are unlikely to state all of their preferences up front, especially for products that are unfamiliar to them, the preference critiquing agent is an effective way to help them incrementally construct their preference model and refine it as they see more options.

16.3.6 Other Design Guidelines

This chapter derives a set of usability design guidelines based on our recent and related works in the domain of interaction technologies for preference-based search and recommender tools. Therefore, we will not review the related works that contribute to the accumulated list of guidelines in this section. Rather, discussions of

these works will be provided throughout the chapter in areas that correspond to the established guidelines. We will, however, describe two papers which share our goal of deriving good design guidelines for decision support systems. The first cited work proposes a set of recommendations derived from marketing research in order to increase users' motivation to interact with the recommender agent of an online store and its website. The second cited work describes a list of "building code" guidelines for an effective preference construction procedure in decision problems involving higher-stake outcomes.

Based on a critical but well-justified view of preference-based product search and recommender tools available at the time, Spiekermann and Paraschiv proposed a set of nine design recommendations to augment and stimulate the interaction readiness between a user and such systems [62], but there is no overlap between these recommendations and our guidelines.

Much of the basis for these recommendations are insights from marketing literature relative to information search, and perceived risk theory that defines a user's readiness to interact with a product recommender as her motivation to reduce the functional, financial, and emotional risks associated with the purchase decision. The design recommendations were therefore derived from methods concerned with reducing user risk in all of these dimensions. Compared to our work, this is a "horizontal" approach that covers the general design of an entire e-commerce website, in which the recommender agent is the principal technical component. We perform an in-depth examination of the recommender engine's interaction technologies using a more "vertical" approach, ensuring that consumers are offered the optimal usability support for preference elicitation and decision making when interacting with the product recommender tools.

Although the subject matter is more concerned with making high-stake decisions, the valuation process described by Payne et al. [41] is similar to our goal of addressing the needs and preferences of a consumer facing a purchase decision. Their work challenged many of the traditional assumptions about well-defined and pre-conceived preferences and strongly promoted a theory of preference construction. Under that new framework, the authors proposed a set of "building codes" (similar to guidelines) to help professional decision makers establish value functions and make high quality decisions. Their discussions on the nature of human preferences, the way people construct and measure preferences, and how to face tradeoffs to obtain rational decisions are especially influential to our work. References to the specific details of this and other works in behavior decision research will be given at relevant areas in this chapter.

16.4 Initial Preference Elicitation

Example critiquing systems are decision tools to help people find multi-attribute products such as flights, digital cameras, tablet PCs, etc. We use $P = \{(a_i, w_i)\}$ where $1 \leq i \leq n$ to specify a user's preferences over a total of n attributes of her

desired product for utility-based recommenders. Furthermore, a_i represents the desired characteristics on the i^{th} attribute and w_i the degree to which such characteristics should be satisfied. This model is also known as the value function in [24]. It is called the preference model in most systems discussed here. Some methods assume the same weights for all attributes and therefore do not elicit such information [9, 53]. Preference elicitation (also known as query specification) is the initial acquisition of this model for a given user.

It would seem apparent that a user's preferences could be elicited by simply asking her to state them. Many online search tools use a form-filling type of graphical user interface or a natural language dialog system to collect such information. Users are asked to state their preferences on every aspect, such as departure and arrival date and time, airlines, intermediate airports, etc., and are given the impression that all fields must be filled. We call such approaches non-incremental since all preferences must be obtained up-front.

To understand why this simple-minded approach does not work, we turn to behavior decision theory literature to understand the nature of user preference expression. According to the adaptive decision theory [40], user preferences are inherently adaptive and constructive depending on the current decision task and environment. Due to this nature, users may lack the motivation to answer demanding initial elicitation questions prior to any perceived benefits [62], and they may not have the domain knowledge to answer the questions correctly. In another words, if a system imposes a heavy elicitation process in the beginning, the preferences obtained in this way are likely to be uncertain and erroneous.

Similar literature reveals that users' preferences are context-dependent and are constructed gradually as a user is exposed to more domain information regarding his/her desired product [40, 41]. For example, Tversky et al. reported a user study about asking subjects to buy a microwave oven [66]. Participants were divided into 2 groups with 60 users each. In the first group, each user was asked to choose between an Emerson priced at \$110 and a Panasonic priced at \$180. Both items were on sale, and these prices represented a discount of one third off the regular price. In this case only 43% of the users chose the more expensive Panasonic at \$180. A second group was presented with the same choices except with an even more expensive item: a \$200 Panasonic, which represented a 10% discount from its original \$220 price. In this context, 60% of the users chose the Panasonic priced at \$180. In other words, more subjects prefer the same item just because the context has changed. This finding demonstrates that people are not likely to reveal their preferences as if they were innate to them, but construct them in an incremental and adaptive way based on contextual information.

To verify if these classical behavioral theories shed light on user preference expression in online environments, we conducted some empirical studies. 22 subjects were asked to interact with a preference elicitation interface [67]. The average user stated only preferences on 2.1 attributes out of a total of 10 when they had the possibility to freely state all preferences. Their preferences only increased to 4.19 attributes at the time of selecting the final choice.

Another study was conducted soon afterwards to further confirm the finding that users were unlikely to state all of their preferences in the beginning. It compared how users perform product search tasks in terms of decision accuracy and effort while interacting with a non-incremental procedure versus the incremental one [68]. With the former procedure, users were required to specify all of their preferences in a single graphical user interface (called form filling), whereas with the latter approach, each preference was constructed by a user. All 40 users were randomly and evenly divided into two groups, and each group was assigned to one system (either non-incremental or incremental approach) to evaluate. In the non-incremental approach, users stated on average 7.5 preferences on the 10 attributes, while the results for the incremental approach remained the same. However, the non-incremental approach had an accuracy of only 25%, while the incremental method achieved 70% accuracy with comparable user effort. That is, only 25% of users found the target products when they had to state preferences in the non-incremental form-filling style. Thus, while the non-incremental approach may produce the required data, the quality of what has been elicited may be questionable. There is no guarantee that users will provide correct and consistent answers on attributes for which their preferences are still uncertain.

Similar findings were reported in preference elicitation for collaborative filtering based recommender systems. McNee et al. compared three interface strategies for eliciting movie ratings from new users [34]. In the first strategy, the system asked the user to rate movies that were chosen based on entropy comparisons to obtain a maximally informative preference model. That is, the system decided which movies users should initially rate. In another strategy, users were allowed to freely propose movies they wanted to rate. In a mixed strategy, the user had both possibilities. A total of 225 new users participated in the experiment which found that the user-controlled strategy obtained the best recommendation accuracy compared to the other two strategies in spite of a lower number of ratings completed by each user (14 vs. 36 for the system-controlled interface). Furthermore, the user-controlled interface was more likely to motivate users to return to the system to assign more ratings. This demonstrates that a higher level of user control over the amount of interaction effort gives rise to more accurate preference models.

Incremental elicitation methods, as described in [1] and [5], can be used to revise users' stated preferences (or value functions) in general cases. Other incremental methods improve decision quality in specific areas. In [6, 36, 42], researchers addressed preference uncertainty by emphasizing the importance of displaying a diverse set of options in the early stage of user-system interaction. Faltings et al. [13, 69, 70] described ways to stimulate users to express more preferences in order to help them increase decision accuracy. Pu and Faltings showed how additional preference information can be acquired via preference revision, especially as users perform tradeoff tasks [45]. More detail on these topics will be given in Sections 16.5.2 and 16.6.

Based on both classical and recent empirical findings, we have derived several design guidelines concerning the initial preference elicitation process:

Guideline 1 (any effort): *Consider novice users' preference fluency. Allow them to reveal preferences incrementally. It is best to elicit initial preferences that concern them the most and choose an effort level that is compatible with their knowledge and experience of available options.*

A rigid elicitation procedure obtains users' preferences using a system pre-designed order of elicitation. When users are forced to formulate preferences in a particular order or using attributes that do not correspond to their actual objectives, they can fall prey to incorrectly formulating *means objectives* that prevent them from achieving their fundamental objectives [23]. For example, when planning a trip by airplanes a user may be prompted by the system to first enter her preferred airline company before being allowed to state her preferences on the flight. Thus, a user with the fundamental objective of flying at a certain hour has to formulate a different objective, the airline company to use, as a *means* of achieving the fundamental objective. This new objective is therefore called a means objective. To correctly translate the true objective into means objectives, the user needs to have detailed knowledge of the product offering, in this case the flight time tables offered by the different airlines. Since her knowledge in this area can be poor in the beginning, the system may fail to find the most optimal results.

Thus we propose

Guideline 2 (any order): *Consider allowing users to state their preferences in any order they choose.*

An elicitation procedure can also be too rigid by not allowing users to state preferences on a sufficiently rich set of attributes. Consider this example [70]: in a travel planning system, suppose that the user's objective is to be at his destination at 15:00, but that the tool only allows search by the desired departure time. The user might erroneously believe that the trip requires a plane change and takes about 5 hours, thus forming a means objective of a 10:00 departure in order to answer the question. However, the best option might be a new direct flight that leaves at 12:30 and arrives at 14:30.

Means objectives are intermediary goals formulated to achieve fundamental decision objectives. Assume that a user has a preference on attribute a_i (e.g., the arrival time), but the tool requires expressing preferences on attribute a_j (e.g., the departure time). Using beliefs about the available products, the user will estimate a transfer function $t_i(a_j)$ that maps values of attribute a_i to values of attribute a_j (e.g., arrival time = departure time + 5 hours). The true objective $p(a_i)$ is then translated into the means objective $q(a_j) = p(t(a_j))$. When the transfer function is inaccurate, the means objective often leads to very inaccurate results.

Note also that unless there is a strong correlation between attributes a_i and a_j , an accurate transfer function may not even exist. A recent visit to a comparison shopping website (www.pricegrabber.com) showed that the site does not include

the weight attribute of Tablet PCs in the search field, even though such information is encoded in their catalog. For many portability conscious users, the weight of a Tablet PC is one of the most important decision parameters. When unable to examine those products directly based on the weight attribute, the consumer might infer that weight is correlated to the screen size and perhaps the amount of storage in the hard drive. Consequently, she may consider searching for a Tablet PC with less disk space and a smaller screen size than she actually desires. These users could potentially make unnecessary sacrifices because many manufacturers are now able to offer light-weight Tablet PCs with considerable disk space and comfortable screen sizes.

Thus we propose

Guideline 3 (any preference): *Consider allowing users to state preferences on any attributes they choose.*

When designing interfaces based on these three guidelines, a good balance between giving the user the maximum amount of control, yet not overwhelming her with interface complexity is necessary. We recommend the use of adaptive interfaces, where users can click on attributes for which they want to state preferences or leave them unclicked or on default values if they do not have strong preferences at that point. Alternatively, designers can use a “ramp-up” approach for the initial query specification, where users specify preferences over a small number of attributes initially and are prompted to specify more once some results are displayed.

16.5 Stimulating Preference Expression with Examples

An effective elicitation tool should collect users’ preferences in an incremental and flexible way. In practice, we are also interested in mechanisms that stimulate users to state preferences.

Undesirable means objectives arise mainly due to users’ unfamiliarity with available options. At the same time, it has been observed in behavior theory that people find it easier to construct a model of their preferences when considering examples of actual options [41]. According to Tversky [65], people do not maximize a pre-computed preference order, but construct their choices in light of the available options. These classical theories seem to justify why example critiquing is an effective method for building preference elicitation tools [43, 47].

This method is called example critiquing since users build their preferences by critiquing the example products that are shown to them. Users initially state preferences on any number of attribute values that they determine to be relevant. From that point on, the system engages the user in successive cycles of “examples and critiques”: it displays a set of example products and elicits user feedback in the form of critiques such as “I like this laptop computer, but with more disk space”. The

critiques determine the set of examples to display next. This interaction terminates when users are able to identify their preferred products.

Users can quickly build their preferences by critiquing the example products shown to them. As users only have to state critiques rather than preferences, the model requires little effort from users. Most importantly, the example critiquing paradigm appears to satisfy both the goal of educating users with available options and the goal of stimulating them to construct their preferences in the context of given examples.

We will review a suite of critiquing systems and derive guidelines that illustrate the most effective components of this method. For additional information of this subject, please refer to the chapter by Lorraine McGinty and James Reilly entitled "On the Evolution of Critiquing Recommenders" (in this book).

Example critiquing was first mentioned in [72] as a new interface paradigm for database access, especially for novice users to specify queries. Recently, example critiquing has been used in two principal forms by several researchers: those supporting product catalog navigation and those supporting product search based on an explicit preference model.

In the first type of system, for example the FindMe systems [8, 9], search is described as a combination of search and browsing called assisted browsing. The system first retrieves and displays the best matching product from the database based on a user's initial query. It then retrieves other products based on the user's critiques of the current best item. The interface implementing the critiquing model is called tweaking, a technique that allows users to express preferences with respect to a current example, such as "look for an apartment similar to this, but with a better ambiance." According to this concept, a user navigates in the space of available products by tweaking the current best option to find her target choice. The preference model is implicitly represented by the current best product, i.e., what a user chooses reflects her preference of the attribute values. Reilly et al. have recently proposed dynamic critiquing [53] based on some improvements of the tweaking model. In addition to the unit-value tweaking operators, compound critiques allow users to choose products which differ from the current best item in two or more attribute values. For example, the system would suggest a digital camera based on the initial query. It also recommends cameras produced by different manufacturers, with less optical zoom, but with more storage. Compound critiques are generated by the Apriori algorithm [2] and allow users to navigate to their target choice in bigger steps. In fact, users who more frequently used the compound critiques were able to reduce their interaction cycles from 29 to 6 in a study involving real users [32].

In the second type of example-critiquing systems, an explicit preference model is maintained. Each user feedback in the form of a critique is added to the model to refine the original preference model. An example of a system with explicit preference models is the SmartClient system used for travel planning [43, 63]. It shows up to 30 examples of travel itineraries as soon as a set of initial preferences have been established. By critiquing the examples, users state additional preferences. These preferences are accumulated in a model that is visible to the user through the interface (see the bottom panel under "Preferences" of Figure 6 in [64]) and can be

revised at any time. ATA [28], ExpertClerk [60], the Adaptive Place Advisor [16], and the incremental dynamic critiquing systems function similarly [30]. The advantage of maintaining an explicit model is to avoid recommending products which have already been ruled out by the users. Another advantage is that a system can suggest products whose preferences are still missing in the stated model, as is further discussed in section 16.5.2.

Therefore, to educate users about the domain knowledge and stimulate them to construct complete and sound preferences, we propose the following guideline:

Guideline 4: *Consider showing example options to help users gain preference fluency.*

16.5.1 How Many Examples to Show

Two issues are critical in designing effective example-based interfaces: how many examples and which examples to show in the display. Faltings et al. investigated the minimum number of items to display so that the target choice is included even when the preference model is inaccurate [14]. Various preference models were analyzed. If preferences are expressed by numerical utility functions that differ from the true preferences by a factor of at most ϵ and they are combined using either the weighted sum or the min-max rule, then

$$t = \left(\frac{1 + \epsilon}{1 - \epsilon} \right)^d \quad (16.4)$$

where d is the maximum number of stated preferences, and t is the number of displayed items so that the target solution is guaranteed to be included. Since this number is independent of the total number of available items, this technique of compensating inaccurate preferences by showing a sufficient number of solutions scales to very large collections. For a moderate number (up to 5) of preferences, the correct amount of display items typically falls between 5 and 20. When the preference model becomes more complex, inaccuracies have much larger effects. A much larger number of examples is required to cover the model inaccuracy.

16.5.2 What Examples to Show

The most obvious examples to include in the display are those that best match the users' current preferences. However, this strategy proves to be insufficient to guarantee optimality. Since most users are often uncertain about their preferences and are more likely to construct them as options are shown to them, it becomes important

for a recommender system to guide the user to develop a preference model that is as complete and accurate as possible. However, it is important to keep the initiative to state more preferences on the user's side. Therefore we call examples chosen to stimulate users to state preferences *suggestions*. We present two suggestion strategies: diversity- and model-based techniques.

The ATA system was the first to show suggestions [28], which were extreme-valued examples where some attributes, for example departure time or price, took extreme values such as earliest or cheapest. However, a problem with this technique is that extreme options are not likely to appeal to many users. For example, a user looking for a digital camera with good resolution might not want to consider a camera that offers 4 times the usual resolution but also has 4 times the usual weight and price. In fact, a tool that suggests this option will discourage the user from even asking for such a feature, since it implies that high resolution can only exist at the expense of many other advantages.

Thus, it is better to select the suggestions among examples that are already good given the currently known preferences, and focus on showing *diverse* rather than extreme examples. Bradley and Smyth were the first to recognize the need to recommend diverse examples, especially in the early stage of using a recommender tool [6]. They proposed the bounded greedy algorithm for retrieving the set of cases most similar to a user's query, but at the same time most diverse among themselves. Thus, instead of picking the k best examples according to the preference ranking $r(x)$, a measure $d(x, Y)$ is used to calculate the relative diversity of an example x from the already selected set Y according to a weighted sum

$$s(x, Y) = \alpha r(x) + (1 - \alpha) d(x, Y) \quad (16.5)$$

where α can be varied to account for varying importance of optimality and diversity. For example, as a user approaches the final target, α can be set to a higher value (e.g. 0.75 in the experiment setup) so that the system emphasizes the display set's similarity rather than diversity. In their implementations, the ranking $r(x)$ is the similarity $sim(x, t)$ of x to an ideal example t on a scale of 0 to 1, and the relative diversity is derived as

$$d(x, Y) = 1 - \frac{1}{|Y|} \sum_{y \in Y} sim(x, y) \quad (16.6)$$

The performance of diversity generation was evaluated in simulations in terms of its relative benefit, i.e. the maximum gain in diversity achieved by giving up similarity [61]. Subsequently, McSherry has shown that diversity can often be increased without sacrificing similarity [36]. A threshold t was fixed on the ranking function, and then a maximally diverse subset among all products x for which $r(x) > t$ was selected. When k options are shown, the threshold might be chosen as the value of the k -th best option, thus allowing no decrease in similarity, or at some value that does allow a certain decrease.

We thus propose the following guideline:

Guideline 5: *Consider showing diverse examples to stimulate preference expression, especially when users are still uncertain about their final preferences.*

The adaptive search algorithm used in [33] alternates between a strategy that emphasizes similarity and one that emphasizes diversity to implement the interaction “show me more like this” by varying the α in the ranking measure. At each point, a set of example products is displayed and the user is instructed to choose her most preferred option among them. Whenever the user chooses the same option twice consecutively, the system considers diversity when proposing the next examples in order to refocus the search. Otherwise, the system assumes that the user is making progress and it continues to suggest new options based on optimality. Evaluations with simulated users show that this technique is likely to reduce the length of the recommendation cycles by up to 76% compared to the pure similarity-based recommender.

More recent work on diversity was motivated by the desire to compensate for users’ preference uncertainty [42] and to cover different topic interests in collaborative filtering recommenders [74]. For general preference models, it is less clear how to define a diversity measure. Viappiani et al. considered the user’s motivation to state additional preferences when a suggestion is displayed [69, 70, 50]. A suggestion is a choice that may not be optimal under the current preference model, but should provide a high likelihood of optimality when an additional preference is added. For example, a user may add “an apartment with a balcony” preference after seeing examples of such apartments. 40 (9 females) subjects from 9 different nationalities took part in a user study to search for an apartment. The experiment’s results show that the use of suggestions almost doubled decision accuracy and allowed the user to find the most preferred option 80% of the time. A user is likely to be opportunistic and will only bother to formulate new preferences if she believes that this might lead to a better choice. Thus, they propose the following *look-ahead principle* [69, 70, 50]:

Guideline 6: *Consider suggesting options that may not be optimal under the current preference model, but have a high likelihood of optimality when additional preferences are added.*

The look-ahead principle can be applied to constructing model-based suggestions by explicitly computing, for each attribute a_i , a difference measure $diff(a_i, x)$ that corresponds to the probability that a preference on this attribute would make option x most preferred. Items are then ranked according to the expected difference measure over all possible attributes:

$$F_a(x) = \sum_{a_i \in A} P_{a_i} diff(a_i, x) \quad (16.7)$$

where P_{a_i} is the probability that the user is motivated to state a preference on attribute a_i . Such probabilities are summed over all attributes for which the user has not yet expressed a preference. The best suggestions to display are therefore those items possessing the highest probability of becoming optimal after considering hidden preferences. It is possible to adapt these techniques to generate a set of suggestions that jointly maximize the probability of an optimal item. More details are given in [13, 50]. To investigate the importance of suggestions in producing accurate decisions, several empirical user studies were carried out [50, 69, 67]. One was conducted in an unsupervised setting, where users' behavior was monitored on a publicly accessible online system. The scientists conducting the experiment collected logs from 63 active users who went through several cycles of preference revision. Another study was carried out in a supervised setting. The scientists recruited 40 volunteers and divided them into two groups. One group evaluated the interface with model-based suggestions, and another group evaluated the one without. Both user studies showed the significant effects of using these model-based suggestions: users who used the suggestion interfaces stated significantly more preferences than those who did not (an increase of 2.09 preferences vs. only 0.62 without suggestions, $p < 0.01$, in supervised studies [69, 67], and an increase of 1.46 vs. 0.64 without suggestions, $p < 0.002$, for online users [69, 50]) and users who used the suggestion interfaces also reached significantly higher decision accuracy (80 vs. 45 percent without suggestions, $p < 0.01$, in supervised user studies [50]).

16.6 Preference Revision

Preference revision is the process of changing one or more desired characteristics of a product that a user has stated previously, the degree to which such characteristics should be satisfied, or any combination of the two. In [48] 28 subjects (10 females) were recruited to participate in a user study in which the user was asked to find his or her most preferred apartment from a list of available candidates. The user's preferences could be specified on a total of six attributes: type, price, area, bathroom, kitchen and distance to work place. Each participant was first asked to make a choice, and then used the decision aid tool to perform tradeoffs among his/her preferences until the desired item was chosen. In this user study, every user changed at least one initial preference during the entire search process for finding a product. Many users change preferences because there is rarely an outcome that satisfies all of the initial preferences. Two frequently encountered cases often require preference revision: 1) when a user cannot find an outcome that satisfies all of her stated preferences and must choose a partially satisfied one, or 2) when a user has too many possibilities and must further narrow down the space of solutions. Even though both activities can be treated as the process of query refinement, the real challenge is to help users specify the correct query in order to find the target item. Here we present a unified framework of treating both cases as a *tradeoff process* because finding an

acceptable solution requires choosing an outcome that is desirable in some respects but perhaps not so attractive in others.

16.6.1 Preference Conflicts and Partial Satisfaction

A user who inputs a query for a spacious apartment with a low price range and obtains “nothing found” as a reply, learns very little about how to state more suitable preferences.

The current industry practice manages preference conflicts by browsing-based interaction techniques. A user is only allowed to enter her preferences one at a time starting from the point where all of the product space is available. As she specifies more preferences, she essentially drills down to a sub product space until either she selects her target in the displayed options or no product space remains. For example, if someone desires a notebook with minimal weight (less than 2 kilos), then after specifying the weight requirement, she is only allowed to choose those notebooks weighing less than 2 kilos. If the price of these lightweight notebooks is very high, she is likely to miss a tradeoff alternative that may weigh 2.5 kilos and cost much less. This interaction style has become very popular in comparison shopping websites (see www.shopping.com, www.pricegrabber.com, www.yahoo.shopping.com). As the system designers have prevented users from specifying conflicting preferences, this interaction style is very limited. Users are unable to specify contextual preferences and especially tradeoffs among several attributes. If a user enters the set of preferences successively for each attribute, the space of matching products could suddenly become null with the message “no matching products can be found.” At this point, the user may not know which attribute value to revise among the set of values that she has specified so far, requiring her to backtrack several steps and try different combinations of preference values on the concerned attributes.

A more sensible method, such as the one used in SmartClient [43, 64], manages a user’s preference conflicts by first allowing her to state all of her preferences and then showing her options that maximally satisfy subsets of the stated preferences based on partial constraint satisfaction techniques [15]. These maximally satisfied products educate users about available options and facilitate them in specifying more reasonable preferences. In the same spirit, McCarthy et al. propose to educate users about product knowledge by explaining the products that do exist instead of justifying why the system failed to produce a satisfactory outcome [31]. FindMe systems rely on background information from the product catalog and explain preference conflicts on a higher level [8, 9]. In the case of a user wanting both a fuel-efficient and high-powered car, FindMe attempts to illustrate the tradeoff between horsepower and fuel efficiency. This method of showing partially satisfied solutions is also called soft navigation by Stolze [63].

To convince users of the partially satisfied results, we can also adopt the approach used by activedecision.com. It not only shows the partial solutions, but also explains in detail how the system satisfies some of users’ preferences and not oth-

Search Results

There is NO apartment completely satisfying all your preferences, but

these apartments are cheaper and bigger, although they are slightly farther

ID	Type	Price (Fs)	Area (m ²)	Bathroom	Kitchen	Distance (mins)	
27	shared apartment	450	25	private	private	20	Basket
30	room in a house	480	27	private	not available	20	Basket

More

these apartments are closer and bigger, although they are slightly more expensive

ID	Type	Price (Fs)	Area (m ²)	Bathroom	Kitchen	Distance (mins)	
77	shared apartment	550	25	private	not available	5	Basket
34	room in a house	600	30	shared	private	5	Basket

More

these apartments provide private bathrooms, although they are slightly smaller

ID	Type	Price (Fs)	Area (m ²)	Bathroom	Kitchen	Distance (mins)	
69	shared apartment	470	15	private	shared	10	Basket
72	shared apartment	500	12	private	shared	15	Basket

More

Fig. 16.2: Partially Satisfied Products in an Organization Interface.

ers. A qualitative user survey about such explanation mechanisms was conducted in the form of a carefully constructed questionnaire, based on a series of hypotheses and corresponding applicable questions. 53 participants completed the survey, and most of them strongly agreed that the explanation components are more likely to inspire their trust in the recommended solutions [10]. In addition, an alternative explanation technique, the organization interface where partially satisfied products are grouped into a set of categories (Figure 16.2), was preferred by most subjects, compared to the traditional method where each item is shown along with an explanation construct [10]. A follow-up comparative user study (with 72 participants) further proved that this interface method can significantly inspire competence-induced user trust in terms of the user's perceived competence, intention to return and intention to save effort (see some details of the experiment in section 7.3) [49].

Guideline 7: Consider resolving preference conflicts by showing partially satisfied results with compromises clearly explained to the user.

16.6.2 Tradeoff Assistance

As catalogs grow in size, it becomes increasingly difficult to find the target item. Users may achieve relatively low decision accuracy unless a tool helps them efficiently view and compare many potentially interesting products. Even though a recommender agent is able to improve decision quality by providing filtering and

comparison matrix components [20], a user can still face the bewildering task of selecting the right items to include in the consideration set.

Researchers in our group found that online tools could increase the level of decision accuracy by up to 57% by helping users select and compare options which share tradeoff properties [48]. 28 subjects (10 females) took part in the experiment; each of the participants was first asked to make a choice, and then use the decision aid tool to perform a set of tradeoff navigation tasks. The results showed that after a user has considered an item as the final candidate, the tool can help her/him reach higher decision accuracy by prompting them to see a set of tradeoff alternatives. The same example critiquing interfaces as discussed in Section 5 can be used to assist users to view tradeoff alternatives, for example, “I like this portable PC, but can I find something lighter?” This style of interaction is called tradeoff navigation and is enabled by the “modify” widget together with the “tweaking panel” (see Figure 4 in [47]). Tweaking (used in FindMe [8, 8]) was the first tool to implement this tradeoff assistance. It was originally designed to help users navigate to their targets by modifying stated preferences, one at a time. Example critiquing (used in SmartClient [48, 47]) is more intentional about its tradeoff support, especially for tradeoffs involving more than two participating attributes. In a single interaction, a user can state her desire to improve the values of certain attributes, compromise on others, or any combination of the two.

Reilly et al. introduced another style of tradeoff support with dynamic critiquing methods [53]. Critiques are directional feedback at the attribute level that users can select in order to improve a system’s recommendation accuracy. For example, after recommending a Canon digital camera, the system may display “we have more matching cameras with the following: 1) less optimal zoom and thinner and lighter weight; 2) different manufacturer and lower resolution and cheaper; 3) larger screen size and more memory and heavier.” Dynamic critiquing is an approach of automatically generating useful compound critiques so that users can indicate their preference on multiple attributes simultaneously. The experiment in [53] shows that the dynamic critiquing approach has the ability to reduce the interaction session length by up to 40% compared to the approach with only unit critiques.

Although originally designed to support navigation in recommender systems, the unit and compound critiques described in [53] correspond to the simple and complex tradeoffs defined in [47]. They are both mechanisms to help users compare and evaluate the recommended item with a set of tradeoff alternatives. However, the dynamic critiquing method provides system-proposed tradeoff support because it is the system which produces and suggests the tradeoff categories, whereas example critiquing provides a mechanism for users to initiate their own tradeoff navigation (called user motivated critiques in [11]).

A recent study from our group compared the performance of user-motivated vs. system-proposed approaches [11]. A total of 36 (5 females) volunteers participated in the experiment. It was performed in a within-subjects design, and each participant was asked to evaluate two interfaces with the respective two approaches one after the other. All three evaluation criteria stated in section 2.2 were used: decision accuracy, user interaction effort and user confidence. The results indicate that

the user-motivated tradeoff method enables users to achieve a higher level of decision accuracy with less cognitive effort, mainly due to its flexibility in allowing users to freely combine unit and compound critiques. In addition, the confidence in a choice made with the user-motivated critique method is higher, resulting in users' increased intention to purchase the product they have found and return to the agent in the future. We thus propose:

Guideline 8: *In addition to providing the search function, consider providing users with tradeoff assistance in the interface using either system-proposed or user motivated approaches. The latter is likely to provide users with more flexibility in choosing their tradeoff desires and thus enable them to achieve higher decision accuracy and confidence.*

16.7 Display Strategies

At least three display strategies are currently employed in preference-based search and recommender tools: recommending items one at a time, showing top k matching results (where k is a small number between 3 and 30), or displaying products with explanations on how ranking scores are computed. We discuss these various strategies using the accuracy, confidence, and effort evaluation framework discussed in Section 2.

16.7.1 Recommending One Item at a Time

The advantage of such recommender systems is that it is relatively easy to design the display, users are not likely to be overwhelmed by excessive information, and the interface can be easily adapted to small display devices such as mobile phones. The obvious disadvantage is that a user may not be able to find her target choice quickly. As mentioned in Section 5, a novice user's initial preferences are likely to be uncertain. Thus the initially recommended results may not include her target choice. Either a user has to interact with the system much longer due to the small result set, or if a user exhausts her interaction effort before reaching the final target, she is likely to achieve very low decision accuracy. Thus we propose:

Guideline 9: *Showing one search result or recommending one item at a time allows for a simple display strategy which can be easily adapted to small-display devices; however, it is likely to engage users in longer interaction sessions or only allow them to achieve relatively low decision accuracy.*

16.7.2 *Recommending K best Items*

Some product search tools present a set of top- k alternatives to the users. We call this style of display the k -best interface. Commercial tools employing this strategy can be found at ActiveDecision.com ($k > 10$). Academic prototypes include those used by SmartClient ($7 \leq k \leq 30$) [14, 47], ATA ($k = 3$) [28], ExpertClerk ($k = 3$) [60], FirstCase ($k = 3$) [37] and TopCase ($k = 3$) [38].

When k approaches 10, the issue of ordering the alternatives becomes important. The most commonly used method is to select the best k items based on how well they match users' stated preferences using utility scores (see multi-attribute utility theory [24]). We can also use the " k nearest neighbor" retrieval algorithm (or simply k -NN) [12] to rank the k items, such as those used in the case-based reasoning field [25]. The k items are displayed in descending order from the highest utility score or rank to the lowest (activedecision.com, SmartClient). This method has the advantages of displaying a relatively high number of options without overwhelming the users, pre-selecting the items based on how well they match the stated preferences of a user, and achieving relatively high decision accuracy [48, 47].

Pu and Kumar compared an example critiquing based system ($k = 7$ rank ordered by utility scores) with a system using the ranked list display method ($k = n$ rank ordered on user selected attribute values such as price) [47, 51]. 22 volunteers participated in the user study. Each of them was asked to test two interfaces (example critiquing and ranked list) in random order by performing a list of given tasks. The results showed that while users performed the instructed search tasks more easily using example critiquing (less task time and smaller error rate, with statistical significance) and achieved higher decision accuracy [48, 51], more of them expressed a higher level of confidence that the answers they found were correct for the ranked list interface. Further analysis of users' comments recorded during the user study revealed that the confidence issue depends largely on the way items were ordered and how many of them were displayed. Many users felt that the EC system (displaying only 7 items) was hiding something from them and that the results returned by the EC interface did not correspond to their ranking of products. With the help of a pilot study, it was observed that users generally did not scroll down to view the additional products displayed, but their confidence level increased and the interaction time was not affected. Therefore we suggest the following guideline for the top- k display strategy:

Guideline 10: *Displaying more products and ranking them in a natural order is likely to increase users' sense of control and confidence.*

16.7.3 Explanation Interfaces

When it comes to suggesting decisions, such as which camera to buy, the recommender system's ability to establish trust with users and convince them of its recommendations is a crucial factor for success. Researchers from our group started investigating the user confidence issue and other subjective factors in a more formal framework involving trust relationships between the system and the user. It is widely accepted that trust in a technological artifact (like the recommender agent) can also be conceptualized as competence, benevolence, and integrity, similar to trust in a person. Trust is further seen as a long term relationship between the user and the organization that the recommender system represents [10]. When a user trusts a recommender system, she is more likely to purchase items and return to the system in the future. A carefully designed qualitative survey with 53 users revealed that an important construct of trust formation is an interface's ability to explain its results [10], as mentioned in section 6.1.

The explanation interface can be implemented in various ways. For example, ActiveDecision.com uses the tool tip with a "why" label to explain how each of the recommended products matches a user's stated preferences, similar to the interface shown in Figure 16.3. Alternatively, it is possible to design an organization-based explanation interface where the best matching item is displayed at the top of the interface along with several categories of tradeoff alternatives [49]. Each category is labeled with a title explaining the characteristics of the items the respective category contains (Figure 16.4).

In order to understand whether the organization interface is a more effective way to explain recommendations, a significant-scale empirical study was conducted to compare the organization interface with the traditional "why" interface in a within-subjects design. A total of 72 volunteers (19 females) were recruited as participants in the user study. The results showed that the organization interface significantly increases user perception of its competence, which more effectively inspires users' trust and enhances their intention to save cognitive effort and use the interface again in the future [49]. Moreover, the study found that the actual time spent looking for a product did not have a significant impact on users' subjective perceptions. This indicates that less time spent on the interface, while very important in reducing decision effort, cannot be used alone in predicting what users may subjectively experience. Five principles for the effective design of organization interfaces were developed and an algorithm was presented for generating the content of such interfaces [49]. Here we propose:

Guideline 11: *Consider designing interfaces that explain how ranking scores are computed because they are likely to inspire user trust.*

<input type="radio"/>	Why?	-	\$1'379.00	3.3 GHz	2 hours
<input type="radio"/>	Why?	-	\$1'179.00	3.2 GHz	2 hours
<input type="radio"/>	Why?	-	\$1'529.00	1.7 GHz	6.5 hours
<input type="radio"/>	Why?	-	\$1'599.00	1.7 GHz	6.5 hours
<input type="radio"/>	Why?	-	\$1'425.00	1.6 GHz	5.5 hours
<input type="radio"/>	Why?	-	\$2'235.00	1.8 GHz	2.5 hours
<input type="radio"/>	Why?	-	\$1'190.00	3.2 GHz	1 hours
<input type="radio"/>	Why?	-	\$1'125.00	1.5 GHz	6 hours
<input type="radio"/>	Why?	-	\$2'319.00	1.67 GHz	4.5 hours
<input type="radio"/>	Why?	-	\$1'499.00	1.5 GHz	5 hours
<input type="radio"/>	Why?	-	\$1'739.99	1.5 GHz	4.5 hours
<input type="radio"/>	Why?	-	\$1'629.00	1.8 GHz	5.8 hours
<input type="radio"/>	Why?	-	\$1'625.99	1.5 GHz	5 hours
<input type="radio"/>	Why?	-	\$1'426.99	1.5 GHz	5 hours

Fig. 16.3: A generic recommendation interface with simple “why” labels.

16.8 A Model for Rationalizing the Guidelines

We have developed a set of guidelines to ensure the design of usable product search tools relying on a general framework of three evaluation criteria: (i) decision accuracy, (ii) user interaction effort, and (iii) user decision confidence. We now develop a model of user behavior that allows us to rationalize the guidelines as a means of optimizing recommender system performance. We first consider the fact that product search tools must serve the needs of a significant and heterogeneous user population. They must adapt to the characteristics of individual users, in particular to their willingness to put in continuous interaction effort in order to obtain their desired results. In our theoretical model, we let e be the user’s interaction effort, measured in interaction steps. With this effort, users hope to obtain an increasingly high confidence that they are finding the best option. We characterize this user confidence numerically as the accuracy that the user believes the system to have achieved, called the perceived accuracy a .

As each individual user has different aspirations for effort and perceived accuracy, we characterize each user by the three following parameters:

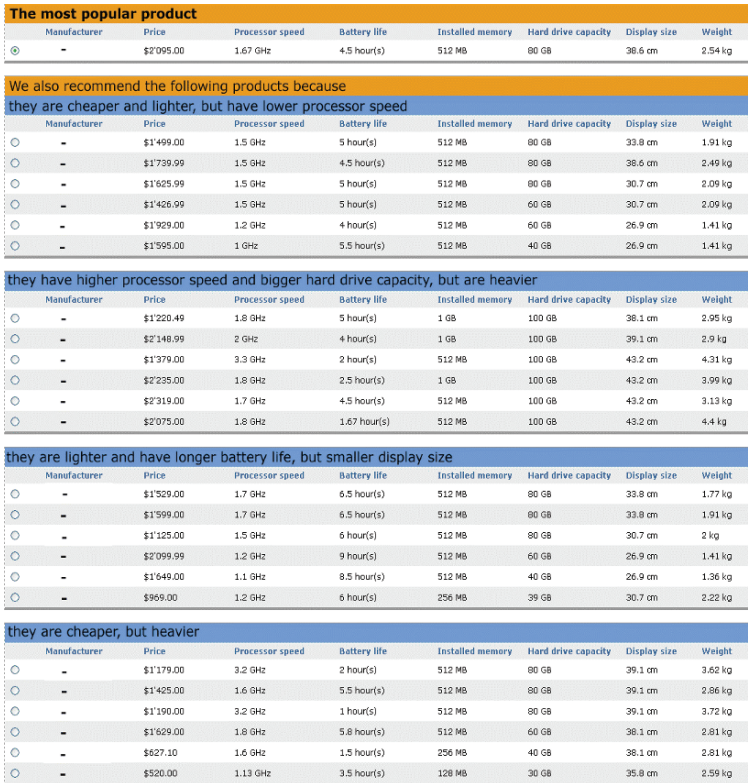


Fig. 16.4: The more trust-inspiring organization interface.

- *Confidence threshold* θ : the amount of perceived accuracy that is required for the user to be satisfied with the current result of the search process and buy the product;
- *Effort threshold* ϵ : the amount of effort the user is willing to spend to obtain a recommendation;
- *Effort increment threshold* δ : the amount of additional perceived accuracy that is required to motivate the user to spend an additional interaction cycle.

A poorly designed tool can lose users due to an insufficient level of confidence for the confidence threshold, or an increase in confidence that is too small to justify the interaction effort. Figure 16.5 shows the perceived accuracy achieved by a hypothetical recommender tool as a function of effort, measured in interaction cycles. The confidence threshold is indicated by a horizontal dashed line, and the user will not buy the product unless the perceived accuracy exceeds it. The effort threshold ϵ and effort increment threshold δ characterize the amount of effort a user is willing to spend. It is characterized by another dashed line labelled as the *effort limit* in Fig-

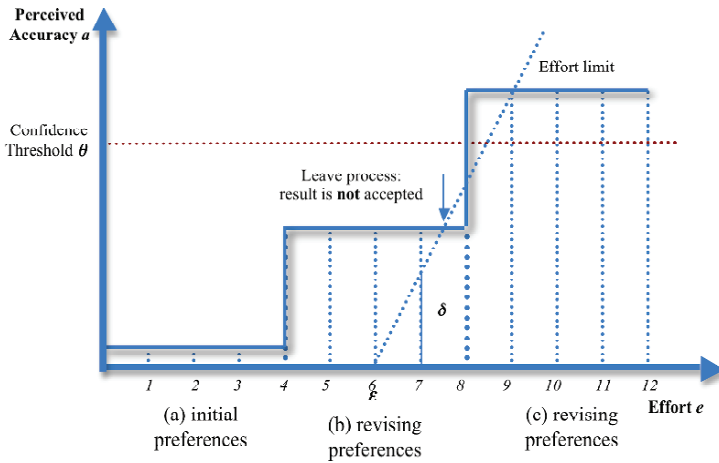


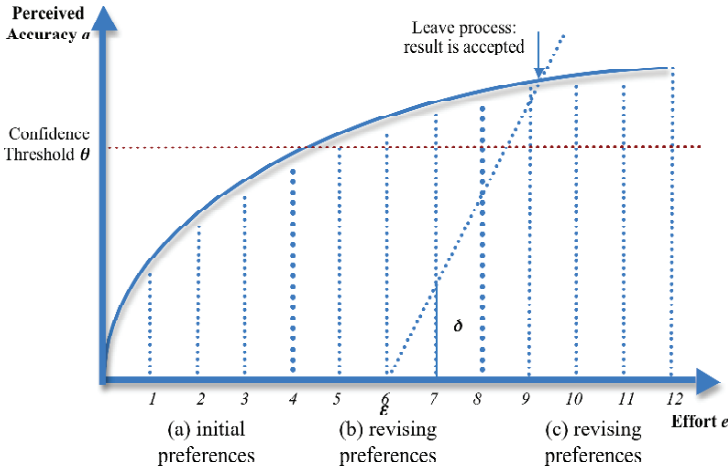
Fig. 16.5: Perceived accuracy as a function of interaction effort for an interview-based tool.

ure 16.5. The user will leave the process as soon as the accuracy/effort curve falls below the effort limit.

Figure 16.5 assumes a tool where users are asked to specify their preferences in a dialogue interface consisting of a list of questions. Such an interface requires a significant amount of effort of preference elicitation before any result is shown, and thus the perceived accuracy remains low until the first results are obtained. The process then proceeds by stages of revising preferences and obtaining successively better perceived accuracy in a stepwise manner. Note that in this example, the user will leave the process before reaching the confidence threshold and the interaction has thus not been a success. With slightly more patience, the user could have obtained a result above the confidence threshold and with acceptable effort, namely at the third intersection of the effort limit line with the curve. However, since this was not apparent to the user, this point will not be reached. Similar problems can occur with other interface designs that do not pay attention to ensuring a significant increase in perceived accuracy per effort invested.

To avoid this pitfall, the tool should ensure that the perceived accuracy is a concave function of effort, as shown in Figure 16.6. Such a function exploits a user’s effort threshold to the best possible degree: if the increase in perceived accuracy becomes insufficient to keep the user interested in using the tool, she will not interact with the tool at a later stage either.

A concave function could only be achieved by ensuring instant feedback to the user’s effort, and by placing the steps that result in the greatest increase in perceived accuracy at the beginning of the interaction. An early increase in perceived accuracy also serves to convince the user to stay with the system longer, as indicated by the dashed line in Figure 16.6. Instant feedback is ensured by example-based



Guidelines for (a)	Guidelines for (b)	Guidelines for (c)
1. Any effort; 2. Any order; 3. Any preferences.	4. Showing example options; 5. Showing diverse examples; 6. Suggesting options with look-ahead principle.	7. Preference conflict management; 8. Tradeoff assistance.
Guidelines for (a) – (c) 9. Showing one search result at a time is good for small-display devices, but it is likely to achieve relatively low decision accuracy; 10. Displaying more products and ranking them in a natural order is likely to increase users' sense of control and confidence; 11. Designing interfaces which are capable of explaining how ranking scores are computed can inspire user trust.		

Fig. 16.6: Perceived accuracy as a function of interaction effort for an example-based tool, and guidelines that apply to achieving the desired concave shape in the different stages.

interaction and the general guidelines 9-11 of showing multiple solutions in a structured and confidence-inspiring way. In general, it can be assumed that users will themselves choose to add the information that they believe to maximize their decision accuracy, and so user initiative is key to achieving a concave curve. In the first phase (a), the system can achieve the biggest accuracy gains by exploiting users' initial preferences. However, it is important at this stage to avoid asking them questions that they cannot accurately answer (guideline 1). Furthermore, the curve can be made steeper by letting the user formulate these initial preferences with as little effort as possible. We therefore derived guidelines (2 and 3) to make this possible.

Once these initial preferences have been obtained, the biggest increase in perceived accuracy during phase (b) can be obtained by completing the initial prefer-

ences with others of which the user was not initially aware. This can be stimulated by showing examples (guideline 4), and by choosing them to specifically educate the user about available options (guidelines 5 and 6). This provides the main cost-effect tradeoff for the second phase of a typical interaction. Finally, in the third phase (c) the set of preferences can be fine-tuned by adjusting their relative weights and making tradeoffs. This can be supported by tools that show partial solutions (guideline 7) and actively support decision tradeoffs among preferences (guideline 8).

As the tool cannot verify when the user transitions between the phases, and in fact the transition may be gradual, it should provide continuous support for each of them, but always encourage actions that are likely to increase perceived accuracy as much as possible. Thus, adjustment of tradeoff weights should be shown less prominently than the possibility to add new preferences.

These requirements are best addressed by the example-based recommender tools as described in Section 5. More precisely, the incremental establishment and refinement of the user's preference model increases the true decision accuracy. To keep the user engaged in the interaction process and convinced to accept the result of the search, this true accuracy must also be perceived by the user. This is supported by showing several results at the same time (guideline 9), which tends to correct inaccuracies, by providing structure to their display (guideline 10), and by providing explanations (guideline 11). These are necessary elements to motivate users to put in enough effort in achieving an accurate decision as much as possible.

16.9 Conclusion

This chapter presents eleven essential guidelines that should be observed when designing interactive preference-based recommender systems. In presenting and justifying the guidelines, we provided a broad and in-depth review of our prior work related to *example critiquing* regarding user interaction issues in recommender systems. Most importantly, a framework of three evaluation criteria was proposed to determine the usability of such systems: decision accuracy, user confidence, and user interaction effort (ACE). Within this framework, we have selected techniques, which have been validated through empirical studies, to demonstrate how to implement the guidelines. Emphasis was given to those techniques that achieve a good balance on all of the criteria. Adopting these guidelines, therefore, should significantly enhance the usability of product recommender systems and consequently the wide adoption of such systems in e-commerce environments.

References

1. Adomavicius, G., Tuzhilin, A., Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data*

- Engineering* 17 (6) (2005) 734-749.
2. Agrawal, R., Imielinski, T., Swami, A., Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ACM Press, 1993, 207216.
 3. Belkin, N.J., Croft, W.B., Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM* 35 (12) (1992) 29-38.
 4. Bernoulli, D., Exposition of a new theory on the measurement of risk (original 1738). *Econometrica* 22 (1) (1954) 23-36.
 5. Blythe, J., Visual exploration and incremental utility elicitation. *Proceedings of the 18th National Conference on Artificial Intelligence*, AAAI press, 2002, 526-532.
 6. Bradley, K., Smyth, B., Improving recommendation diversity. *Proceedings of the 12th Irish Conference on Artificial Intelligence and Cognitive Science*, 2001, 85-94.
 7. Burke, R., Hybrid recommender systems: survey and experiments. *User Modeling and User-Adapted Interaction* 12 (4) (2002) 331-370.
 8. Burke, R., Hammond, K., Cooper, E., Knowledge-based navigation of complex information spaces. *Proceedings of the 13th National Conference on Artificial Intelligence*, AAAI press, 1996, 462-468.
 9. Burke, R., Hammond, K., Young, B., The FindMe approach to assisted browsing. *IEEE Expert: Intelligent Systems and Their Applications* 12 (4) (1997) 32-40.
 10. Chen, L., Pu, P., Trust building in recommender agents. *Proceedings of the Workshop on Web Personalization, Recommender Systems and Intelligent User Interfaces at the 2nd International Conference on E-Business and Telecommunication Networks*, 2005, 135-145.
 11. Chen, L., Pu, P., Evaluating critiquing-based recommender agents. *Proceedings of Twenty-first National Conference on Artificial Intelligence (AAAI-06)*, 2006, 157-162.
 12. Cover, T.M., Hart, P.E., Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13, 1967, 21-27.
 13. Faltings, B., Pu, P., Torrens, M., Viappiani, P., Designing example-critiquing interaction. *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI'04)*, ACM Press, 2004, 22-29.
 14. Faltings, B., Torrens, M., Pu, P., Solution generation with qualitative models of preferences. *Computational Intelligence* 20 (2) (2004), 246-263.
 15. Freuder, E.C., Wallace, R.J., Partial constraint satisfaction. *Artificial Intelligence* 58 (1-3) (1992) 21-70.
 16. Goker, M., Thompson, C., The adaptive place advisor: a conversational recommendation system. *Proceedings of the 8th German Workshop on Case Based Reasoning*, 2000.
 17. Goldberg, D., Nichols, D., Oki, B.M., Terry, D., Using collaborative filtering to weave an information tapestry. *Communications of the ACM* (35) 12, Special issue on information filtering (1992) 61-70.
 18. Good, N., Schafer, J.B., Konstan, J.K., Borchers, A., Sarwar, B.M., Herlocker, J.L., Riedl, J., Combining collaborative filtering with personal agents for better recommendations. *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'99)*, AAAI press, 1999, 439-446.
 19. Ha, V.A., Haddawy, P., Problem-focused incremental elicitation of multi-attribute utility models. In Shenoy, P. (ed.), *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI'97)*, 1997, 215-222.
 20. Haubl, G., Trifts, V., Consumer decision making in online shopping environments: the effects of interactive decision aids. *Marketing Science* 19 (1) (2000) 4-21.
 21. Herstein, I.N., Milnor, J., An axiomatic approach to measurable utility. *Econometrica* 21 (2) (1953) 291-297.
 22. Hill, W., Stead, L., Rosenstein, M., Furnas, G., Recommending and evaluating choices in a virtual community of use. *Proceedings of the CHI '95 Conference on Human Factors in Computing Systems*, 1995, 194-201.
 23. Keeney, R.L., *Value-Focused Thinking: A Path to Creative Decision Making*, Harvard University Press (1992).

24. Keeney, R.L., Raiffa, H., *Decisions with Multiple Objectives: Preferences and Value Trade-offs*, New York: Wiley (1976).
25. L. J., Kolodner. *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann (1993).
26. Krulwich, B., Lifestyle finder: intelligent user profiling using large-scale demographic data. *Artificial Intelligence Magazine* 18 (2) (1997) 37-45.
27. Lang, K., Newsweeder: learning to filter news. *Proceedings of the 12th International Conference on Machine Learning*, 1995, 331-339.
28. Linden, G., Hanks, S., Lesh, N., Interactive assessment of user preference models: the automated travel assistant. *Proceedings of the 6th International Conference on User Modeling (UM'97)*, New York: Springer Wien New York, 1997, 67-78.
29. Marschak, J., Rational Behavior, Uncertain Prospects, and Measurable Utility. *Econometrica* 18 (2) (1950) 111-141.
30. McCarthy K., McGinty L., Smyth, B., Reilly, J., A live-user evaluation of incremental dynamic critiquing. *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR'05)*, 2005, 339-352.
31. McCarthy K., Reilly, J., L. McGinty, Smyth, B., Thinking positively explanatory feedback for conversational recommender systems. *Proceedings of the Workshop on Explanation in CBR at the 7th European Conference on Case-Based Reasoning (ECCB'04)*, 2004, 115-124.
32. McCarthy K., Reilly, J., L. McGinty, Smyth, B., Experiments in dynamic critiquing. *Proceedings of the 10th International Conference on Intelligent User Interfaces (IUI'05)*, New York: ACM Press, 2005, 175-182.
33. McGinty L., Smyth, B., On the role of diversity in conversational recommender systems. *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR'03)*, 2003, 276-290.
34. McNee S.M., Lam, S.K., Konstan, J., Riedl, J., Interfaces for eliciting new user preferences in recommender systems. *Proceedings of User Modeling Conference*, Springer, 2003, 178-187.
35. McNee S.M., Riedl, J., Konstan, J., Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI '06)*, ACM, New York, NY, 1097-1101.
36. McSherry, D., Diversity-conscious retrieval. In, Craw, S., Preece, A. (eds.), *Proceedings of the 6th European Conference on Advances in Case-Based Reasoning*, London: Springer-Verlag, 2002, 219-233.
37. McSherry, D., Similarity and compromise. *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR'03)*, Springer-Verlag, 2003, 291-305.
38. McSherry, D., Explanation in recommender systems. *Workshop Proceedings of the 7th European Conference on Case-Based Reasoning (ECCBR'04)*, 2004, 125-134.
39. Mongin, P., *Expected Utility Theory*. Handbook of Economic Methodology, Edward Elgar, 1998, 342-350.
40. Payne, J.W., Bettman, J.R., Johnson, E.J., *The Adaptive Decision Maker*, Cambridge University Press (1993).
41. Payne, J.W., Bettman, J.R., Schkade, D.A., Measuring constructed preferences: towards a building code. *Journal of Risk and Uncertainty* 19 (1999) 243-270.
42. Price, B., Messinger, P.R., Optimal recommendation sets: covering uncertainty over user preferences. *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, 2005, 541-548.
43. Pu, P., Faltings, B., Enriching buyers' experiences: the SmartClient approach. *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI'00)*, New York: ACM Press, 2000, 289-296.
44. Pu, P., Faltings, B., Torrens, M., User-involved preference elicitation. *Working Notes of the Workshop on Configuration. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003, 56-63.
45. Pu, P., Faltings, B., Decision tradeoff using example-critiquing and constraint programming. *Constraints: an International Journal* 9 (4) (2004) 289-310.

46. Pu, P., Faltings, B., Torrens, M., Effective interaction principles for online product search environments. *Proceedings of the IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology and Web Intelligence*, 2004, 724-727.
47. Pu, P., Kumar, P., Evaluating example-based search tools. *Proceedings of the 5th ACM Conference on Electronic Commerce (EC'04)*, ACM Press, 2004, 208-217.
48. Pu, P., Chen, L., Integrating tradeoff support in product search tools for e-commerce sites. *Proceeding of the 6th ACM Conference on Electronic Commerce (EC'05)*, ACM Press, 2005, 269-278.
49. Pu, P., Chen, L., Trust building with explanation interfaces. *Proceedings of the 11th International Conference on Intelligent User Interface (IUI'06)*, 2006, 93-100.
50. Pu, P., Viappiani, P., Faltings, B., Stimulating decision accuracy using suggestions. *SIGCHI conference on Human factors in computing systems (CHI'06)*, 2006, 121-130.
51. Pu P., Chen L. and Kumar P., Evaluating Product Search and Recommender Systems for E-Commerce Environments. *Electronic Commerce Research Journal* 8(1-2), June 2008, 1-27.
52. Pu P., Chen L., User-Involved Preference Elicitation for Product Search and Recommender Systems. *AI Magazine* 29(4), Winter 2008, 93-103.
53. Reilly, J., K. McCarthy, L. McGinty, Smyth, B., Dynamic critiquing. *Proceedings of the 7th European Conference on Case-Based Reasoning (ECCBR'04)*, Springer, 2004, 763-777.
54. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J., GroupLens: an open architecture for collaborative filtering of Netnews. *CSCW '94: Conference on Computer Supported Cooperative Work*, ACM, 1994, 175-186.
55. Rich, E., User modeling via stereotypes. *Cognitive Science* 3 (1979) 329-354.
56. Rokach, L., Maimon, O., Averbuch, M., Information Retrieval System for Medical Narrative Reports, Lecture Notes in Artificial intelligence 3055, page 217-228 Springer-Verlag, 2004.
57. Schafer, J.B., Konstan, J.A., Riedl, J., Recommender systems in e-commerce. *Proceedings of the ACM Conference on Electronic Commerce*, ACM, 1999, 158-166.
58. Shardanand, U., Maes, P., Social information filtering: algorithms for automating "Word of Mouth". *Proceedings of the Conference on Human Factors in Computing Systems (CHI '95)*, 1995, 210-217.
59. Schoemaker, P., The Expected Utility Model: Its Variants, Purposes, Evidence and Limitations. *Journal of Economic Literature* 20 (2) (1982), 529-563.
60. Shimazu, H., ExpertClerk: navigating shoppers' buying process with the combination of asking and proposing. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001, 1443-1448
61. Smyth, B., P. McClave. Similarity vs. diversity. *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR'01)*, Springer-Verlag, 2001, 347-361.
62. Spiekermann, S., Parachiv, C., Motivating human-agent interaction: transferring insights from behavioral marketing to interface design. *Journal of Electronic Commerce Research* 2 (3) (2002) 255-285.
63. Stolze, M., Soft navigation in electronic product catalogs. *International Journal on Digital Libraries* 3 (1) (2000) 60-66.
64. Torrens, M., Faltings, B., Pu, P., SmartClients: constraint satisfaction as a paradigm for scaleable intelligent information systems. *International Journal of Constraints* 7 (1) (2002) 49-69.
65. Tversky, A., *Contrasting rational and psychological principles in choice. Wise Choices: Decisions, Games, and Negotiations*, Boston, MA: Harvard Business School Press (1996) 5-21.
66. Tversky, A., Simonson, I., Context-dependent preferences. *Management Science* 39 (10) (1993) 1179-1189.
67. Viappiani, P., Faltings, B., V. Schickel-Zuber, Pu, P., Stimulating preference expression using suggestions. Mixed-Initiative Problem-Solving Assistants, *AAAI Fall Symposium Series*, AAAI press, 2005, 128-133.
68. Viappiani, P., Faltings, B., Pu, P., Evaluating preference-based search tools: a tale of two approaches. *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI-06)*, 2006, 205-210.

69. Viappiani, P., B. Faltings and Pu, P., Preference-based Search using Example-Critiquing with Suggestions. *Journal of Artificial Intelligence Research (JAIR)*, 27, 2006, 465-503.
70. Paolo Viappiani, Pearl Pu, and Boi Faltings. Preference-based Search with Adaptive Recommendations. *AI Communications* 21(2-3), 2008, 155-175.
71. J. von Neumann, Morgenstern, O., *The Theory of Games and Economic Behavior*, Princeton University Press (1944).
72. Williams, M.D., Tou, F.N., RABBIT: an interface for database access. *Proceedings of the ACM '82 Conference*, ACM Press, 1982, 83-87.
73. Zhang, J., Pu, P., Performance evaluation of consumer decision support systems. *International Journal of E-Business Research 2 (2006)* Idea Group Publishing.
74. Ziegler, C.N., S.M. McNee, Konstan, J.A., Lausen, G., Improving recommendation lists through topic diversification. *Proceedings of the 14th International World Wide Web Conference (WWW'05)*, 2005, 22-32.
75. Zukerman, I., Albrecht, D.W., Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction 11 (1-2)* (2001) 5-18.

Chapter 17

Map Based Visualization of Product Catalogs

Martijn Kagie, Michiel van Wezel and Patrick J.F. Groenen

17.1 Introduction

Traditionally, recommender systems present recommendations in ranked lists to the user. In content- and knowledge-based recommender systems, these lists are often sorted on some notion of similarity with a query, ideal product specification, or sample product. However, a lot of information is lost in this way, since two products with the same similarity to a query can differ from this query on a completely different set of product characteristics. When using a two dimensional map based visualization of the recommendations, it is possible to retain part of this information. In the map, we can then position recommendations that are similar to each other in the same area of the map.

A domain in which this map based approach can be very useful is electronic commerce, since electronic commerce stores sell a large number of products, often described by a large number of characteristics, but that are, in certain product categories, easily recognizable by an image of the product at hand. E-commerce domains for which this holds are, for example, consumer electronics and real estate. In the industry, one tries nowadays to improve usability of this kind of websites, using a trend called visual shopping. An example is CrispyShop.com¹, which compares products on one characteristic and price using a chart. Another approach taken in visual shopping is selecting items based on visual similarity (like color and shape), such as is done by Like.com² and Modista³ which is especially useful in fashion related fields. An approach more closely related to the map based approach, is taken

Martijn Kagie · Michiel van Wezel · Patrick J.F. Groenen
Econometric Institute, Erasmus University Rotterdam, The Netherlands, e-mail: martijn@kagie.net, mvanwezel@acm.org, groenen@ese.eur.nl

¹ <http://www.crispyshop.com>

² <http://www.like.com>

³ <http://www.modista.com>

at BrowseGoods.com⁴, where the products are shown in a map ordered as a department store. Both Musiccovery⁵ and LivePlasma⁶ show a map of songs, where the latter also created a similar map for movies. Finally, YouTube⁷ has an option called Warp!, which recommends movies similar to a movie you watched and shows them in a map.

Despite this wide list of commercial map based interfaces, these interfaces lack a, publicly available, scientific foundation. In this chapter, we discuss several issues in product catalog map interfaces which we have published earlier in [21, 22, 23, 24]. This chapter combines these issues and shows applications on a real e-commerce product catalog.

In the next section, we first review some scientific methods that can be used to create such maps that are used in the e-commerce domain or in related fields. Two of these methods, namely multidimensional scaling (MDS) [3] and nonlinear principal components analysis (NL-PCA) [13, 31, 34], are discussed in more detail in Section 17.3, where they are used in two different approaches to create a product catalog map interface. The first which is based on MDS uses a flexible dissimilarity measure between products to create the map. This approach has been published earlier in [21]. The second approach based on NL-PCA and published earlier in [24] has the advantage that it also shows points representing category values (i.e. the possible values of a categorical (nominal) attribute) of attributes that can be used for navigation over the map.

One problem in both map based visualization and content-based recommendation is that different characteristics of products are not considered to be equally important by users. In Section 17.4, we introduce a way to determine attribute weights using clickstream data. We counted how often products were sold. Based on the assumption that attributes that have a high influence on sales of products are attributes that are considered to be important by users, we estimate a Poisson regression model [32, 35] and derive weights from that. This approach has earlier been described in [22].

In Section 17.5, we describe one way in which map based visualization can be combined with a recommender system. The system we propose, the graphical shopping interface, does this by combining MDS based maps with the idea of recommending by proposing [45]. In an iterative process, each time a map containing a limited set of products is shown to the user in which she can select the product she likes most. Then, in the next map, the products that are more similar to this product are shown. This recommender system approach was introduced in [23].

All these methods are applied to a new real commercial product catalog of MP3 players in Section 17.6. Prototypes of these methods are available at <http://www.browsingmap.com/mapvis.html>. This product catalog was provided by Compare Group, owner of the Dutch price comparison site <http://>

⁴ <http://www.browsegoods.com>

⁵ <http://www.musiccovery.com>

⁶ <http://www.liveplasma.com>

⁷ <http://www.youtube.com>

www.vergelijk.nl. Finally, we draw conclusions and give directions for future research in Section 17.7.

17.2 Methods for Map Based Visualization

The use of maps displaying areas with similar items has become increasingly popular in fields where users need to browse through relatively large collections of data, such as, web searching, image browsing, and managing music playlists. Also, map based visualization fits in the trend of visual shopping interfaces introduced in industry. In this section, we discuss four scientific methods for map based visualizations in the fields mentioned above, where we focus on the visualization methods used and their advantages and disadvantages when used to browse through product catalogs. In this discussion, we specifically pay attention to the type of data normally contained in (commercial) product catalogs. These catalogs often contain numerical, categorical, and multi-valued categorical attributes. Also, there can be a lot of missing values, due to, for instance, different product specifications by different manufacturers. Note that, some of the visualization methods could also be used for visualizations in three dimensions. However, we think these 3D visualizations will be too complex for users and, therefore, we only discuss 2D visualizations.

To give somewhat more insight in the differences between the four visualization methods we discuss next, we apply them to a small product catalog. This product catalog, shown in Table 17.1, consists of ten popular MP3 players derived from the larger MP3 player product catalog we use later to show our applications on. Furthermore, we limited the number of attributes to four, which were all chosen to be numerical or binary, such that all four methods could handle them easily. Also, we selected products that did not have any missing values on these four attributes.

Table 17.1: Example data set

Name	Price	HDD	Memory Size (GB)	Weight
1. Apple iPod Nano	180.55	0	8	40.0
2. Apple iPod Video (30GB)	248.41	1	30	130.0
3. Apple iPod Video (60GB)	73.50	1	60	156.0
4. Apple iPod Video (80GB)	324.00	1	80	156.0
5. Apple iPod Shuffle	76.00	0	1	15.5
6. Creative Zen Nano Plus	76.00	0	1	22.0
7. Creative Zen Vision M (30GB)	199.50	1	30	163.0
8. Creative Zen Vision M (60GB)	250.50	1	60	178.0
9. Sandisk Sansa e280	129.99	0	8	75.2
10. Sony NW-A1200	143.90	1	8	109.0

17.2.1 Self-Organizing Maps

In the field of web (search) visualization, Kohonen's Self-Organizing Map (SOM) [27, 28, 29] is among the most popular methods to create maps. Following the early work of Chen et al. [5], SOMs have, for example, been used in applications by Chung et al. [6, 7] and Yang et al. [50]. Ong et al. [36] use a SOM to create a map for online news and Van Gulik et al. [49] for a music collection on an MP3 player.

Self-organizing maps use an unsupervised neural network to cluster and reduce dimensionality at the same time. First, a grid needs to be chosen that represents the structure of the map. Often, this is a rectangular or a hexagonal grid. Informally, the SOM training process works in the following way. All grid points (often called models, weights, or prototype vectors in SOM literature) are initialized with their location on the grid and a vector in the original attribute space.

Then, following an incremental-learning approach items are randomly shown to the SOM. For an item, we first compute which model represents the item best. This model is called the winner or best matching unit (BMU) and is often determined using Euclidean distance. Then, the model and neighbors of the model, to a lesser extent, are adapted to better fit the item at hand. This is done for all items and after that the next iteration is started. During each iteration, a seen item gets less influence on both the model (grid point) and its neighbors, such that the SOM converges to a solution. There also exists a batch-learning algorithm [29] for SOM which is faster.

The main advantage of SOM is that it generally provides nice clustered maps, in which clusters are formed by a set of neighboring grid points to which items are connected and relative many empty grid points. Also, neighboring models are similar to each other providing a similarity interpretation of the map. However, this similarity interpretation is not always valid for the complete map. Although a neighboring model is generally similar to another model, it can be that there exists a model even more similar in another part of the map.

Another disadvantage is that items have the same position in the map, when they are assigned to the same grid point (model), although this can be overcome by specifying a larger grid or making a hierarchical SOM, in which a SOM is created for each model in the original SOM. Also, the original SOM is not able to handle missing values or (multi-valued) categorical attributes, but with an adaptation of the comparison metric used to determine the BMU this is possible [28].

A SOM based on the example product catalog of Table 17.1 and shown in Figure 17.1 was created using the SOM Toolbox for Matlab [1] with a 10×10 rectangular grid. This grid is also shown in the map. As can be seen, each of the products is assigned to one of the squares (grid points). The complete map is used, that is, all corners have a product assigned to them, but there seem not to be real clusters of products. Looking at the ordering of the products on the map, the vertical dimension clearly corresponds to memory size and whether the MP3 player has a hard disk drive (HDD). The horizontal dimension captures the other two attributes, but these effects are not consistent over the map, due to the nonlinear character of SOMs.

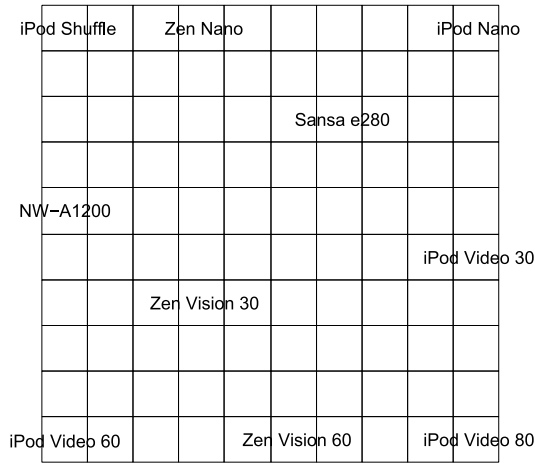


Fig. 17.1: Example of a self-organizing map on the product catalog shown in Table 17.1

17.2.2 Treemaps

Another popular visualization method which was also used to visualize web search results [48], but also to visualize the news in the commercial application NewsMap⁸ of which a screenshot is shown in Chapter 15, is the treemap algorithm [44].

A treemap uses a tree, for example, resulting from a hierarchical clustering as its input. Each leaf node represents a single item. All nodes in the tree are labeled by a weight value. For non-leaf nodes, this weight value is the sum of weights of its children. When all leaf nodes are given a weight of 1, the weight value of all nodes represents the cluster size. Alternatively, items can be labeled by some measure of popularity to make these items more important in the visualization.

Starting at the root node, we divide the map vertically into as many parts as the root has children. The size of the resulting rectangles is relative to the weight values of the nodes. Then, we divide each of the rectangles horizontally using the children of the corresponding node. Alternating vertical and horizontal division, we can continue in this way until the bottom of the tree is reached and each item has a rectangle with an area relative to its weight on the map.

A main advantage of treemaps is that the complete map is filled, there are no spaces without any items and there is no overlap between items in the map. Also, its hierarchical structure is very useful for implementing zooming functionality. Since treemaps can be used in combination with hierarchical clustering algorithms, which are based on a similarity measure, they are also very flexible. A similarity measure can be chosen that is very suitable to the data at hand. For instance, when visualizing

⁸ <http://marumushi.com/apps/newsmap/index.cfm>.

a product catalog, one could use the dissimilarity measure we introduce in the next section, that is able to handle missing values and mixed attribute types.

The treemap approach has two drawbacks. First of all, the original treemap algorithm often produces tall rectangles as results for small clusters or single items, which makes visualization of single products quite hard. This problem can be partly overcome by using squarified treemaps [4] or, even better, quantum treemaps [2], which guarantee a certain aspect ratio for the rectangles representing the items. The second drawback is that, although similar products are clustered together, there is no clear distance interpretation and ordering between clusters might be lost, that is, two quite similar products that are assigned to two different clusters might not be close to each other in the map.

A treemap based on the example product catalog of Table 17.1 is depicted in Figure 17.2. To create this map we first used Matlab’s average linkage hierarchical clustering algorithm based on normalized Euclidean distances and visualized the resulting tree using a treemap algorithm for Matlab written by Hicklin [18]. In the map, wider lines correspond to higher level divisions of the data. Each rectangle, which all have the same area, represents a product. Compared to the SOM, the products that were at the top are on the right in the treemap. These products are the MP3 players without HDD. On the left side, we see now that the Apple iPod Video 60GB is closer to the Apple iPod Video 30GB, the Creative Zen Vision M 30GB, and the Sony NW-A1200 (since the line dividing the Apple iPod Video 80GB and the Creative Zen Vision M 60GB from the Apple iPod Video 60GB is wider than the line under it) than to the Apple iPod Video 80GB and the Creative Zen Vision M 60GB.

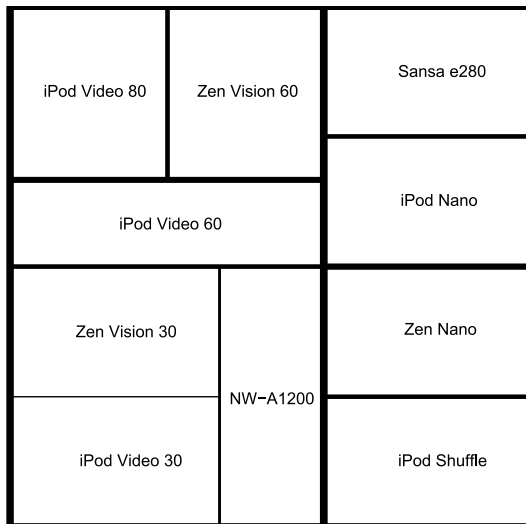


Fig. 17.2: Example of a treemap on the product catalog shown in Table 17.1

17.2.3 Multidimensional Scaling

A third class of applications uses multidimensional scaling (MDS) [3] algorithms to visualize relative large data collections. There exists a wide variety of MDS algorithms, such as classical scaling [47] (e.g. used to visualize music playlists by Donaldson [11]), nonmetric MDS [30], Sammon Mapping [42] (e.g. used to visualize an image collection by Pečenović et al. [37]), and SMACOF [10] (e.g. used to visualize a roller skates catalog by Stappers et al. [46]). However, they are all based on mapping a (symmetric) dissimilarity matrix into low dimensional Euclidean space, such that distances between pairs of points represent the dissimilarities as closely as possible. MDS is one of the methods we use to create product catalog maps and is better described in the next section.

The main advantage of MDS is that the distances in the map really correspond to the similarity between items. Secondly, when using a flexible dissimilarity measure MDS can handle missing values and mixed attribute types. Disadvantages of MDS compared to SOMs and TreeMap are that there may be a lot of empty space in and/or around the map and very similar items may (partially) overlap each other in the map. Empty spaces are a disadvantage, since more zooming is necessary on specific parts of the map to be able to use the map in a satisfying way.

In Figure 17.3, a map based on the example product catalog made using MDS is shown. We created this map using PROXSCAL, available under SPSS Categories [33], which uses the SMACOF algorithm. The dissimilarity matrix used was computed based on normalized Euclidean distances. As was expected, the corners of the map are more empty compared to the SOM. However, the positions of the products on the map (after rotation) do not differ much. Nevertheless, the MDS configuration maps the dissimilarities better, but on the other hand this makes the map less organized. Note that in most of the situations it is impossible to map all dissimilarities perfectly in two dimensions. Therefore, the solution is a compromise in which some dissimilarities are mapped better than others.

17.2.4 Nonlinear Principal Components Analysis

Probably the most well-known method to perform dimension reduction and, thus, for creating maps is principal components analysis (PCA). PCA has a numerical data matrix as input and linearly transforms the data, such that the first dimension, normally plotted in horizontal direction, explains the variance in the data as much as possible. The next dimensions (in case of a map only the second) try to do the same for the remaining variance. Also, all dimensions are uncorrelated with each other. Nonlinear principal components analysis (NL-PCA) [13, 31, 34] is a method that does the same as PCA, but is also able to handle categorical attributes and missing values. Also, using ordinal transformation, numerical attributes can be transformed nonlinearly. Additionally, the categories of the attributes also have a location in the map, that is in the center of the items belonging to that category. These category

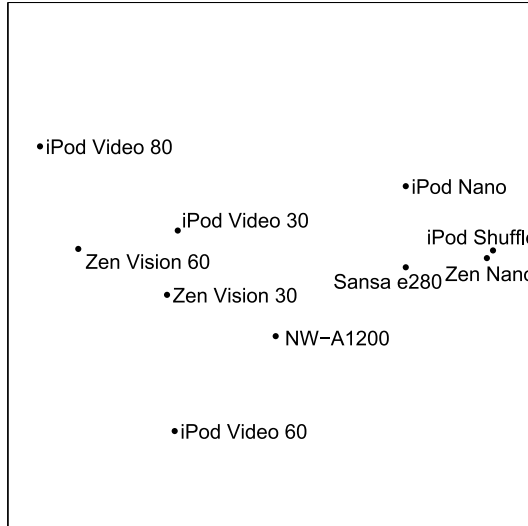


Fig. 17.3: Example of a map made with multidimensional scaling on the product catalog shown in Table 17.1

points can be used to give an interpretation to the map, but also as a navigation tool in the map based interface as is done in our product catalog map using NL-PCA, which is introduced in the next section.

Figure 17.4 shows the NL-PCA configuration of the example product catalog which was made using CATPCA also available under SPSS Categories [33]. All numerical attributes were treated as ordinal variables, such that nonlinear transformation was possible. However, due to the large flexibility of NL-PCA and the small data set, the resulting map has a lot of overlapping products. In fact, NL-PCA created clearly three clusters. Although the products in the clusters were also close together in the maps of the other three methods, they did not really form clusters. This example shows, in extreme, the ability of NL-PCA to cluster products having the same category value together.

17.3 Product Catalog Maps

In this section, we introduce two ways to create a product catalog map. We start by describing a product catalog map based on multidimensional scaling (MDS), which is combined with a k -means clustering algorithm to highlight some prototypical products. Thereafter, a product catalog map based on nonlinear principal components analysis (NL-PCA) is introduced which uses the category points to provide a navigation mechanism. These two methodologies were published earlier in [21] and [24].

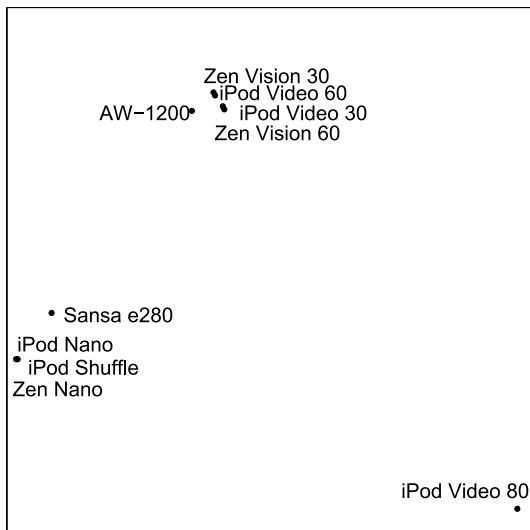


Fig. 17.4: Example of a map made with nonlinear principal components analysis on the product catalog shown in Table 17.1

In the description of these methods (and in the remainder of this paper), we use the following mathematical notation. A product catalog D contains I products \mathbf{x}_i having K attributes $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iK})$. The final two-dimensional map can be described by an $I \times 2$ matrix \mathbf{Z} containing the coordinates of the products in the map.

17.3.1 Multidimensional Scaling

The first approach to create a product catalog map that we describe is based on MDS. As was mentioned in Section 17.2.3, the basis of an MDS map is a dissimilarity matrix. To compute a dissimilarity matrix $\mathbf{\Delta}$ from the product catalog, we need a dissimilarity measure. This measure should be able to cope with the specific data contained in a product catalog, that is, it should be able to handle missing values and numerical, categorical, and multi-valued categorical attributes.

Many popular (dis)similarity measures such as the Euclidean distance, Pearson's correlation coefficient, and the Jaccard similarity measure are not able to handle all of these attribute types. Moreover, they can not handle missing values naturally. Therefore, we use a dissimilarity measure which is an adaptation of the general coefficient of similarity proposed by Gower [14] and was introduced in [23]. Note that the MDS based product catalog map approach can also be used with other dissimilarity measures, such as co-purchases or item-item correlations.

The dissimilarity δ_{ij} between products i and j is defined as the square root of the average of nonmissing dissimilarity scores δ_{ijk} on the K attributes

$$\delta_{ij} = \sqrt{\frac{\sum_{k=1}^K w_k m_{ik} m_{jk} \delta_{ijk}}{\sum_{k=1}^K w_k m_{ik} m_{jk}}}, \tag{17.1}$$

where w_k is the weight of attribute k and m_{ik} and m_{jk} are binary indicators having a value of 0 when attribute k is missing for product i or j respectively and 1 otherwise. The weights w_k can be used to make some attributes more important than others. In Section 17.4, we show how these weights could be assigned automatically to match users’ general preferences. The definition of the dissimilarity score δ_{ijk} depends on the type of attribute k . For all attribute types we use the same kind of normalization that ensures that the average nonmissing dissimilarity score for each attribute is equal to 1 in the product catalog. This normalization is used, to make the dissimilarity scores equally important and independent of the number of missing values.

The numerical dissimilarity score is based on the absolute distance

$$\delta_{ijk}^N = \frac{|x_{ik} - x_{jk}|}{(\sum_{i < j} m_{ik} m_{jk})^{-1} \sum_{i < j} m_{ik} m_{jk} |x_{ik} - x_{jk}|}. \tag{17.2}$$

The dissimilarity score for categorical attributes is computed as

$$\delta_{ijk}^C = \frac{1(x_{ik} \neq x_{jk})}{(\sum_{i < j} m_{ik} m_{jk})^{-1} \sum_{i < j} m_{ik} m_{jk} 1(x_{ik} \neq x_{jk})}, \tag{17.3}$$

where $1()$ is the indicator function returning a value of 1 when the condition is true and 0 otherwise.

To handle categorical attributes for which a product can belong to multiple categories (multi-valued categorical attributes), this dissimilarity framework was extended in [22]. There, the dissimilarity score for multi-valued categorical attributes was defined as

$$\delta_{ijk}^M = \frac{|x_{ik} \Delta x_{jk}|}{(\sum_{i < j} m_{ik} m_{jk})^{-1} \sum_{i < j} m_{ik} m_{jk} (|x_{ik} \Delta x_{jk}|)}, \tag{17.4}$$

where both x_{ik} and x_{jk} are sets of values and Δ is the symmetric difference set operator. This measure counts how many categories there are to which one of the products at hand belongs and the other not.

As mentioned in Section 17.2.3, the aim of MDS is to map a dissimilarity matrix Δ (having elements δ_{ij} and in our approach computed using (17.1)) as good as possible to distances between points in a low dimensional Euclidean space. This objective can be formalized by minimizing the raw Stress function [30]

$$\sigma_r(\mathbf{Z}) = \sum_{i < j} w_{ij} (\delta_{ij} - d_{ij}(\mathbf{Z}))^2. \tag{17.5}$$

In this equation, \mathbf{Z} is an $I \times 2$ coordinate matrix which forms the basis for the map, δ_{ij} is the dissimilarity between items i and j and $d_{ij}(\mathbf{Z})$ is the Euclidean distance between the coordinates of i and j , that is

$$d_{ij}(\mathbf{Z}) = \sqrt{\sum_{s=1}^2 (z_{is} - z_{js})^2} . \quad (17.6)$$

Also, weights w_{ij} can be specified to force some dissimilarities to be fit better than others.

The dissimilarity measure we use is able to handle missing values. However, dissimilarities based on only a couple of nonmissing (and maybe even unimportant) attributes are more unreliable than dissimilarities for which no dissimilarity scores were missing. Therefore, the latter should receive higher weights. This can be done by defining the weights to be used in (17.5) as the weighted proportion of nonmissing attributes used for pair ij

$$w_{ij} = \frac{\sum_{k=1}^K w_k m_{ik} m_{jk}}{\sum_{k=1}^K w_k} . \quad (17.7)$$

We minimize $\sigma_r(\mathbf{Z})$ using the SMACOF algorithm [10] which is based on majorization. One of the advantages of this method is that it is reasonable fast and that the iterations yield monotonically improved Stress values and the difference between subsequent coordinate matrices \mathbf{Z} converges to zero [10]. This property has an important and vital consequence for dynamic visualizations: the algorithm produces smooth changes to the points in the display leading to a (local) minimum solution of (17.5). In effect, the objects follow a smooth trajectory on the screen.

The resulting maps may look overwhelming to the user when the number of products is large. To make the map more appealing to the user, a small number of products is highlighted by showing larger sized and full color images, while other products are represented by a smaller monochrome image. These highlighted products are helpful to the user to get a quick overview of the map. Therefore, it is nice when these products represent different groups of products in this map. This was done, by first clustering the products in the map using k -means clustering [17]. We decided to perform a k -means clustering on the map \mathbf{Z} instead of a hierarchical clustering procedure on the original dissimilarities for two reasons. First, this procedure is faster and, second, it is consistent with the visualization, that is, there is no overlap between the clusters in the map. In each cluster, one product is chosen to be highlighted, that is, the product closest to the cluster center based on Euclidean distance. In Section 17.6, we show a prototype of this approach using a product catalog of MP3-players.

17.3.2 Nonlinear Principal Components Analysis

The second approach for creating a product catalog map discussed here is based on NL-PCA. In this approach, a map is created in which not only the products are plotted, but also the category values of the attributes. These can then be used for navigation and selection. NL-PCA is a generalization of ordinary principal components analysis to ordinal (nonlinearly ordered) and categorical attributes. When only having numerical attributes, NL-PCA simplifies to ordinary PCA and when all attributes are categorical and a so-called multiple nominal transformation is chosen, then NL-PCA is identical to homogeneity or multiple correspondence analysis.

In homogeneity analysis, the $I \times K$ data matrix \mathbf{X} is modeled by an indicator matrix \mathbf{G}_k for every attribute. Let L_k denote the number of categories of attribute k . Every category $\ell, \ell = 1, \dots, L_k$ has its own column in the $I \times L_k$ matrix \mathbf{G}_k , in which a 1 denotes that the object belongs to this category and a 0 that it does not. Multi-valued categorical attributes are modeled using an $I \times 2$ indicator matrix for every category. Missing values are incorporated using an $I \times I$ binary diagonal matrix \mathbf{M}_k for all attributes having a value of 1 on the diagonal for nonmissing values for attribute k and 0 otherwise. Using this data representation, we can define the following loss function for homogeneity analysis [13, 34] by

$$\sigma(\mathbf{Z}; \mathbf{Y}_1, \dots, \mathbf{Y}_K) = K^{-1} \sum_{k=1}^K \text{tr}(\mathbf{Z} - \mathbf{G}_k \mathbf{Y}_k)' \mathbf{M}_k (\mathbf{Z} - \mathbf{G}_k \mathbf{Y}_k) , \quad (17.8)$$

where \mathbf{Z} is a $I \times 2$ matrix representing the objects in 2D Euclidean space and the matrix \mathbf{Y}_k is the 2 dimensional representations of the category values of the attribute k . Both \mathbf{Z} , the coordinates of objects, and all \mathbf{Y}_k 's, the coordinates of the attribute category values, can be plotted in a joint space which is called a biplot [15]. Essentially, $\mathbf{Z} - \mathbf{G}_k \mathbf{Y}_k$ gives the differences (or error) between the position of the individual products and the positions of the category centroids they belong to for variable k . Ideally, no error would exist and all products in the same category would be fully homogeneous and coincide with the position of their category. As there are more attributes and the products fill in different categories on the different attributes, (17.8) simply measures the squared distances of the products relative to their category centroids, hence how homogeneous the products are. The matrix \mathbf{M}_k removes the contribution to the error for any product having a missing value for attribute k . Equation (17.8) can be minimized using an alternating least squares (ALS) procedure called Homals [13, 34].

From Homals, it is an easy change to NL-PCA by imposing extra restrictions on the category points. Numerical and ordinal attributes can be incorporated in the Homals framework when we impose a rank-1 restriction of the form

$$\mathbf{Y}_k = \mathbf{q}_k \mathbf{a}_k' \quad (17.9)$$

for the numerical and ordinal attributes, where \mathbf{q}_k is a L_k dimensional column vector of category quantifications and \mathbf{a}_k is a 2 dimensional column vector of weights.

Using this restriction the category points are forced to be on a line. However, this restriction is not sufficient to preserve the order for ordinal attributes or even the relative distance for numerical attributes. Therefore, \mathbf{q}_k is constrained every ALS iteration to satisfy such restrictions. In the case of ordinal attributes this transformation is done by a weighted monotone regression [9] and in the case of numerical attributes this results in replacing \mathbf{q}_k by the attribute's original values \mathbf{x}_k . A detailed description of the ALS algorithm for NL-PCA can be found in [13, 31, 34].

NL-PCA solutions have a number of advantageous properties that make it very suitable to create maps that contain both products and attribute category values.

- NL-PCA provides a joint space of object and attribute category points, called a biplot [15], while other visualization methods only provide the object points. The category points can be used to provide an easier interpretation of the map and to navigate through the map by selecting subsets of products.
- For categorical attributes, the category points are located in the centroids of the object points belonging to that category. This implies that when a certain attribute is well represented in the map, the object points are clustered around their corresponding category value of that attribute and a selection of all points belonging to this category will lead to a selection of a subspace of the map. For ordinal attributes, the category point will be the point on the line closest to the centroid of the object points.
- The third advantage is shared by NL-PCA and most other visualization methods. That is, the distance between object points in the map is, in general, related to dissimilarity between objects.

In Section 17.6, we also show an application of this approach to the MP3 player data set.

17.4 Determining Attribute Weights using Clickstream Analysis

Both product catalog map approaches introduced in the previous section consider all attributes of a product as equally important to the user. However, we showed that attribute weights can be incorporated in the dissimilarity measure used for MDS and also NL-PCA can be adapted to incorporate different weights for attributes. This holds for most visualization methods, including self-organizing maps and treemaps which were described in Section 17.2.

In this section, we will introduce an approach to determine these weights automatically using clickstream data. For every product, we count how often it was sold during some period. In our application, shown in Section 17.6, we actually counted outclicks, which are clicks out of the site (we used data of a price comparison site) to a shop where the product can be bought. For ease of readability, we use the term sales instead of outclicks during the remainder of this paper. Using these counts and the product attributes, we estimate a Poisson regression model, which is a model belonging to the class of generalized linear models. Using the coefficients of this

model and their corresponding standard errors, we compute t -values which form the basis of our attribute weights. This approach was described earlier in [22].

17.4.1 Poisson Regression Model

A collection of models frequently used in the field of statistics are the generalized linear models (GLM) [32, 35]. To model a dependent count variable being discrete and nonnegative, such as sales in our domain, we use an appropriate member of the GLM family, that is, the Poisson regression model. In Poisson regression, we cannot use (multi-valued) categorical attributes directly, so we have to create dummy attributes instead. Therefore, every categorical attribute is represented by $L_k - 1$ dummies $x_{ik\ell}$, which are 1 for the category where the item belongs to and 0 for all other attributes, where L_k is the number of different categories for attribute k . When an item belongs to the last category L_k all dummies for this attribute will be 0. This representation is chosen to avoid multicollinearity. For multi-valued categorical attributes the same approach is used, only now all categories are represented by, in total, L_k dummies. For numerical attributes, we can just use the attribute itself. Hence, $x_{ik} = x_{ik1}$ and $L_k = 1$. We collect all $x_{ik\ell}$ for item i in vector \mathbf{x}_i . Also, an intercept term x_{i0} is incorporated in this vector, which equals 1 for all items. Hence, $\mathbf{x}_i = (x_{i0}, x_{i11}, \dots, x_{iKL_K})$. Furthermore, we have the dependent count variable value y_i for all I items. Now, we can express the Poisson regression model as

$$y_i \approx \exp(\mathbf{x}_i' \mathbf{b}) \quad , \quad (17.10)$$

where \mathbf{b} is a vector of regression parameters to be estimated. Furthermore, it is assumed that y_i is Poisson distributed having expectation $E(\exp(\mathbf{x}_i' \mathbf{b}))$. The Poisson regression model can be fitted by maximizing its loglikelihood function, which is often done by an iteratively reweighted least squares algorithm. Besides the model parameters $b_{k\ell}$, also standard errors σ_{kl} can be derived by this algorithm.

17.4.2 Handling Missing Values

Unfortunately, the Poisson regression model cannot cope with missing values in an integrated way. Missings are in general a problem in GLMs and Imbrahim et al. [20] recently compared different techniques that can be used to handle missing values in combination with GLMs. One of the best methods (leading to unbiased estimates and reliable standard errors) in their comparison was multiple imputation (MI) [41]. MI methods create a number of ‘complete’ data sets in which values for originally missing values are drawn from a distribution conditionally on the non-missing values. These imputed data sets can be created using two different methods: data augmentation [43] and sampling importance/resampling [26]. Although

both methods lead to results having the same quality, the second method computes these results much faster. Therefore, an algorithm based on the second approach, namely the Amelia algorithm [26] which is available as a package [19] for the statistical software environment R, is used in our approach. For a discussion about how the regression coefficients and standard errors of these imputed datasets can be used to estimate the parameters and standard errors of the Poisson regression model, we refer to [22].

17.4.3 Choosing Weights Using Poisson Regression

There are three reasons why we cannot use the coefficients $b_{k\ell}$ resulting from the Poisson regression model as weights directly. The first reason is that dissimilarity scores and attributes do not have the same scale. Second, uncertainty about the correctness of the coefficient is not taken into account when using $b_{k\ell}$ directly. Although the value of a coefficient can be relatively high, it can still be unimportant. Consider, for example, a dummy attribute having very few 1's and a high coefficient value. Then, this high impact of the coefficient is only applicable to a limited number of items and its total importance is limited. By taking the uncertainty we have into account, we can correct for this. Third, weights should always be equal to or larger than 0, while $b_{k\ell}$ can also be negative.

The first two problems can be overcome by using the t -value

$$t_{k\ell} = \frac{b_{k\ell}}{\sigma_{k\ell}} \quad (17.11)$$

of coefficient $b_{k\ell}$ as a basis to compute weights. Due to standardization, all $t_{k\ell}$'s are on the same scale and, since these are standardized by division by the corresponding standard error $\sigma_{k\ell}$, uncertainty is taken into account. When we use $|t_{k\ell}|$ instead of $t_{k\ell}$ we guarantee the weights to be larger than or equal to 0.

For numerical attributes, we can map $|t_{k1}|$ directly to a 'pseudo' absolute t -value v_k for attribute k , that is, $v_k = |t_{k1}|$. Then, including a normalization of the weights (for ease of interpretability), we can compute w_k using

$$w_k = \frac{v_k}{\sum_{k'=1}^K v_{k'}} \quad (17.12)$$

For (multi-valued) categorical attributes, we first have to compute v_k using the L_k values of $t_{k\ell}$. This is done by taking the average of the absolute $t_{k\ell}$ values

$$v_k = \frac{1}{L_k} \sum_{\ell=1}^{L_k} |t_{k\ell}| \quad (17.13)$$

These v_k 's can then be used to compute the weights for (multi-valued) categorical attributes using (17.12).

17.4.4 Stepwise Poisson Regression Model

The t -values $t_{k\ell}$ can be compared to a t -distribution to determine whether these values are significantly different from 0. In a so-called stepwise model, this significance testing is used for attribute selection, finally resulting in a model only having significant attributes. The stepwise model approach starts off with a model containing all attributes. Then, each time the most insignificant attribute is deleted from the model, until the model only contains significant attributes. Note that this stepwise approach leads to a different model when all insignificant attributes are deleted at once. In fact, due to collinearity, significance of an attribute may change when another attribute is deleted. When using the stepwise model to determine weights w_k , we consider L_k to be the number of dummies incorporated in the final model. This stepwise model is used in the application shown in Section 17.6.

17.5 Graphical Shopping Interface

The idea of map based visualization can be combined with recommender systems [39]. The most direct way for doing this is by just representing recommendations of the system in a map instead of a list. This approach is taken in the graphical recommender system introduced in [23]. Another system introduced in that paper, the graphical shopping interface (GSI), implements a more interactive recommendation process for users who do not have a clear idea about what they are looking for and need to shape their preferences. The GSI implements an approach which in recommender system literature is called recommendation by proposing [45] or inspiration seeking [40]. The idea is to let the user navigate through the complete product catalog in steps, where at each step a set of products is represented in a map. In this map, the user can select a product and then a new set of products, generally more similar to the selected product, is produced and visualized by MDS in a similar way as is done by the MDS based product catalog map introduced in Section 17.3.1.

In [23], three different types of the GSI were proposed, the random system, the clustering system, and the hierarchical system. Since the random system performed best in a simulation study in that paper, we only consider the random system here.

In this system, each time a small set of products is randomly selected to be shown to the user out of a larger set. This larger set contains products that are similar to a product selected by the user. First, the GSI needs to be initialized. We refer to this initialization as iteration $t = 0$. In this initialization, the larger set of products D_t is set to be the complete product catalog, that is, $D_0 = D$. Out of set D_0 , p products are selected at random (without replacement). These products form together the smaller set D_0^* . Then, dissimilarity matrix Δ_0^* is computed using the adapted Gower coefficient introduced in Section 17.3.1 and D_0^* . Finally, a map \mathbf{Z}_0 in which these randomly selected products are mapped is created using MDS and shown to the user.

The iterative process starts when the user selects one of the shown products we denote by \mathbf{x}_i^* . Then, the dissimilarities between \mathbf{x}_i^* and all other products in D are

computed. To create D_t , we select the $\max(p - 1, \alpha^t I - 1)$ products that are most similar to \mathbf{x}_t^* . The parameter α , where $0 < \alpha \leq 1$, determines how much the size of D_t is decreased each iteration. Thereafter, the process is almost identical to the steps taken in the initialization. We create a small set D_t^* consisting of \mathbf{x}_t^* and $p - 1$ products randomly selected from D_t and compute a dissimilarity matrix $\mathbf{\Delta}_t^*$ based on these products. This matrix is the input for the MDS algorithm returning the new map \mathbf{Z}_t .

The parameter α determines how large the influence of the selections of the user are on the complete process. When $\alpha = 1$, this influence is very small, since each time a completely random selection is shown to the user except for the product selected by the user in the last iteration. When α is lower, this influence is higher, but the variance in D_t also decreases more quickly. The random system is summarized in Figure 17.5.

```

procedure RANDOMgSI( $D, p, \alpha$ )
   $D_0 = D$ .
  Generate random  $D_0^* \subset D_0$  with size  $p$ .
  Compute  $\mathbf{\Delta}_0^*$  given  $D_0^*$  using (17.1).
  Compute  $\mathbf{Z}_0$  given  $\mathbf{\Delta}_0^*$  using MDS.
   $t = 0$ .
  repeat
     $t = t + 1$ .
    Select a product  $\mathbf{x}_t^* \in D_{t-1}$ .
    Get  $D_t \subset D$  containing  $\max(p - 1, \alpha^t I - 1)$  products most similar to  $\mathbf{x}_t^*$  using (17.1).
    Generate random  $D_t^* \subset D_t$  with size  $p - 1$ .
     $D_t^* = D_t^* \cup \mathbf{x}_t^*$ .
    Compute  $\mathbf{\Delta}_t^*$  given  $D_t^*$  using (17.1).
    Compute  $\mathbf{Z}_t$  given  $\mathbf{\Delta}_t^*$  using MDS.
  until  $D_t^* = D_{t-1}^*$ .
end procedure

```

Fig. 17.5: Pseudocode of the graphical shopping interface.

In Section 17.6, we also show an application of the GSI.

17.6 E-Commerce Applications

In this section, we describe three working prototypes that are available online. Two of them implement the product catalog maps based on MDS and NL-PCA and the third prototype is a graphical shopping interface (GSI). Both the MDS based product catalog map and the GSI also have an option to use weights determined by the method described in Section 17.4. All prototypes are using a product catalog of MP3 players that is described next.

The product catalog of MP3 players was made available to us by Compare Group. Compare Group hosts, among other European price comparison sites, the Dutch

price comparison site <http://www.vergelijk.nl>. The product catalog used is based on a database dump of this site from October 2007. At that moment, there were 225 MP3 players listed on the site. In total, these products were described using 45 attributes of which there were 16 numerical, 20 categorical, and 45 multi-valued categorical. Of these 45 attributes, 26 attributes were missing for more than half of the MP3 players. The other attributes are listed in Table 17.2. Note that although it also concerns MP3 players, it is different from that used in [21, 23, 24]. To be able to determine the weights automatically as described in Section 17.4, we matched the product catalog to a clickstream log of the same website logged during a period of two months from July 15 until September 15, 2007.

Table 17.2: Attribute characteristics of the MP3 player data. Only the attributes having less than 50% missing values are listed. For (multi-valued) categorical attributes only the three most occurring values are shown.

Attribute	% Missing Values	Mean
<i>Numerical Attributes</i>		
Price	0.0%	134.54
Height	40.9%	63.65
Width	40.9%	48.87
Weight	44.0%	71.21
Depth	40.9%	16.54
Memory Size	3.1%	8635.30
Battery Life	44.4%	17.26
<i>Categorical Attributes</i>		
Brand	0.0% Samsung (12.0%), Creative (9.8%), Philips (8.4%)	
Radio	32.0% yes (68.6%), no (31.4%)	
Extendable Memory	44.9% yes (17.7%), no (82.3%)	
Equalizer	39.6% yes (85.3%), no (14.7%)	
Screen	30.2% yes (99.4%), no (0.6%)	
Battery Type	44.4% li-ion (44.8%), 1×AAA (33.6%), li-polymer (20.8%)	
Voice Memo	24.4% yes (81.2%), no (18.8%)	
<i>Multi-Valued Categorical Attributes</i>		
Storage Type	42.7% flash memory (69.0%), hdd (21.7%), sd card (10.1%)	
Interface	4.0% usb (63.4%), usb 2.0 (31.5%), hi-speed usb (7.4%)	
Color	38.2% black (68.3%), white (20.1%), silver (16.5%)	
Operating System	31.1% windows (78.7%), mac os (34.2%), windows xp (32.3%)	
Audio Formats	1.8% MP3 (98.6%), WMA (90.0%), WAV (48.0%)	

17.6.1 MDS Based Product Catalog Map Using Attribute Weights

The MDS based product catalog map prototype can be visited online at <http://www.browsingmap.com/mapvis.html>, where it is available as a Java applet. A screenshot of this prototype is shown in Figure 17.6. The GUI of the pro-

tototype mainly consists of a large focus map which gives a detailed view of a part of the complete product catalog map. In this map, a couple of products, in this case 12, are represented by a larger full color image, the other products are visualized as smaller monochrome images. By default, the monochrome images are colored according to the cluster they belong to. Alternatively, the user has the option to color them by product popularity (based on the clickstream data) or by attribute value. The small overview map at the top always shows the complete product catalog map. The user can navigate over the map in several ways. By selecting a rectangular sub-space in the focus or overview map, the user can zoom in on a specific part of the map. Zooming is also possible using the mouse wheel. Finally, the user can move over the map using dragging. Since the map itself is static, that is, the coordinates of products are fixed, the computation of the map can be done offline, decreasing online computation time. Of course, it is straightforward to add traditional search technology, such as performing crisp queries, to the map based interfaces. All remaining products satisfying the constraints can then be displayed using a map. This would require the map to adapt when the product set changes and in this case client computations may be necessary.

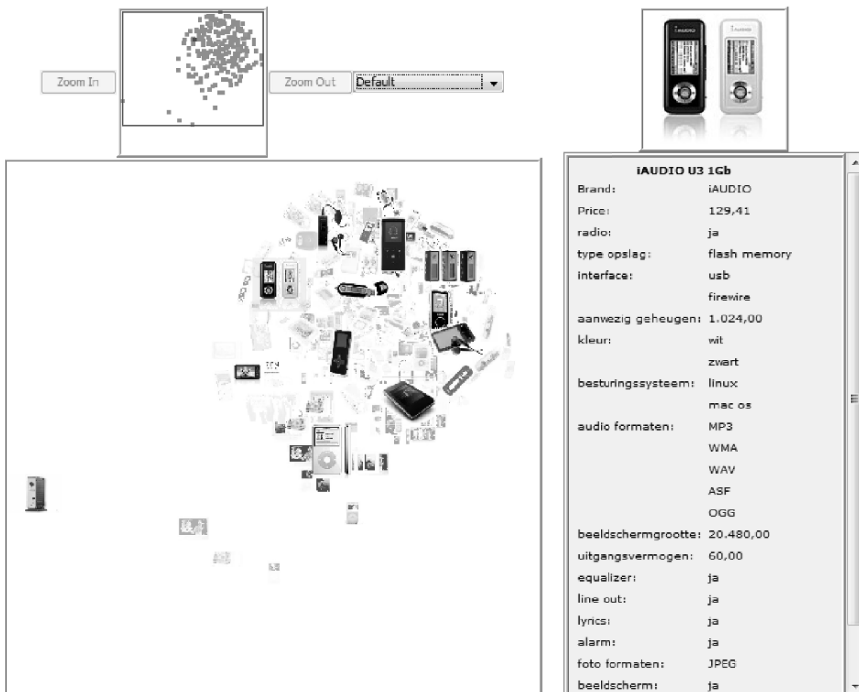


Fig. 17.6: Screenshot of the MDS based product catalog map prototype.

The map of the prototype has been made using the attribute weight determination technique described in Section 17.4. These weights were determined using the data and product catalog described above. To be able to impute the missing values, we excluded attributes having more than 50% missing values and categories of (multi-valued) categorical attributes that were observed less than 10 times. The weights determined in this way are shown in Table 17.3. Note that attributes not mentioned in the table have a weight of zero and, hence, have no influence in the MDS procedure. According to our method, Brand and Memory Size are the most important attributes determining popularity of MP3 players and receive the highest weights.

Table 17.3: Weights determined using stepwise Poisson regression for the MP3 player catalog. Only selected attributes are shown.

Attribute	Weight
Brand	0.242
Memory Size	0.176
Audio Formats	0.131
Battery Type	0.106
Width	0.090
Operating System	0.086
Color	0.084
Storage Type	0.084

In Figure 17.7, we labeled the product points of both a map made by MDS using equal weights for all attributes and a map made using our weighting approach by their corresponding Brand. The second map was rotated using Procrustean transformation [16] to best match the first map. As can be seen in Figure 17.7, the products having the same brand are more clustered in the second map where brand had been made more important.

Figure 17.8a shows the second most important attribute, i.e., Memory Size. This map shows that the MP3 players are generally ordered from cheap at top left to expensive at the bottom right in the map. The most striking attribute absent in Table 17.3 and, thus, receiving a weight of zero, was Price. However, when we look at Figure 17.8b where we have made a map labeled by Price, there is a clear pattern in the map. Both effects exist, since Price highly correlates with other features of MP3 players, such as the Memory Size. These features are, hence, the features explaining most of the price that is given to a product.

More generally, we can see that the MDS produces a kind of circular shaped map with some outliers that represent products very different from all other ones. Due to this, large parts of the map are unused. On the other hand, the map provides at first sight a good interpretation, where the important attributes show a clear pattern.

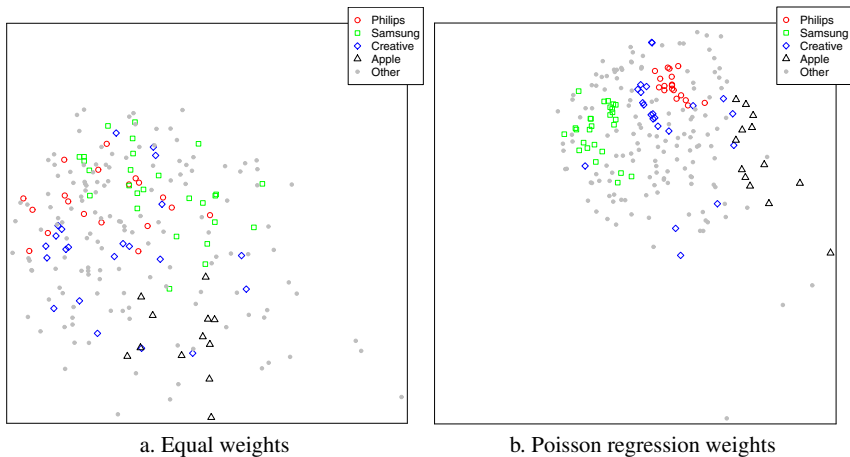


Fig. 17.7: Product catalog maps based on MDS labeled by brand.

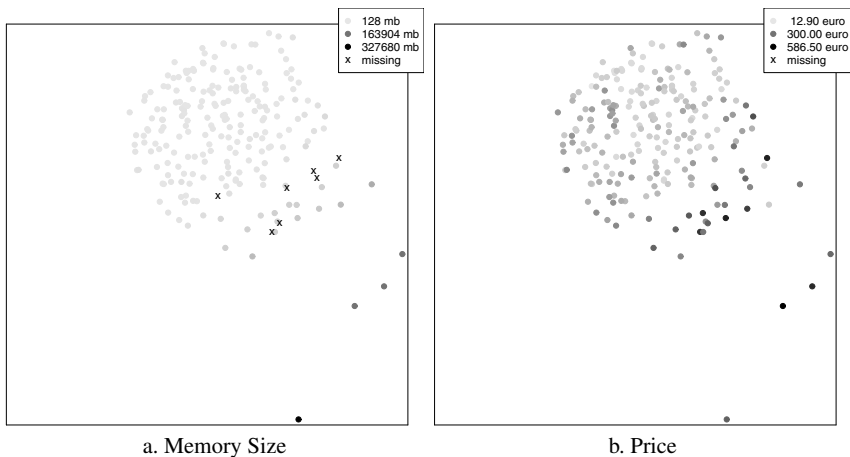


Fig. 17.8: Product catalog maps based on MDS labeled by Memory Size and Price.

17.6.2 NL-PCA Based Product Catalog Map

A screenshot of the prototype using NL-PCA is shown in Figure 17.9. This application is available online at <http://www.browsingmap.com/mapvis.html>. The main part of the GUI is the product catalog map. Initially, this map shows all products. A couple of products, in this case 12, are highlighted using a larger full color image, the other products are represented by a smaller monochrome image. For the initial map, the products that are highlighted are selected using a k -means clustering procedure identical to the one used in the MDS based product catalog map.

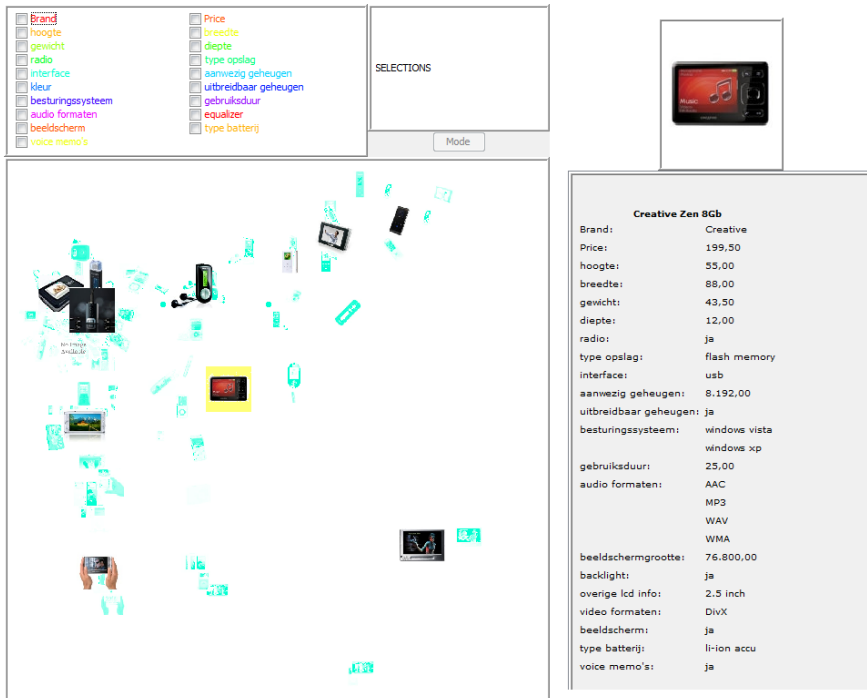


Fig. 17.9: Screenshot of the NL-PCA based product catalog map prototype.

Above the product catalog map, there is a list with all product attributes. Each attribute can be selected by clicking on its check box. If one selects a certain attribute, the category points of this attribute are added to the map. The category points of the selected attributes not only help the user to interpret the map, they are also tools to navigate through the map. By clicking on a category point on the map this category is added to the selection list.

The selection list, which is shown to the right of the list of attributes, determines the products that are highlighted on the map. The set of highlighted products is

determined as follows. For each of the selected attributes, a shown product should belong to at least one of the selected categories.

When the selection list has been adapted by adding or removing a category point, a couple of things are modified in the visualization of the map. The first thing is that all products satisfying the new constraints are colored red, while all other products are colored blue. Furthermore, a number of products is randomly selected from this set to be highlighted.

Since a selection will often lead to a subspace of the map, it is also possible to zoom in on this part of the map. However, there is no guarantee that all points in this subspace satisfy all constraints imposed by the selection list. We have chosen not to delete these product points, since these may be interesting to the user. Although these products do not satisfy all the demands of the user, they are very similar to the products that do and may have some appealing characteristics the user until then did not think of.

In both the complete and the zoomed in map, the user can click on the highlighted products to get a full product description of this selected product, which is shown at the right side of the application. However, by moving over both a full color or a monochrome image, a tooltip is shown containing an image of the product and the values of some of its most important attributes. Furthermore, the values for the attributes in the selection list are shown in this tooltip, colored green when they match the preferred category value and red when they do not. Since the GUI is based on a single NL-PCA map, this map too can be computed offline just as the MDS product catalog map.

Since the quality of NL-PCA maps may become quite poor when having a lot of missing values, we removed attributes having more than 50% missing values. Then, we also removed products having more than 50% missing values on this limited set of attributes. This resulted in a set of 189 MP3-players described by 19 attributes, namely the attributes shown in Table 17.2.

In the NL-PCA algorithm, we will treat all numerical attributes as being ordinal, because of two reasons. In the first place, many of the numerical attributes do not have a linear interpretation for users, such as, for example, the memory size. The second advantage of using the ordinal transformation is that due to the underlying monotone regression procedure some adjacent categories can be merged into a single category point. Since a numerical attribute has a lot of categories (i.e. all unique values in the data set), visualizing all these categories may become unclear and selection using these category values may become useless, since a lot of category values only belong to a single object. Using an ordinal transformation this becomes less of a problem, since categories with a small number of objects are often merged with their neighbors.

In Figure 17.10, two biplots are shown resulting from the NL-PCA algorithm. A biplot visualizes both the products labeled by their attribute value and the category points also labeled by their value. Also, the origin is plotted in the biplots. By the design of the NL-PCA method, ordinary products (products having attribute values that are similar to many other products) should be close to the origin, while more distinct products should be far away. This also holds for the categories. Since both

biplots in Figure 17.10 are plots of numerical attributes, the category points are on a line. Like in the MDS map, also here both Price and Memory Size correlate with each other and are well represented, in this case on the second dimension.

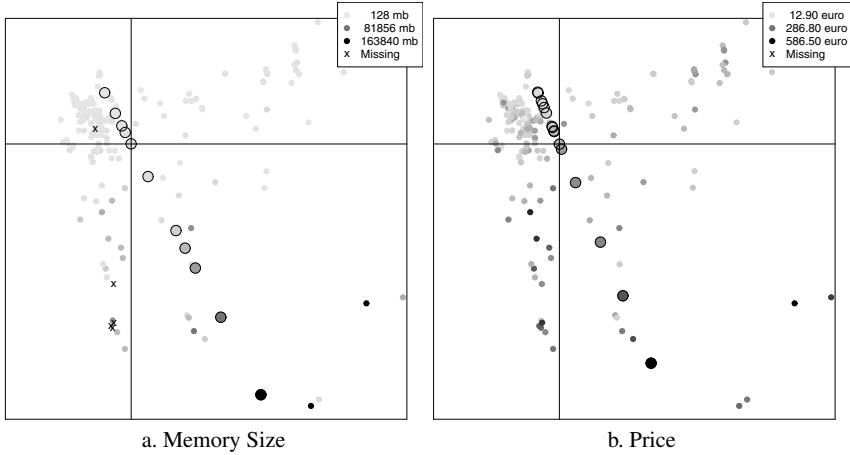


Fig. 17.10: Biplots based on NL-PCA labeled by Memory Size and Price. Large circles are category points and small dots represent object points.

Where the second dimension tells something about the memory size and attributes correlating with that, the first dimension seems to separate older from newer products. This can, for instance, be seen in Figure 17.11, where we labeled products by the Interface attributes. Since Interface is a multi-valued categorical attribute, we have chosen to label each product only by a single value, namely the one most frequent in the product catalog. Also, we only show the category points of the three most occurring categories, since all products belong to at least one of them. As said, there seem to be two groups, the older MP3 players supporting USB and the newer ones supporting USB 2.0. For the operating systems attribute one can observe a similar pattern.

More generally, the NL-PCA approach creates a map in which more of the available space is used than in the MDS approach. However, most of the map is used to visualize the special, not that common products, while the ordinary products are cluttered together near the right top corner of the map. Also, the map only shows those products and attributes that do not have too many missing values.

17.6.3 Graphical Shopping Interface

A screenshot of the graphical shopping interface prototype is shown in Figure 17.12. This prototype is available at <http://www.browsingmap.com/mapvis>.

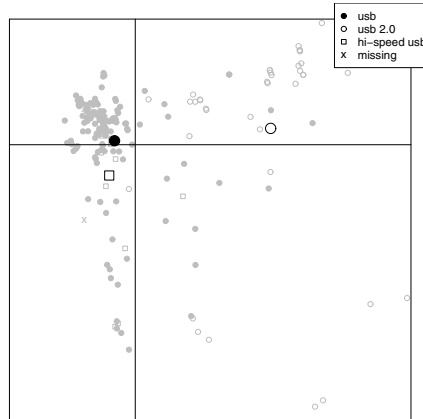


Fig. 17.11: Biplot based on NL-PCA labeled by Interface. Large shapes indicate category points and small transparent shapes represent object points.

html. The interface uses four tabs: The Navigate tab implementing the graphical shopping interface (GSI), the Direct Search tab implementing the graphical recommender system (GRS) [23], a Preferences tab to set weights, and a Saved Products tab to save products in.

The graphical recommender system creates a map of products given a search query. The represented products are those most similar to the query. Representing these results in a map instead of, as is usually done, a list has the advantage that it makes clear which search results are similar to each other and which differ from the query on a different set of attributes.

In both the GSI and GRS map, products are represented by a thumbnail picture. By clicking, products can be selected and detailed information is shown right from the map. Above the map in the GSI tab, there are some buttons to go to the next iteration, to go one iteration back, to restart the process, and to save a product.

In the Preferences tab the user can set the weights used to compute dissimilarity in the GRS and GSI. Also, the user can deselect certain attributes, which means that their weights are set to 0. When weights are changed, the maps are immediately adapted. This tab also has a ‘smart weights’ button which can be used to set the weights to the values determined using the approach discussed in Section 17.4. Hence, the weights are then set to the the values shown in Table 17.3.

There is a smooth transition between two steps in the GSI. After the selection of a product by the user, the new products are added to the map at random positions. Then, the map is optimized using MDS. The optimization process is shown to the user. When the optimization has converged, the old products are gradually made less important (using the weighted version of MDS) until their influence is zero. Finally, the old products are removed from the map and the new map is optimized. This implementation yields smooth visual transitions, which are important for an effective GUI.

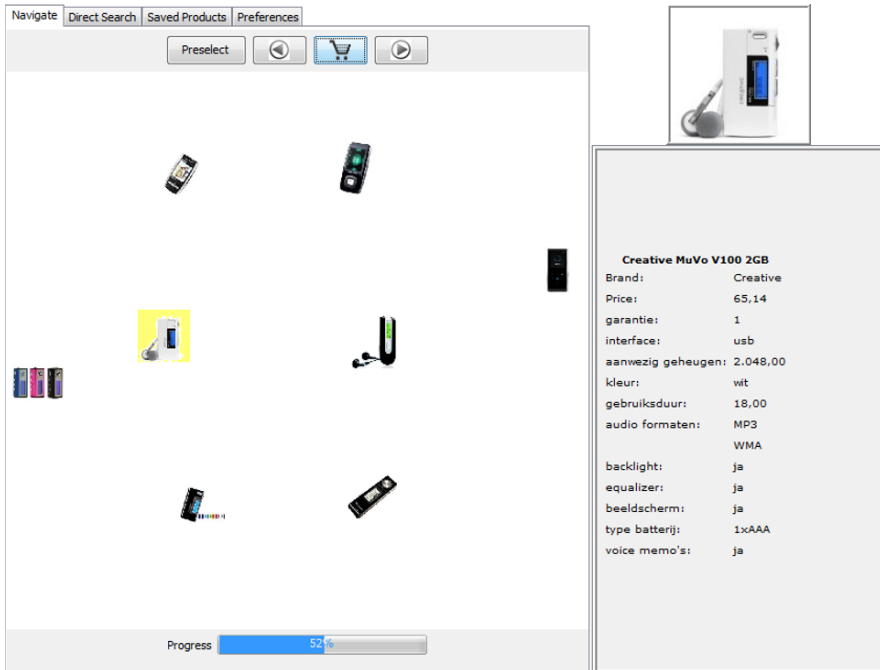


Fig. 17.12: Screenshot of the MDS based product catalog map prototype.

When starting the application, the GSI tab first shows a tree of attributes and attribute values that can be used to preselect a subset of products to be used. In this way, the user can rule out products of which she is certain she is not interested in.

Since the maps of the GSI are dynamically generated, that is, they are dependent on user input and the randomization process, we do not show one here. Using another MP3 player catalog, it was shown in [23] that the theoretical quality (in terms of Stress, see (17.5)) of these maps is high, mainly since they only show about eight products. Also, the recommendation algorithm was tested in a simulation study, showing promising results.

In [23], also a usability study among 71 respondents was reported. Results of this study showed that compared to a traditional list-based interface, the users were not significantly less satisfied with the GSI, while they found it more complex. However, it was only the first time the respondents used the GSI. Some weaknesses of the GSI reported in that usability study have been taken into account in the GSI prototype discussed here. Since the graphical recommender tab was switched off during the experiment, users missed a way to formulate a query. Secondly, people missed a way to restrict the search space, which is facilitated by the selection tree in the new prototype.

However, the main disadvantage of the GSI approach seems to be its complexity. In that sense, the static product catalog maps might be a better alternative. Still, this should be tested in a usability study.

17.7 Conclusions and Outlook

In this chapter, we have discussed how product catalogs can be visualized in a map to provide a way in which e-commerce website users may get a better overview of all products being available. In such a map, products that are similar in their attributes should be located close to each other, while dissimilar products should be located in different areas of the map.

In the framework presented in this chapter, two methods have been used to create such product catalog maps: Multidimensional scaling (MDS) and nonlinear principal components analysis (NL-PCA). MDS has the advantage that it is the only method providing a real distance interpretation. Similarity between products is matched as closely as possible to distances in a two dimensional space. We combined MDS with an adapted version of the Gower coefficient, which is very flexible, since it is able to handle mixed attribute types and missing values. The map made by MDS for the MP3 player application we have shown, seems to have a clear interpretation with a clustering of brands and a important price dimension. The main disadvantage of this map (and MDS in general) is that the map has a circular shape leaving the corners of the map open and positions outliers relatively far away from the rest of the map. However, using a weighting scheme emphasizing the small dissimilarities, this may be overcome.

NL-PCA has the advantage that it is the only method that is able to also visualize attribute categories next to the product visualization. These category points can be used to select subsets of products in the map as was shown in our prototype. In general, the interpretation of the NL-PCA map was in line with the interpretation of the MDS map. Although distinct products also take a large part of the map in the NL-PCA approach, the objects are more spread over the map. The main disadvantage of using the NL-PCA method on our product catalog was that we could not visualize all products, because NL-PCA may create poor maps when introducing objects with too many missing values. Another disadvantage is that the dissimilarity between products is not directly mapped to distances as is done in the MDS method. This can be done in NL-PCA by using a different normalization method. However, then interpretation of category points becomes more difficult which may mean that these cannot be used for navigation anymore.

Since users do usually not consider all product attributes to be equally important we have shown a method based on a Poisson regression model, which can determine attribute importance weights automatically based on counting product popularity in a clickstream log file. Since this method is independent from the visualization technique, it can be used with every technique allowing weights of attributes and it can be even applied in recommender systems which allow for attribute weights. How-

ever, the proposed method has some shortcomings. Weights for categorical attributes are determined in a quite heuristic way and interactions between attributes are ignored. Therefore, we are working on a different way to determine these weights using a more flexible model based on boosted regression trees [25].

Introducing the graphical shopping interface, we have shown one way in which map based visualization could be combined with recommendation techniques, in this case with recommendation by proposing. However, we expect that map based visualization could also be successfully combined with other content- and knowledge-based recommendation techniques, such as critiquing (see [70] and Chapter 13).

Besides combinations with other types of recommendation, we think there are some more challenges in product catalog visualization. First of all, since determining which methods provides the best map is a matter of personal taste and subject to the data to be visualized, one could also try different visualization methods, such as independent component analysis [8] or projection pursuit [12]. A good idea would be to compare different visualization approaches in a user study. In this study, we used a data set of reasonable size. Using larger product catalogs, for instance, having thousands of products, means that both the algorithms used to create the map as well as the interface itself should be able to cope with these numbers. Besides visualizing a large catalog of a single product type, another challenge might be to create a map containing multiple types of products, for instance, different electronic devices.

Acknowledgements

We thank Compare Group for making their product catalog and clickstream log files available to us.

References

1. Alhoniemi, E., Himberg, J., Parviainen, J., Vesanto, J.: SOM Toolbox 2.0 for Matlab 5. Laboratory of Computer and Information Science, Helsinki University of Technology, Helsinki, Finland (1999). Available at <http://www.cis.hut.fi/projects/somtoolbox/>
2. Bederson, B.B., Schneiderman, B., Wattenberg, M.: Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Trans. Graph.* **21**(4), 833–854 (2002)
3. Borg, I., Groenen, P.J.F.: *Modern Multidimensional Scaling*, 2nd edn. Springer Series in Statistics. Springer, New York (2005)
4. Bruls, M., Huizing, K., Van Wijk, J.J.: Squarified treemaps. In: *Proceedings of Joint Eurographics and IEEE TCVG Symposium on Visualization*, pp. 33–42. IEEE Press (2000)
5. Chen, H., Houston, A.L., Sewell, R.R., Schatz, B.R.: Internet browsing and searching: User evaluations of category map and concept space techniques. *J. Am. Soc. Inf. Sci.* **49**(7), 582–603 (1998)
6. Chung, W.: Web searching in a multilingual world. *Commun. ACM* **51**(5), 32–40 (2008)

7. Chung, W., Bonillas, A., Lain, G., Xi, W., Chen, H.: Supporting non-English Web searching: An experiment on the Spanish business and the Arabic medical intelligence portals. *Decis. Support Syst.* **42**, 1697–1714 (2006)
8. Comon, P.: Independent component analysis, a new concept? *Signal Process.* **36**(3), 287–314 (1994)
9. De Leeuw, J.: Correctness of Kruskal's algorithms for monotone regression with ties. *Psychometrika* **42**(1), 141–144 (1977)
10. De Leeuw, J.: Convergence of the majorization method for multidimensional scaling. *J. Classif.* **5**, 163–180 (1988)
11. Donaldson, J.: Music recommendation mapping and interface based on structural network entropy. In: V. Oria, A. Elmagarmid, F. Lochovsky, Y. Saygin (eds.) *Proceedings of the 23rd International Conference on Data Engineering Workshops*, pp. 811–817. IEEE Computer Society (2007)
12. Friedman, J.H., Tukey, J.W.: A projection pursuit algorithm for exploratory data analysis. *IEEE Trans. Comput.* **22**, 881–890 (1974)
13. Gifi, A.: *Nonlinear multivariate analysis*. Wiley, Chichester, UK (1990)
14. Gower, J.C.: A general coefficient of similarity and some of its properties. *Biometrics* **27**, 857–874 (1971)
15. Gower, J.C., Hand, D.J.: *Biplots, Monographs on Statistics and Applied Probability*, vol. 54. Chapman & Hall, London, UK (1996)
16. Green, B.F.: The orthogonal approximation of an oblique structure in factor analysis. *Psychometrika* **17**, 429–440 (1952)
17. Hartigan, J.A., Wong, M.A.: A k-means clustering algorithm. *Appl. Stat.* **28**, 100–108 (1979)
18. Hicklin, J.: *Treemap for Matlab*. Mathworks (2007). Available at <http://www.mathworks.com/matlabcentral/fileexchange/17192>
19. Honaker, J., King, G., Blackwell, M.: *Amelia II: A Program for Missing Data* (2008). R package version 1.1-27, available at <http://gking.harvard.edu/amelia>
20. Ibrahim, J.G., Chen, M.H., Lipsitz, S.R., Herring, A.H.: Missing-data methods for generalized linear models: A comparative review. *J. Am. Stat. Assoc.* **100**(469), 332–346 (2005)
21. Kagie, M., Van Wezel, M., Groenen, P.J.F.: Online shopping using a two dimensional product map. *Lect. Notes Comput. Sci.* **4655**, 89–98 (2007)
22. Kagie, M., Van Wezel, M., Groenen, P.J.F.: Choosing attribute weights for item dissimilarity using clickstream data with an application to a product catalog map. In: *Proceedings of the 2nd ACM Conference on Recommender Systems*, pp. 195–202. ACM Press, New York (2008)
23. Kagie, M., Van Wezel, M., Groenen, P.J.F.: A graphical shopping interface based on product attributes. *Decis. Support Syst.* **46**(1), 265–276 (2008)
24. Kagie, M., Van Wezel, M., Groenen, P.J.F.: An online shopping interface based on a joint product and attribute category map. In: *Proceedings of IUI Workshop on Recommendation and Collaboration ReColl 2008* (2008)
25. Kagie, M., Van Wezel, M., Groenen, P.J.F.: Determination of Attribute Weights for Recommender Systems Based on Product Popularity. Tech. rep. ERS-2009-022-MKT, Erasmus Research Institute in Management, Erasmus University Rotterdam (2009).
26. King, G., Honaker, J., Joseph, A., Scheve, K.: Analyzing incomplete political science data: An alternative algorithm for multiple imputation. *Am. Polit. Sci. Rev.* **95**(1), 49–69 (2001)
27. Kohonen, T.: The self-organizing map. *Proc. IEEE* **78**(9), 1464–1480 (1990)
28. Kohonen, T.: The self-organizing map. *Neurocomputing* **21**, 1–6 (1998)
29. Kohonen, T.: *Self-Organizing Maps*, 3rd edn. Springer Series in Information Sciences. Springer, New York (2001)
30. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* **29**(1), 1–27 (1964)
31. Linting, M., Meulman, J.J., Groenen, P.J.F., Van der Kooij, A.J.: Nonlinear principal components analysis: Introduction and application. *Psychol. Methods* **12**(3), 336–358 (2007)
32. McCullagh, P., Nelder, J.A.: *Generalized Linear Models, Monographs on Statistics and Applied Probability*, vol. 37, 2nd edn. Chapman & Hall, Boca Raton (1989)

33. Meulman, J.J., Heiser, W.J.: SPSS Categories 15. SPSS Inc. (2007)
34. Michailidis, G., De Leeuw, J.: The Gifi system of descriptive multivariate analysis. *Stat. Sci.* **13**(4), 307–336 (1998)
35. Nelder, J.A., Wedderburn, R.W.M.: Generalized linear models. *J. R. Stat. Soc. Ser. A-Stat. Soc.* **135**(3), 370–384 (1972)
36. Ong, T.H., Chen, H., Sung, W., Zhu, B.: Newsmap: a knowledge map for online news. *Decis. Support Syst.* **39**, 583–597 (2005)
37. Pečenović, Z., Do, M.N., Vetterli, M., Pu, P.: Integrated browsing and searching of large image collections. *Lect. Notes in Comput. Sci.* **1929**, 173–206 (2000)
38. Reilly, J., McCarthy, K., McGinty, L., Smyth, B.: Tweaking critiquing. *Knowledge-Based Syst.* **18**(4–5), 143–151 (2005)
39. Resnick, P., Varian, H.R.: Recommender systems. *Commun. ACM* **40**(3), 56–58 (1997)
40. Ricci, F., Wöber, K., Zins, A.: Recommendation by collaborative browsing. In: A.J. Frew (ed.) *Information and Communication Technologies in Tourism 2005*, pp. 172–182. Springer, Vienna (2005)
41. Rubin, D.B.: *Multiple Imputation for Nonresponse in Surveys*. Wiley, New York (1987)
42. Sammon, J.W.: A nonlinear mapping for data structure analysis. *IEEE Trans. Comput.* **18**(5), 401–409 (1969)
43. Schafer, J.L., Olsen, M.K.: Multiple imputation for multivariate missing-data problems: A data analyst's perspective. *Multivariate Behav. Res.* **33**(4), 545–571 (1998)
44. Schneiderman, B.: Tree visualizations with tree-maps: 2-d space filling approach. *ACM Trans. Graph.* **11**(1), 92–99 (1992)
45. Shimazu, H.: ExpertClerk: A conversational case-based reasoning tool for developing salesclerk agents in e-commerce webshops. *Artif. Intell. Rev.* **18**, 223–244 (2002)
46. Stappers, P.J., Pasman, G., Groenen, P.J.F.: Exploring databases for taste or inspiration with interactive multi-dimensional scaling. In: *In Proceedings IEA 2000 / HFES 2000, Ergonomics for the new Millennium*, pp. 3–575–3–578. Santa Monica CA (2000)
47. Torgerson, W.S.: Multidimensional scaling: I. Theory and method. *Psychometrika* **17**, 401–419 (1952)
48. Turetken, O., Sharda, R.: Development of a fisheye-based information search processing aid (FISPA) for managing information overload in the web environment. *Decis. Support Syst.* **37**, 415–434 (2004)
49. Van Gulik, R., Vignoli, F., Van der Wetering, H.: Mapping music in the palm of your hand, explore and discover your collection. In: *Proceedings of the 5th International Conference on Music Information Retrieval* (2004)
50. Yang, C.C., Chen, H., Hong, K.: Visualization of large category map for Internet browsing. *Decis. Support Syst.* **35**, 89–102 (2003)

Part IV
Recommender Systems and Communities

Chapter 18

Communities, Collaboration, and Recommender Systems in Personalized Web Search

Barry Smyth, Maurice Coyle and Peter Briggs

Abstract Web search engines are the primary means by which millions of users access information everyday and the sheer scale and success of the leading search engines is a testimony to the scientific and engineering progress that has been made over the last ten years. However, mainstream search engines continue to deliver largely *one-size-fits-all* services to their user-base, ultimately limiting the relevance of their result-lists. In this chapter we will explore recent research that is seeking to make Web search a more personal and collaborative experience as we look towards a new breed of more social search engines.

18.1 Introduction

Web search engines are among the most important and wide-spread information tools in use today. Every month the leading search engines recommend search results to billions of users and, in the process, generate billions of dollars in advertising revenue annually. In all of this Google stands tall as the clear market leader and one would be forgiven for assuming that all of the major web search challenges have by now been addressed, and that all that remains is the need for some minor algorithmic refinements. The reality is very different however, and while Google may have won the current round of search battles, the web search war is far from over.

Recent research has highlighted how even the leading search engines suffer from low success rates when it comes to delivering relevant results to the average searcher. For example, in one study [24] of more than 20,000 search queries researchers found that, on average, Google delivered at least one result worth selecting only 48% of the time; in other words, in 52% of cases, searchers chose to select none of the results returned. In large part this problem is as much due to the searcher as it is the search

Barry Smyth, Maurice Coyle, Peter Briggs

CLARITY: Centre for Sensor Web Technologies, School of Computer Science & Informatics, University College Dublin, Ireland, {firstname.lastname}@ucd.ie

engine: our search queries tend to be vague and under-specified, and rarely provide a clear indication of our search needs [100, 98, 99, 45, 90]. As frequent searchers we have adapted to these success rates, generally responding to poor result-lists with follow-up or alternative queries. However, at best, this means that web search is far less efficient than it should be — indeed recent studies suggest that among information workers 10% of salary costs are lost due to wasted search time [30] — and at worst a significant proportion of searchers may fail to find the information they need.

Thus, while Google, Yahoo and others continue to provide strong search services for millions of users, there remains plenty of headroom for improvement. In this chapter we will look into the future of web search by reviewing some of most promising research ideas that have the potential to bring game-changing innovation to this exciting technology sector. We will argue that the past is apt to repeat itself, and just as Google's game-changing take on web search led to its relentless rise over the past 10 years, so too will new search technologies emerge to have a similarly disruptive effect on the market over the next 10 years.

Even in their current form, modern search engines can be loosely viewed as a type of recommender system: they respond to users' queries with a set of result page recommendations. But recommendation technologies are set to play an increasingly important role in web search, by helping to address core web search challenges as well as contributing to the solution of a number of secondary search features. For example, recently modern search engines have added *query recommendation* services to supplement core search functionality. As the user enters their query, services like Google Suggest use recommendation techniques to identify, rank and recommend previously successful and relevant queries to the user; see [81]. In this paper, we will focus on two promising and powerful new ideas in web search — personalization and collaboration — that can trace their origins to recent recommender systems research [6, 53, 83, 35, 89, 77] and Chapters 5, 4 and 13. They question the very core assumptions of mainstream web search engines and suggest important adaptations to conventional web search engines. The first assumption concerns the *one-size-fits-all* nature of mainstream web search — two different users with the same query will, more or less, receive the very same result-list, despite their different preferences — and argues that web search needs to become more personalized so that the implicit needs and preferences of searchers can be accommodated. We will describe a number of different approaches to personalizing web search by harnessing different types of user preference and context information to influence the search experience; see for example [19, 23, 33, 97, 2, 48, 49, 108, 22, 69, 86, 14, 31]. The second assumption that will be questioned concerns the *solitary nature* of web search. By and large web search takes the form of a isolated interaction between lone searcher and search engine, however, recent research has suggested that there are many circumstances where the search for information has a distinctly collaborative flavour, with groups of searchers (e.g., friends, colleagues, classmates) cooperating in various ways as they search for and share results. We will describe recent work in the area of *collaborative information retrieval*, which attempts to capitalize on poten-

tial for collaboration during a variety of information seeking tasks; see for example, [70, 71, 73, 72, 58, 59, 94, 1].

In addition we will highlight a new breed of search service that combines elements of personalization and collaboration: so-called *social search* services take advantage of the recent evolution of the web as a social medium, one that promotes interaction and collaboration among individuals during search, so that searchers can benefit from the preferences and experiences of other like-minded individuals. Indeed this provides a new source of information for search engines to use during retrieval: interaction and collaboration information. And this information can be used to drive recommendations at search time so that organic search results, based on term-overlap and link connectivity information, are complimented by additional result recommendations that are based on the preferences and activities of searchers. This will represent a coming together of recommendation systems and search systems and, just as the introduction of connectivity information led to its rise to dominance, there is considerable optimism that this new source of interaction and preference information will lead to an entirely new phase of search engine development in the quest to deliver the right information to the right user at the right time.

18.2 A Brief History of Web Search

Before considering some of the emergent search technologies that have the potential to disrupt the search industry, it is first worth briefly reviewing the history of web search over the past 15 years, to better understand the evolution of modern web search. The early web was not a place of search. Instead if you wanted to get to a particular web page then you either typed the URL directly into your browser, or you used a portal like Yahoo as a starting point to navigate to this page. As the web grew (and grew, and grew) it became clear that portal browsing would not scale, and web search began to emerge in the guise of early search engines such as Lycos, Excite, and Altavista.

These search engines all relied on so-called information retrieval (IR) technologies that had been around since the 1970's [104, 4]. A simplified schematic of a typical search engine architecture is presented in Fig. 18.1. Briefly, early search engines constructed their own index of the web, by *crawling* the web's network of pages and analysing the content of each page in turn, recording the words, and their frequencies, contained in each page. To respond to a search query, the search engine retrieves and ranks pages that contain query terms. During the early days of web search, the emphasis was very much on the size of the index, and search engines that had indexed more of the web had a clear coverage advantage over their rivals. Attention was also paid to the ranking of search results; for the most part, these search engines relied on the frequency of query terms in a web page (relative to the index as a whole) as the primary arbiter of relevance [96], preferring pages that contained frequent occurrences of distinctive query terms. While this approach worked reasonably well in the well-structured, closed-world of information retrieval sys-

tems, where information retrieval experts could be relied upon to submit detailed, well-formed queries, it did not translate well to the scale and heterogenous nature of web content or our vague search queries. The outcome was a poor search experience for most searchers, with relevant results hidden deep within result-lists dominated by results that were, at best, only superficially relevant to the query.

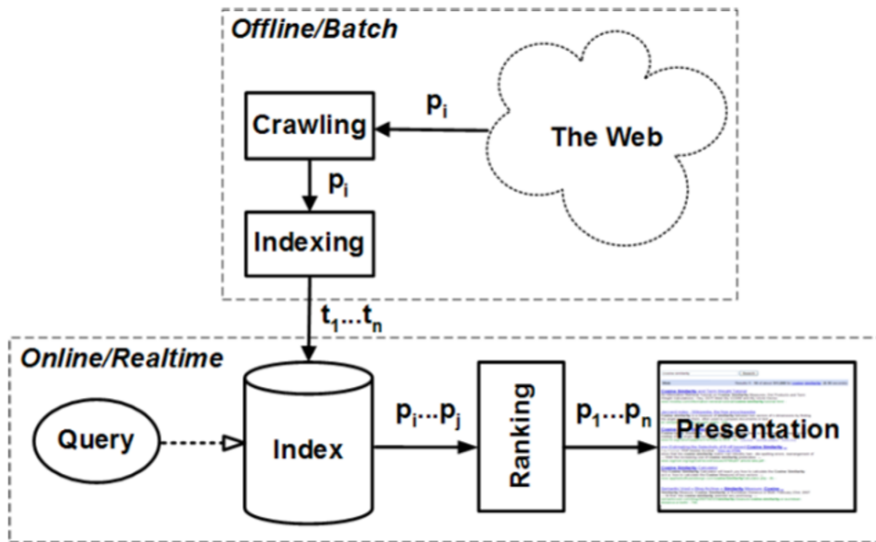


Fig. 18.1: Functional components of a typical web search engine. A page, p_i , is located on the web by the crawler and its content, the terms t_1, \dots, t_n , are retrieved and indexed as part of an offline process. In response to a search query, the engine probes the index to retrieve results which match the query terms, p_i, \dots, p_j , which are then ranked by their relevance according to the search engines particular ranking metrics, before being presented to the searcher as a result-list.

Improving the ranking of search results became the challenge for these early search engines and even the race for the largest search index took a back seat in the face of this more pressing need. It soon became clear, however, that relying solely on the terms in a page was not going to be sufficient, no matter how much time was invested in tweaking these early ranking algorithms. Simply put, there were lots of pages that scored equally well when it came to counting matching query and page terms, but few of these pages turned out to be truly relevant and authoritative. Although term matching information had a role to play in overall relevance, on its own it was insufficient, and it was clear that there was vital information missing from the ranking process.

The missing ingredient came about as a result of research undertaken by a number of groups during the mid 1990's. This included the work of John Kleinberg [40] and, most famously, the work of Google founders Larry Page and Sergey Brin [13].

These researchers were among the first to take advantage of the connectedness of web pages, and they used this information to evaluate the relative importance of individual pages. Kleinberg, Page, and Brin recognised the web as a type of *citation network* (see for example, [60]). Instead of one paper citing another through a bibliographic reference, on the web one page cited another page through a hyperlink connecting the two. Moreover, it seemed intuitive that the importance of a given page should be a function of the various pages that linked to it; the so-called *back-links* of the page. Thus a page could be considered important if lots of other important pages linked to it. This provided the starting point for a fundamentally new way to measure the importance of a page and, separately, the work of [40, 17] and [13] led to novel algorithms for identifying authoritative and relevant pages for even vague web search queries. By the late 1990's Page and Brin's so-called *PageRank* algorithm was implemented in the first version of Google, which combined traditional term-matching techniques with this new approach to link analysis, to provide search results that were objectively superior to the results of other search engines of the day. The rest, as they say, is history.

18.3 The Future of Web Search

There is no doubt that web search represents a very significant recommendation challenge. The size and growth characteristics of the web, and the sheer diversity of content types on offer represent formidable information retrieval challenges in their own right. At the same time, as the demographics of the web's user-base continues to expand, search engines must be able to accommodate a diverse range of user types and search skill levels. In particular, most of us fail to live up to the expectations of the document-centric, term-based information retrieval engines that lie at the heart of modern search technology. These engines, and the techniques they rely upon, largely assume well-formed, detailed search queries, but such queries are far from common in web search today [36, 37, 100, 45]. Instead most web search queries are vague or ambiguous, with respect to the searcher's true information needs, and many queries can contain terms that are not even reflected in the target document(s).

Given that many queries fail to deliver the results that the searcher is looking for there is considerable room for improvement in this most fundamental feature of the search experience. While the problem may reside, at least in part, with the nature of web search queries, as discussed above, it is unlikely that users will improve their query-skills any time soon. In response, researchers have begun to explore two complementary strands of research as a way to improve the overall searcher experience. One widely held view is that web search needs to become more personalized: additional information about users, their preferences and their current context, for example, should be used to deliver a more personalized form of web search by selecting and ranking search results that better match the preferences and context of the individual searcher (see for e.g. [86, 14, 31, 22, 2, 48]). Another view is that there is an opportunity for web search to become more collaborative, by allowing

communities of users to co-operate (implicitly or overtly) as they search (see for e.g. [70, 71, 73, 72, 58, 59, 94, 1]).

In the following sections we will review this research landscape, describing a number of initiatives that are attempting to transform static (non-personalized), solitary (non-collaborative), mainstream search engines into more personalized (see Section 18.3.1) or more collaborative (see Section 18.3.2) search services. These initiatives borrow ideas from recommender systems, user profiling, and computer-supported collaborative working research; see for example [84, 41, 89, 35, 52]. We will also highlight recent research that seeks to bring both of these approaches together leading to a new generation of search services that are both collaborative and personalized. We will refer to these hybrid services as *social search* services and in the remainder of this chapter we will describe two detailed case-studies of two different approaches to social search.

18.3.1 Personalized Web Search

Many recommender systems are designed to make suggestions to users that are relevant to their particular circumstances or their personal preferences — for example, recommender systems help users to identify personally relevant information such as news articles [8, 9, 41], books [46], movies [54, 27, 42], and even products to buy [83, 74, 76, 51, 75, 20]— and the application of recommender technologies to web search allows for a departure from the conventional one-size-fits-all approach to mainstream web search. When it comes to delivering a more personalized search experience there are two key requirements: firstly, we must understand the needs of searchers (*profiling*); secondly, we must be able to use these profiles to influence the output of the search engine, for example by re-ranking results according to the profile, or, indeed, by influencing other components of the web search experience.

To put these research efforts into perspective it is useful to consider two important dimensions to personalizing web search. On the one hand we can consider the nature of the profiles that are learned: some approaches focus on *short-term* user profiles that capture features of the user's current search context (e.g. [86, 14, 31]), while others accommodate *long-term* profiles that capture the user's preferences over an extended period of time (e.g. [22, 2, 48]). On the other hand, when it comes to harnessing these profiles during search, we can usefully distinguish between those approaches that are guided by an *individual* target user's profile (e.g. [15, 89, 38, 43]) versus those that are *collaborative*, in the sense that they are guided by the profiles of a group of users (e.g. [46, 85, 41, 35, 90]).

Generally speaking, user profiles can be constructed in two ways. Explicit profiling interrogates users directly by requesting different forms of preference information, from categorical preferences [22, 48] to simple result ratings [2]. In contrast, *implicit profiling* techniques attempt to infer preference information by monitoring user behaviour, and without interfering with users as they go about their searches; e.g. [22, 47, 69].

With explicit profiling, the users themselves do the profiling work by either specifying search preferences up front, or by providing personal relevance feedback such as rating returned search results. Chirita et al [22] use individual user profiles which are defined by the searcher through ODP¹ web directory categories to re-rank results according to the distance between the profile and ODP categories for each result. They investigate a number of different distance metrics, and report the findings of a live user evaluation that shows that their personalized approach is capable of more relevant result rankings than standard Google search. One of the drawbacks of relying on ODP categories in this way however is that only a small proportion of the web is categorised in the ODP and so many of the returned search results have no category information to base the re-ranking on. Ma et al [48] propose a similar approach whereby user profiles are explicitly expressed through ODP categories, except they re-rank search results based on the cosine similarity between result page content and the ODP directory category profiles. In this way the search results themselves are not required to be categorised in the ODP.

In contrast, *ifWeb* [2] builds user profiles using a less structured approach through keywords, free-text descriptions, and web page examples provided by the user to express their specific information needs, which are stored as a weighted semantic network of concepts. *ifWeb* also takes advantage of explicit relevance feedback where the searcher provides result ratings that are used to refine and update their profile. A similar approach is used by the *Wifs* system [55] in which profiles initially built using terms selected from a list can be subsequently improved with feedback on viewed documents provided by the users. The major drawback with these types of explicit approaches to profiling is that the majority of users are reluctant to make the extra effort in providing feedback [16]. Furthermore, searchers may find it difficult to categorise their information needs and preferences accurately in the first place.

A potentially more successful approach to profiling is to infer user preferences implicitly (*implicit profiling*). As in the work of [22], Liu et al [47] also use hierarchical categories from the ODP to represent a searcher's profile, except in this work the categories are chosen automatically based on past search behaviour such as previously submitted queries and the content of selected result documents. A number of different learning algorithms are analysed for mapping this search behaviour onto the ODP categories, including those based on Linear Least Squares Fit (LLSF) [107], the Rocchio relevance feedback algorithm [78], and k-Nearest Neighbor (kNN) [28]. In a related approach, [103] use statistical language methods to mine contextual information from this type of long-term search history to build a language model based profile, and [69] also infer user preferences based on past behaviour, this time using the browser cache of visited pages to infer subject areas that the user is interested in. These subject areas, or categories, are combined into a hierarchical user profile where each category is also weighted according to the length of time the user spent viewing the pages corresponding to the category.

The above are all examples of long-term user profiles that seek to capture information about the user's preferences over an extended period of time, certainly

¹ The Open Directory Project, <http://dmoz.org>

beyond the bounds of a single search session. The alternative is to capture short-term profiling information, typically related to the particular context of the current information finding task. For example, theUCAIR system [86] concentrates on recently submitted queries and selected results to build a short-term profile that is used to personalize results for the current search task. When a new search session is initiated, a new profile for the user and their current information requirements is created. Similarly Watson [14] and IntelliZap [31] both generate short-term profiles from current context information. Watson identifies informative terms in local documents that the user is editing and web pages that are being browsed, and uses these to modify the user's search queries to personalize results. IntelliZap users initiate a search by selecting a textual query from within a document they are currently viewing, and the search is then guided by additional terms occurring in close proximity to the query terms in the document. In these examples, the profiles guiding the personalization of search results capture context which is pertinent to the users immediate, and possibly temporary, information needs.

The availability of profile and/or context information is the pre-requisite for personalization and there have been a wide range of techniques developed for utilizing profile information to influence different aspects of search experience. These techniques are not limited to influencing the retrieval and ranking of search results, for example, and in fact there has been research on how profiles can be used to influence many other stages in the web search pipeline including the spidering and indexing [32, 44, 34, 29] of raw page content, and query generation [3, 7, 56]. For example, one common way to personalize search results based on a user profile involves using the profile to re-write, elaborate, or expand the original search query so that it returns more specific results that better reflect search interests or context. For example, Koutrika and Ioannidis [43] propose an algorithm they call *QDP* (Query Disambiguation and Personalization) to expand a query submitted by the user according to a user profile represented by weighted relationships between terms. These relationships take the form of operators between terms, such as conjunction, disjunction, negation and substitution, and so in effect the user's profile provides a set of personalized query rewriting rules, which can be applied to the submitted query before it is dispatched to the search engine. Croft et al [26] describe how individualized language models can be used as user profiles with a view to supporting query expansion and relevance feedback. There is also much research in the area of query expansion and disambiguation from the perspective of short term, session-based user profiles from a relevance feedback standpoint which is also highly relevant to work in personalized search [82]. This perspective is not so much targeted at personalizing search per se, but rather at improving search at the level of independent search sessions and many of these approaches can be expanded to encompass longer-term personalized search profiles.

However, perhaps the most popular way to personalize search through user profiles is to directly influence the *ranking* of search results. For example, Jeh and Widom [38] do this by introducing a personalized version of PageRank [13] for setting the query-independent priors on web pages based on user profiles. These profiles consist of a collection of *preferred* pages with high PageRank values which are

explicitly chosen by the user, and used to compute a personalized PageRank score for any arbitrary page based on how related it is to these highly scored preferred pages. Chirita et al [23] build on this idea by automatically choosing these profile pages by analysing the searcher's bookmarked pages and past surfing behaviour, along with a *HubFinder* algorithm that finds related pages with high PageRank scores which are suitable for driving the personalized PageRank algorithm. Both of these approaches are based on long-term user profiles drawn from an extended period of the user's browsing history.

Chang et al [19] propose a personalized version of Kleinberg's HITS [39] ranking algorithm. Their technique harnesses short-term feedback from the searcher, either explicitly or implicitly, to build a profile consisting of a personalized authority list which can then be used to influence the HITS algorithm to personalize the ranking of search results. Experimental results using a corpus of computer science research papers shows that personalized HITS is able to significantly improve result ranking in line with the searcher's preferences, even with only minimal searcher feedback.

Another popular ranking-based approach is the re-ranking of results returned from some underlying, generic web search engine according to searcher preferences without requiring access to the inner workings of the search engine. Speretta and Gauch [97] create individual user profiles by recording the queries and selected result snippets from results returned by Google which are classified into weighted concepts from a reference concept hierarchy. The results from future Google searches are then re-ranked according to the similarity between each result and the searcher's profile concept hierarchy. Rohini and Varma [79] also present a personalized search method where results from an underlying web search engine are re-ranked according to a collaborative filtering technique that harnesses implicitly generated user profiles.

All of the above techniques focus on harnessing single user profiles (the preferences of the target searcher) to personalize that user's search experience. In recommender systems research it is common to take advantage of groups of related profiles when it comes to generating recommendations for a target individual. For instance, the well known *collaborative filtering* approach to recommendation explicitly uses the preferences of a group of users who are similar to the target user when it comes to generating recommendations [77, 85, 46]; see also [35, 52] and Chapter 21. Similar ideas are beginning to influence web search and, indeed, in Section 18.4 we will describe one particular approach that harnesses the preferences of communities of users, albeit in the form of single community profiles rather than a collection of individual user profiles; see also [92, 90]. Sugiyama et al. [101] propose a method whereby long-term user profiles are constructed from similar searchers according to browsing history using a modified collaborative filtering algorithm. The idea is that searchers who issued similar queries and selected similar results in the past can benefit from sharing their search preferences. Sun et al. [102] propose a similar approach called CubeSVD which is also based on collaborative filtering to personalize web search results by analysing the correlation of users, queries and results in click-through data. Both these methods involve the identification of similar searchers to the current searcher in order to create a more comprehensive user profile for the

individual. More recently, the work of [12] describes a peer-to-peer approach to personalizing web search that also leverages the profiles of similar users during result recommendation. Each searcher is profiled in terms of their prior queries and result selections (once again these are long-term profiles). In response to a new target query, recommendations are derived from the user's own personal profile, but in addition, the query is propagated through the peer-to-peer search network so that connected users can also suggest relevant results based on their prior search behaviours. The resulting recommendations are aggregated and ranked according to their relevance to the target query and also in terms of the strength of the *trust* relationship between the target user and the relevant peer; see also recent trust-based recommendation techniques by [63, 65, 64, 66, 67, 62] and Chapter 20.

18.3.2 Collaborative Information Retrieval

Recent studies in specialised information seeking tasks, such as military command and control tasks or medical tasks, have found clear evidence that search-type tasks can be collaborative as information is shared between team members [70, 71, 73, 72]. Moreover, recent work by [57] highlights the inherently collaborative nature of more general purpose web search. For example, during a survey of just over 200 respondents, clear evidence for collaborative search behaviour emerged. More than 90% of respondents indicated that they frequently engaged in collaboration at the level of the *search process*. For example, 87% of respondents exhibited “back-seat searching” behaviours, where they watched over the shoulder of the searcher to suggest alternative queries. A further 30% of respondents engaged in search coordination activities, by using instant messaging to coordinate searches. Furthermore, 96% of users exhibited collaboration at the level of *search products*, that is, the results of searches. For example, 86% of respondents shared the results they had found during searches with others by email. Thus, despite the absence of explicit collaboration features from mainstream search engines there is clear evidence that users implicitly engage in many different forms of collaboration as they search, although, as reported by [57], these collaboration “work-arounds” are often frustrating and inefficient. Naturally, this has motivated researchers to consider how different types of collaboration might be supported by future editions of search engines.

The resulting approaches to *collaborative information retrieval* can be usefully distinguished in terms of two important dimensions, *time* — that is, *synchronous* versus *asynchronous* search — and *place* — that is, *co-located* versus *remote* searchers. Co-located systems offer a collaborative search experience for multiple searchers at a single location, typically a single PC (e.g. [1, 87]) whereas remote approaches allow searchers to perform their searches at different locations across multiple devices; see e.g. [58, 59, 94]. The former enjoy the obvious benefit of an increased faculty for direct collaboration that is enabled by the face-to-face nature of co-located search, while the latter offer a greater opportunity for collabo-

rative search. Alternatively, synchronous approaches are characterised by systems that broadcast a “call to search” in which specific participants are requested to engage in a well-defined search task for a well defined period of time; see e.g. [87]. In contrast, asynchronous approaches are characterised by less well-defined, ad-hoc search tasks and provide for a more open-ended approach to collaboration in which different searchers contribute to an evolving search session over an extended period of time; see e.g. [58, 92].

A good example of the co-located, synchronous approach to collaborative web search is given by the work of [1]. Their CoSearch system is designed to improve the search experience for co-located users where computing resources are limited; for example, a group of school children having access to a single PC. CoSearch is specifically designed to leverage peripheral devices that may be available (e.g. mobile phones, extra mice etc.) to facilitate distributed control and division of effort, while maintaining group awareness and communication. For example, in the scenario of a group of users collaborating though a single PC, but with access to multiple mice, CoSearch supports a *lead searcher* or *driver* (who has access to the keyboard) with other users playing the role of search *observers*. The former performs the basic search task but all users can then begin to explore the results returned by independently selecting links so that pages of interest are added to a page queue for further review. The CoSearch interface also provides various opportunities for users to associate notes with pages. Interesting pages can be saved and as users collaborate a *search summary* can be created from the URLs and notes of saved pages. In the case where observers have access to mobile phones, CoSearch supports a range of extended interface functionality to provide observers with a richer set of independent functionality via a bluetooth connection. In this way observers can download search content to their mobile phone, access the page queue, add pages to the page queue and share new pages with the group.

The purpose of CoSearch is to demonstrate the potential for productive collaborative web search in resource-limited environments. The focus is very much on dividing the search labour while maintaining communication between searchers, and live user studies speak to the success of CoSearch in this regard [1]. The work of [88] is related in spirit to CoSearch but focuses on image search tasks using a table-top computing environment, which is well suited to supporting collaboration between co-located users who are searching together. Once again, preliminary studies speak to the potential for such an approach to improve overall search productivity and collaboration, at least in specific types of information access tasks, such as image search, for example. A variation on these forms of synchronous search activities is presented in [87], where the use of mobile devices as the primary search device allows for a remote form of synchronous collaborative search. The iBingo system allows a group of users to collaborate on an image search task with each user using a ipod touch device as their primary search/feedback device (although conventional PCs appear to be just as applicable). Interestingly, where the focus of CoSearch is largely on the division of search labour and communication support, iBingo offers the potential to use relevance feedback from any individual searcher to the benefit of others. Specifically, the iBingo collaboration engine uses information about the

activities of each user in order to encourage other users to explore different information trails and different facets of the information space. In this way, the ongoing activities of users can have an impact on future searches by the group and, in a sense, the search process is being “personalized” according to the group’s search behaviour.

Remote search collaboration (whether asynchronous or synchronous) is the aim of SearchTogether, which allows groups of searchers to participate in extended shared search sessions as they search to locate information on particular topics; see also [58]. In brief, the SearchTogether system allows users to create shared search sessions and invite other users to join in these sessions. Each searcher can independently search for information on a particular topic, but the system provides features to allow individual searchers to share what they find with other session members by recommending and commenting on specific results. In turn, SearchTogether supports synchronous collaborative search by allowing searchers to invite others to join in specific search tasks, allowing cooperating searchers to synchronously view the results of each others’ searches via a split-screen style results interface. As with CoSearch above, one of the key design goals in SearchTogether is to support a division of labour in complex, open-ended search tasks. In addition, a key feature of the work is the ability to create a shared awareness among group members by reducing the overhead of search collaboration at the interface level. SearchTogether does this by including various features, from integrated messaging, query histories, and recommendations arising out of recent searches.

In the main, the collaborative information retrieval systems we have so far examined have been largely focused on supporting collaboration from a division of labour and shared awareness standpoint, separate from the underlying search process. In short, these systems have assumed the availability of an underlying search engine and provided a collaboration interface that effectively *imports* search results directly, allowing users to share these results. As noted by [68], one of the major limitations of these approaches is that collaboration is restricted to the interface in the sense that while individual searchers are notified about the activities of collaborators, they must individually examine and interpret these activities in order to reconcile their own activities with their co-searchers. Consequently, the work of [68] describes an approach to collaborative search that is more tightly integrated with the underlying search engine resource so that the operation of the search engine is itself influenced by the activities of collaborating searchers in a number of ways. For example, mediation techniques are used to prioritise, as yet, unseen documents, while query recommendation techniques are used to suggest alternative avenues for further search exploration.

18.3.3 Towards Social Search

So far we have focused on two separate strands of complementary research in the field of web search and information finding motivated by questions that cut to the

very core of conventional web search. The one-size-fits-all nature of mainstream web search is questioned by researchers developing more personalized web search techniques, and the assumption that search is largely a solitary experience is questioned by recent studies that highlight the inherently collaborative nature of many search scenarios.

To date, these different strands of research have been separated by different motivations and objectives. The world of personalized search, for example, has been largely guided by the need to produce result-lists that are better targeted to the needs of the individual searcher, whereas collaborative information retrieval has focused on supporting groups of searchers by facilitating the division of search labour and by promoting shared awareness among cooperating searchers. However both of these research communities are linked by a common thread of research from the recommender systems field and a recommender systems perspective has helped to identify opportunities to bring these two different strands of research together. In what follows we will describe two related case-studies that attempt to bring together these strands of research in the pursuit of web search techniques that are both collaborative and personalized. The result is an approach to web search that is both more collaborative — each case study assumes the involvement of groups of searchers — and more personalized, albeit at the level of the group rather than the individual searcher. Both of these case-studies will describe remote, asynchronous forms of collaborative web search and we will summarize the results of recent live-user studies to highlight their potential end-user benefits. In each case we will describe the central role that recommendations play in adding-value to a conventional search result-list. For example, we will describe how the preferences and activities of communities and groups of users can be harnessed to promote recommended search results in addition to conventional result-lists.

18.4 Case-Study 1 - Community-Based Web Search

In this first case-study we review recent work in the area of Community-based web search in which the search activities of communities of like-minded users are used to augment the results of a mainstream search engine to provide a more focused community-oriented result-list; see [91, 92]. This can include well-defined or ad-hoc communities, and our aim is to take advantage of the query repetition and selection regularity that naturally occurs within the search behaviour of such communities as a source of result recommendations. In this case-study we describe and evaluate one particular implementation of this approach to web search that has been designed to work with a mainstream search engine such as Google.

18.4.1 Repetition and Regularity in Search Communities

There are many scenarios in which search can be viewed as a community-oriented activity. For example, the employees of a company will act as a type of search community with overlapping information needs. Similarly, students in a class may serve as a search community as they search for information related to their class-work. Visitors to a themed website (e.g., a wildlife portal or a motoring portal) will tend to share certain niche interests and will often use the site's search facilities to look for related information. And of course, groups of friends on a social networking site may act as a community with shared interests.

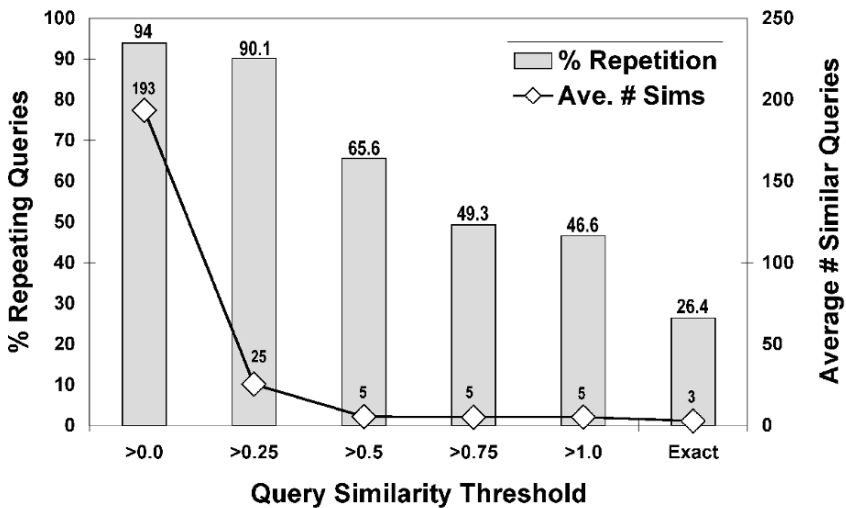


Fig. 18.2: Repetition and similarity amongst the search queries used by the employees of a software company.

We became interested in these emergent search communities because we believed that there was a high likelihood that similarities would exist between the search patterns of community members. For example, Figure 18.2 presents the results of a recent 17-week study of the search patterns for 70 employees of a local software company; this study preceded the trial discussed later in this paper. During the study we examined more than 20,000 individual search queries and almost 16,000 result selections. We see that, on average, just over 65% of queries submitted shared at least 50% (> 0.5 similarity threshold) of their query terms with at least 5 other queries; and more than 90% of queries shared at least 25% of their terms with about 25 other queries. In other words, searchers within this ad hoc corporate search community do search for similar things in similar ways, much more so than

in generic search scenarios where we typically find much lower repetition rates of about 10% at the 0.5 similarity threshold [92].

This is an important result which is supported by similar studies on other communities of searchers [92], and which motivates our collaborative web search approach. It tells us that, in the context of communities of like-minded searchers, the world of web search is a repetitive and regular place. A type of community search knowledge is generated from the search experiences of individuals as they search. This in turn suggests that it may be possible to harness this search knowledge by facilitating the sharing of search experiences among community members. So, as a simple example, when a visitor to the previously mentioned wildlife portal searches for “jaguar pictures” they can be recommended search results that have been previously selected by other community members for *similar* queries. These results will likely relate to the wildlife interests of the community and so, without any expensive processing of result content, we can personalize search results according to the learned preferences of the community. In this way, novice searchers can benefit from the shared knowledge of more experienced searchers.

18.4.2 The Collaborative Web Search System

Figure 18.3 presents the basic architecture for our collaborative web search system, which is designed to work alongside an underlying mainstream search engine — in this case, Google. Briefly, a proxy-based approach is adopted to intercept queries on their way to the underlying search engine, and to manipulate the results that are returned from this engine back to the searcher. In this way users get to use their favourite search engine in the normal way, but with collaborative web search (CWS) promotions incorporated into the result-lists directly via the proxy. For example, consider a user U_i submitting query q_T to Google. This request is redirected to the CWS system whereupon two things happen. First, the query is passed on to Google and the result-list R_S is returned in the normal way. Second, in parallel the query is also used to access a local store of the search activity for U_i 's community – the CWS *hit-matrix* – to generate a ranked set of promotion candidates, R_P , as outlined below. These promotion candidates are annotated by the *explanation engine* to present the searcher with a graphical representation of their community history. Result-lists R_P and R_S are merged and the resulting list R_{final} is returned to the user; typically this merge involves promoting the k (e.g., $k = 3$) most relevant promotions to the head of the result-list.

Thus for a target search query, CWS combines a default result-list, R_S , from a standard search engine, with a set of recommended (*promoted*) results, R_P , drawn from the community's past search history. To do this the search histories of a given community, C , of users ($C = \{U_1, \dots, U_n\}$) are stored in a *hit-matrix*, H^C , such that each row corresponds to some query q_i and each column to some selected result page p_j . The value stored in H_{ij}^C refers to the number of times that page p_j has been selected for query q_i by members of C . In this way, each hit-matrix acts as a repos-

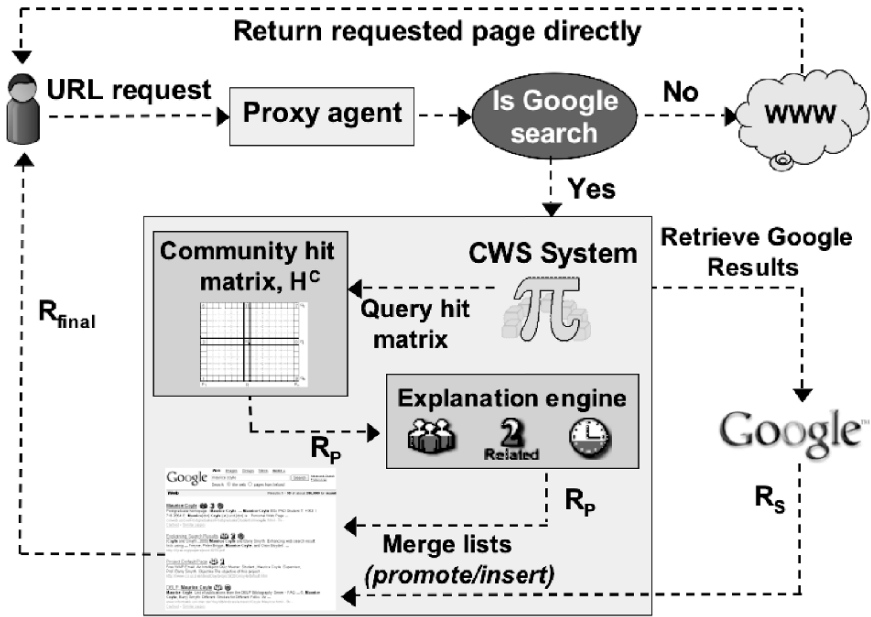


Fig. 18.3: Proxy architecture for a CWS system.

itory of community search experiences: the results that the community members have found to be relevant for their queries.

$$Relevance(p_j, q_i) = \frac{H_{ij}}{\sum_{\forall j} H_{ij}} \tag{18.1}$$

$$Sim(q, q') = \frac{|q \cap q'|}{|q \cup q'|} \tag{18.2}$$

$$WRel(p_j, q_T, q_1, \dots, q_n) = \frac{\sum_{i=1 \dots n} Relevance(p_j, q_i) \bullet Sim(q_T, q_i)}{\sum_{i=1 \dots n} Exists(p_j, q_i) \bullet Sim(q_T, q_i)} \tag{18.3}$$

When responding to a new target query, q_T , H_C is used to identify and rank results that have been regularly selected in the past. The relevance of a result p_j in relation to a query q_i can be estimated by the relative frequency that p_j has been selected for q_i in the past, as shown in Equation 18.1. More generally, we can pool the results that have been selected for queries that are similar to q_T (see Equation 18.2) and rank each result according to the weighted model of relevance shown in Figure 18.3, which weights each individual result’s relevance by the similarity of the associated

query to q_T ; note that the predicate *Exists* returns 1 if page p_j has been previously selected for query q_i in the target community, and 0 otherwise.

The screenshot shows a Google search interface. At the top left is the Google logo. To its right are navigation links: Web, Images, Groups, News, and more. A search input field contains the text 'michael jordan', followed by a 'Search' button and a link to 'Advanced & Preferences'. Below the search bar, there are radio buttons for 'the web' (selected) and 'pages from Ireland'. A horizontal bar below the search bar indicates 'Web' and 'Results 1 - 10 of about 74,300'. The search results are listed below, starting with 'NBA.com: Michael Jordan Bio' and 'NBA.com: Michael Jordan Summary'. The first result is a link to 'www.nba.com/playerfile/michael_jordan.html' with a snippet: 'Michael Jordan | 23. Season statistics & Notes · Season splits · Game-by-game stats · Bio · Printable player file. 2002-03 Statistics. PPG, 20.0. RPG, 6.10 ...'. The second result is a link to 'www.nba.com/history/players/jordan_summary.html' with a snippet: 'NBA.com: Michael Jordan Summary Michael Jordan By acclamation, Michael Jordan is the greatest basketball player of all time. Although, a summary of his basketball career and influence on ...'. The third result is a link to 'en.wikipedia.org/wiki/Michael_Jordan' with a snippet: 'Michael Jordan - Wikipedia, the free encyclopedia Michael Jordan's basketball talent was clear from his rookie season in the NBA. ... NAME, Jordan, Michael Jeffrey. ALTERNATIVE NAMES, MJ; Air Jordan ...'. Each result includes a snippet and a link to 'Cached' or 'Similar pages'.

Fig. 18.4: The result-list returned by Google in response to the query ‘michael jordan’.

Figures 18.4 and 18.5 present example screen shots for the result-list returned by Google for the query ‘*Michael Jordan*’. In the case of Figure 18.4 we see the default Google result-list, with results for the basketball star clearly dominating. In Figure 18.5, however, we see a result-list that has been modified by our proxy-based version of CWS, trained by (in this example) a community of computer science researchers. The results are presented through the standard Google interface, but we see that the top 3 results are promotions for the well-known Berkeley professor. In addition, promoted results are annotated with explanation icons designed to capture different aspects of the result’s community history. These include icons that capture the popularity of the result among community members, information about how recently it has been selected, and information about the other queries that have led to its selection.

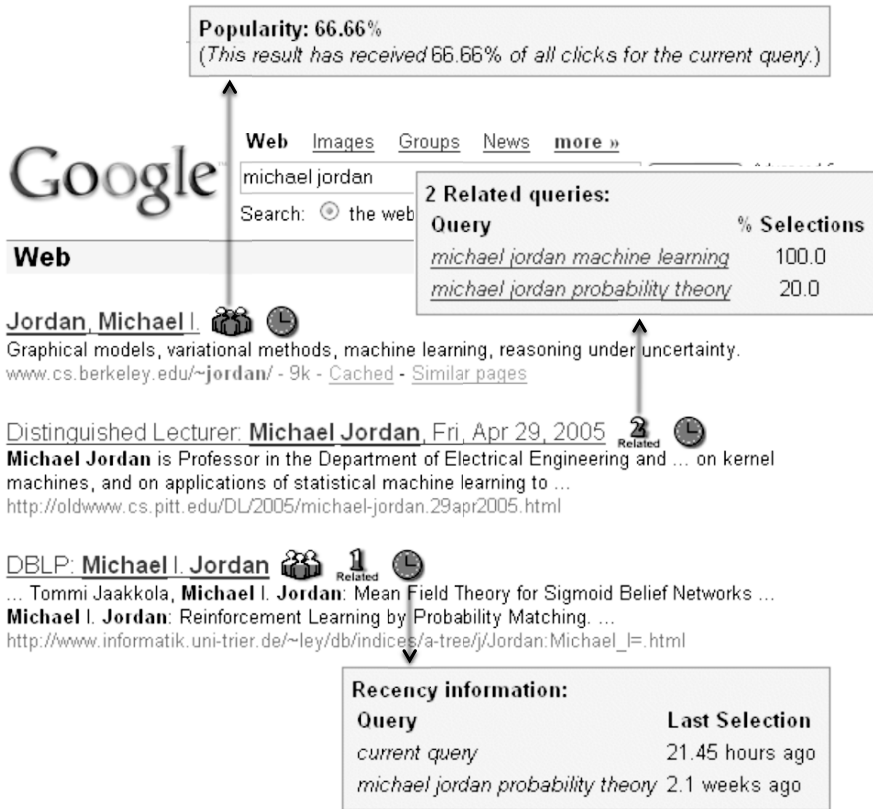


Fig. 18.5: The result-list returned by CWS in response to the query ‘michael jordan’ issued within a community with a shared interest in computer science. The extra explanation information available by mousing-over each promoted result icon type is also shown.

18.4.3 Evaluation

The current proxy-based architecture has been used as the basis of a long-term trial of the CWS approach in a corporate search scenario. In this section we will describe some recent results drawn from this trial, which speak to the value of the community-based promotions offered by CWS.

The trial participants included the 70+ employees of a local Dublin software company where the CWS architecture was configured to work with the standard Google search engine so that all Google requests were redirected through the CWS system. The search experience was based on the standard Google interface with a maximum of 3 results promoted (and annotated with explanations) in any session;

if more than 3 promotions were available then non-promoted results were annotated with explanation icons, but left in their default Google position. The results presented here are drawn from just over 10 weeks of usage and cover a total of 12,621 individual search sessions.

One of the challenges in evaluating new search technologies in a natural setting is how to evaluate the quality of individual search sessions. Ideally we would like to capture direct relevance feedback from users as they search. While it would be relatively straightforward to ask users to provide such feedback during each session, or as they selected specific results, this was not feasible in the current trial because participants were eager to ensure that their search experience did not deviate from the norm, and were unwilling to accept pop-ups, form-filling or any other type of additional feedback. As an alternative, in this evaluation, we used a less direct measure of relevance based on the concept of a *successful session* (see also [92, 91]). We define a successful session to be one where at least one search result has been selected, indicating that the searcher has found at least one (partially) relevant result. In contrast, search sessions where the user does not select any results are considered to be unsuccessful, in the sense that the searcher has found no relevant results. While this is a relatively crude measure of overall search performance, it at least allows us to compare search sessions in a systematic way.

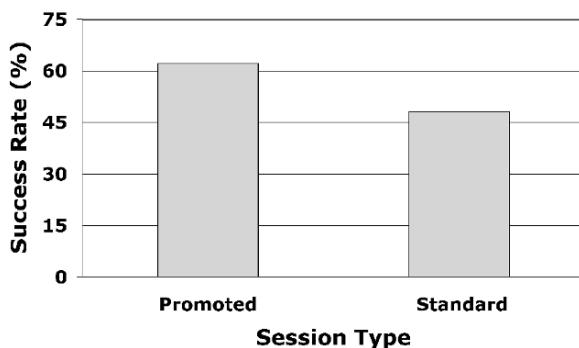


Fig. 18.6: The success rates for sessions containing promotions compared to those without promotions.

A comparison of success rates between sessions with promotions (*promoted sessions*) and search sessions without promotions (*standard sessions*) is presented as Figure 18.6. The results show that during the course of the 10 week trial, on average, sessions with promotions are more likely to be successful (62%) than standard sessions (48%) containing only Google results, a relative benefit of almost 30% due to the community-based promotion of results. In other words, during the course of the trial we found that for more than half of the standard Google search sessions users failed to find any results worth selecting. In contrast, during the same period,

the same searchers experienced a significantly greater success rate for sessions that contained community promotions, with less than 40% of these sessions failing to attract user selections. Within an enterprise these results can have an important impact when it comes to overall search productivity because there are significant savings to be made by eliminating failed search sessions in many knowledge-intensive business scenarios. For example, a recent report [30] by the International Data Corporation (IDC) found that, on average, knowledge workers spend 25% of their time searching for information, and an enterprise employing 1,000 knowledge workers will waste nearly \$2.5 million per year (at an opportunity cost of \$15 million) due to an inability to locate and retrieve information. In this context any significant reduction in the percentage of failed search sessions can play an important role in improving enterprise productivity, especially in larger organisations.

18.4.4 Discussion

The model of collaborative web search presented here is one that seeks to take advantage of naturally occurring query repetition and result selection regularity among communities of like-minded searchers. In this case-study we have focused on one particular type of search community in the form of a group of employees. Obviously this is a reasonably straightforward community to identify and it is perhaps not surprising that we have found a high degree of repetition and regularity to take advantage of during collaborative web search. Nonetheless, this type of community, where groups of individuals come together to perform similar information finding tasks, is a common one, whether it is employees in a company or students in a class or researchers in a research group.

There are of course many other types of community. For example, we have already mentioned the scenario where a group of visitors to a themed web site can be considered to be an ad-hoc search community. More generally, it is interesting to consider the open question of community discovery and identification, and there is considerable research at the present time devoted to exploring various approaches to automatically identifying online communities; see for example [11, 5, 21, 106, 105]. And as we develop a better understanding of the nature of online communities in the new world of the social web it may be possible to offer a more flexible form of search collaboration, facilitated by a more flexible and dynamic definition of search community.

18.5 Case-Study 2 - Web Search. Shared.

The previous case-study looked at a community-oriented view of collaborative web search, where the search activities of like-minded communities of searchers were used to influence mainstream search engine results. In this section we describe an

alternative model of collaborative web search, as implemented in a system called HeyStaks, that is different in two important ways. First of all, HeyStaks adopts more user-led approach to collaborative web search, one that is focused on helping users to better organise and share their search experiences. HeyStaks does this by allowing users to create and share repositories of search experiences as opposed to coordinating the participation of search communities. Secondly, we adopt a very different approach to search engine integration. Instead of the proxy-based approach described in the previous case-study, HeyStaks is integrated with a mainstream search engine, such as Google, through a browser toolbar, which provides the collaborative search engine with the ability to capture and guide search activities. Finally, we will also summarize the findings of a recent live-user study to investigate the nature of search collaboration that manifests within HeyStaks' user population.

18.5.1 The HeyStaks System

HeyStaks adds two basic features to a mainstream search engine. First, it allows users to create *search staks*, as a type of folder for their search experiences at search time. Staks can be shared with others so that their searches will also be added to the stak. Second, HeyStaks uses staks to generate recommendations that are added to the underlying search results that come from the mainstream search engine. These recommendations are results that stak members have previously found to be relevant for similar queries and help the searcher to discover results that friends or colleagues have found interesting, results that may otherwise be buried deep within Google's default result-list.

As per Fig. 18.7, HeyStaks takes the form of two basic components: a client-side *browser toolbar* and a back-end *server*. The toolbar allows users to create and share staks and provides a range of ancillary services, such as the ability to tag or vote for pages. The toolbar also captures search click-throughs and manages the integration of HeyStaks recommendations with the default result-list. The back-end server manages the individual stak indexes (indexing individual pages against query/tag terms and positive/negative votes), the stak database (stak titles, members, descriptions, status, etc.), the HeyStaks social networking service and, of course, the recommendation engine. In the following sections we will briefly outline the basic operation of HeyStaks and then focus on some of the detail behind the recommendation engine.

Consider the following motivating example. Steve, Bill and some friends are planning a European vacation and they know that during the course of their research they will use web search as their primary source of information about what to do and where to visit. Steve creates a (private) search stak called "European Vacation 2008" and shares this with Bill and friends, encouraging them to use this stak for their vacation-related searches.

Fig. 18.8 shows Steve selecting this stak as he embarks on a new search for "Dublin hotels", and Fig. 18.9 shows the results of this search. The usual Google results are shown, but in addition HeyStaks has made two promotions. These have

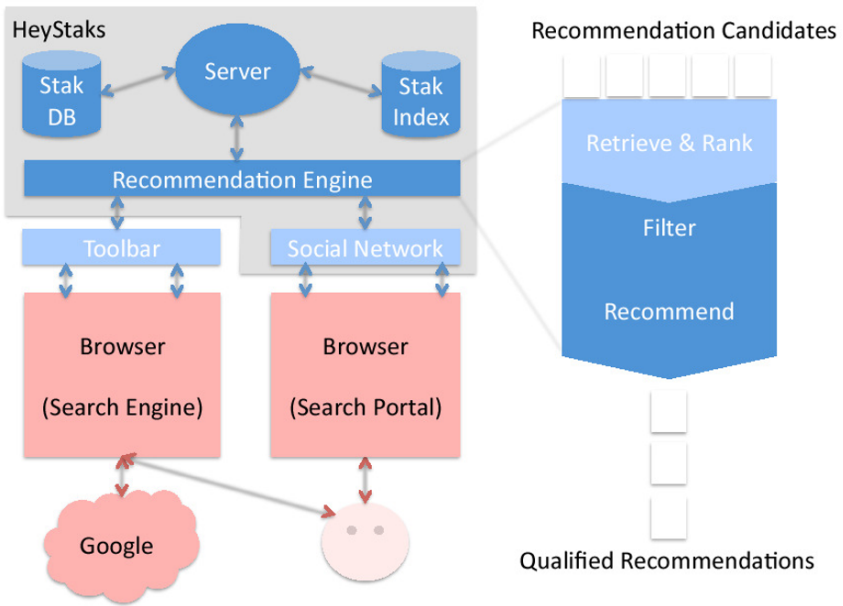


Fig. 18.7: The HeyStaks system architecture and outline recommendation model.



Fig. 18.8: Selecting a new active stak.

been promoted because other members of the “European Vacation 2008“ stak had recently found these results to be relevant; perhaps they selected them for *similar* queries, or voted for them, or tagged them with related terms. These recommendations may have been promoted from much deeper within the Google result-list, or they may not even be present in Google’s default results for the target query. Other relevant results may also be highlighted by HeyStaks, but left in their default Google position. In this way Steve and Bill benefit from promotions that are based on their previous similar searches. In addition, HeyStaks can recommend results from other related public staks as appropriate, helping searchers to benefit from the search knowledge that other groups and communities have created.

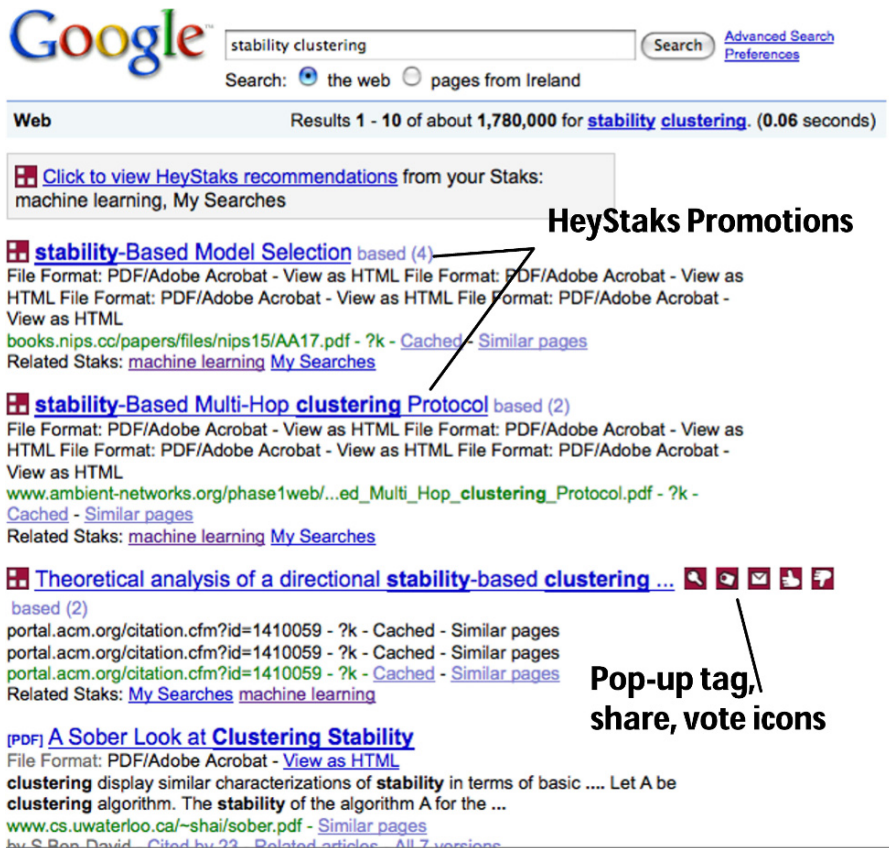


Fig. 18.9: Google search results with HeyStaks promotions.

Separately from the toolbar, HeyStaks users can also benefit from the HeyStaks *search portal*, which provides a social networking service built around people’s search histories. For example, Fig. 18.10 shows the portal page for the “European Vacation 2008“ stak, which is available to all stak members. It presents an activity

feed of recent search history and a query cloud that makes it easy for the user to find out about what others have been searching for. The search portal also provides users with a wide range of features such as stak maintenance (e.g., editing, moving, copying results in staks and between staks), various search and filtering tools, and a variety of features to manage their own search profiles and find new search partners.

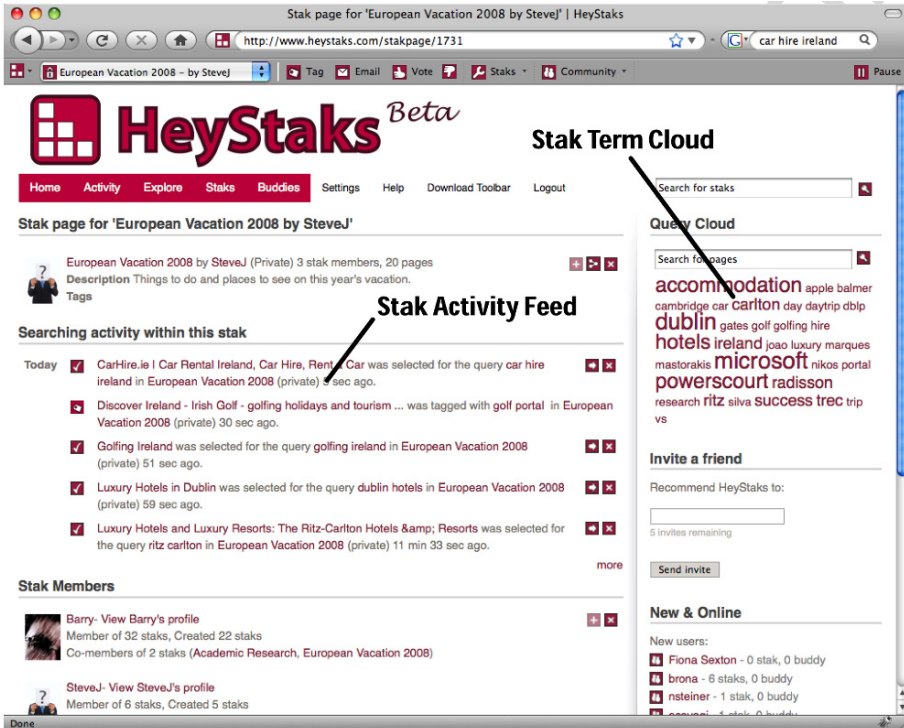


Fig. 18.10: The HeyStaks search portal provide direct access to staks and past searches.

18.5.2 The HeyStaks Recommendation Engine

In HeyStaks each search stak (S) serves as a profile of the search activities of the stak members and HeyStaks combines a number of implicit and explicit profiling techniques to capture a rich history of search experiences. Each stak is made up of a set of result pages ($S = \{p_1, \dots, p_k\}$) and each page is anonymously associated with a number of implicit and explicit interest indicators, including the total number of times a result has been selected (sel), the query terms (q_1, \dots, q_n) that led to its

selection, the number of times a result has been tagged (*tag*), the terms used to tag it (t_1, \dots, t_m), the votes it has received (v^+, v^-), and the number of people it has been shared with (*share*) (all explicit indicators of interest) as indicated by Eq. 18.4.

$$p_i^S = \{q_1, \dots, q_n, t_1, \dots, t_m, v^+, v^-, sel, tag, share\} \quad (18.4)$$

In this way, each page is associated with a set of *term data* (query terms and/or tag terms) and a set of *usage data* (the selection, tag, share, and voting count). The term data is represented as a Lucene (lucene.apache.org) index table, with each page indexed under its associated query and tag terms, and provides the basis for retrieving and ranking *promotion candidates*. The usage data provides an additional source of evidence that can be used to filter results and to generate a final set of recommendations. At search time, a set of recommendations is produced in a number of stages: relevant results are retrieved and ranked from the Lucene stak index; these promotion candidates are filtered based on an *evidence model* to eliminate noisy recommendations; and the remaining results are added to the Google result-list according to a set of *recommendation rules*.

Briefly, there are two types of promotion candidates: *primary promotions* are results that come from the active stak S_t ; whereas *secondary promotions* come from other staks in the searcher's stak-list. To generate these promotion candidates, the HeyStaks server uses the current query q_t as a probe into each stak index, S_i , to identify a set of relevant stak pages $P(S_i, q_t)$. Each candidate page, p , is scored using Lucene's *TF.IDF* retrieval function as per 18.5, which serves as the basis for an initial recommendation ranking.

$$score(q_t, p) = \sum_{t \in q_t} tf(t \in p) \bullet idf(t)^2 \quad (18.5)$$

Staks are inevitably noisy, in the sense that they will frequently contain pages that are not on topic. For example, searchers will often forget to set an appropriate stak at the start of a new search session and, although HeyStaks includes a number of automatic stak-selection techniques to ensure that the right stak is active for a given search, these techniques are not perfect, and misclassifications do inevitably occur; see also [18, 95]. As a result, the retrieval and ranking stage may select pages that are not strictly relevant to the current query context. To avoid making spurious recommendations HeyStaks employs an *evidence filter*, which uses a variety of threshold models to evaluate the relevance of a particular result, in terms of its usage evidence; tagging evidence is considered more important than voting, which in turn is more important than implicit selection evidence. For example, pages that have only been selected once, by a single stak member, are not automatically considered for recommendation and, all other things being equal, will be filtered out at this stage. In turn, pages that have received a high proportion of negative votes will also be eliminated. The precise details of this model are beyond the scope of this paper but suffice it to say that any results which do not meet the necessary evidence thresholds are eliminated from further consideration.

After evidence pruning we are left with revised primary and secondary promotions and the final task is to add these *qualified recommendations* to the Google result-list. HeyStaks uses a number of different recommendation rules to determine how and where a promotion should be added. Once again, space restrictions prevent a detailed account of this component but, for example, the top 3 primary promotions are always added to the top of the Google result-list and labelled using the HeyStaks promotion icon. If a remaining primary promotion is also in the default Google result-list then this is labeled in place. If there are still remaining primary promotions then these are added to the secondary promotion list, which is sorted according to TF.IDF scores. These recommendations are then added to the Google result-list as an optional, expandable list of recommendations; for further details see [93, 94]

18.5.3 Evaluation

In this section we examine a subset of 95 HeyStaks users who have remained active during the course of the early beta release of the toolbar and service. These users registered with HeyStaks during the period October-December 2008 and the results below represent a summary of their usage during the period October 2008 - January 2009. Our aim is to gain an understanding of both how users are using HeyStaks, and whether they seem to be benefiting from its search promotions. Because this is a study of live-users *in the wild* there are certain limitations about what we have been able to measure. There is no control group, for example, and it was not feasible, mainly for data privacy reasons, to analyse the relative click-through behaviour of users, by comparing their selections of default Google results to their selections of HeyStaks promotions. However, for the interested reader, our earlier work does report on this type of analysis in more conventional control-group laboratory studies [10, 25, 92].

Key to the HeyStaks proposition is that searchers need a better way to organise and share their search experiences. HeyStaks provides these features but do users actually take the time to create staks? Do they share them with others or join those created by others?

During the course of the initial deployment of HeyStaks users did engage in a reasonable degree of stak creation and sharing activity. For example, as per Fig. 18.11, on average, beta users created just over 3.2 new staks and joined a further 1.4. Perhaps this is not surprising: most users create a few staks and share them with a small network of colleagues or friends, at least initially.

In total there were over 300 staks created on a wide range of topics, from broad topics such as travel, research, music and movies, to more niche interests including archaeology, black and white photography, and mountain biking. A few users were prolific stak creators and joiners: one user created 13 staks and joined another 11, to create a search network of 47 other searchers (users who co-shared the same staks).

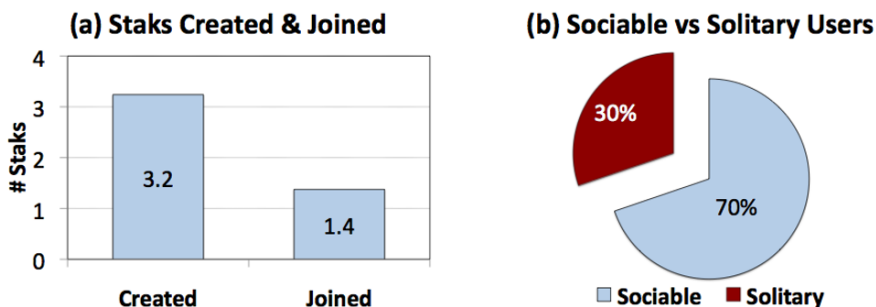


Fig. 18.11: (a) Average staks created and joined per user. (b) The percentage of *sociable* and *solitary* users.

In fact on average, each user was connected to a search network of just over 5 other searchers by the staks that they shared.

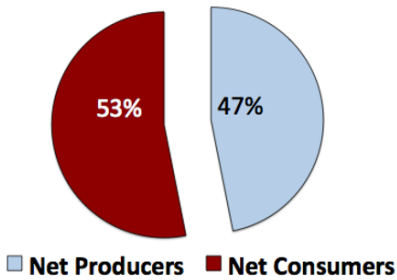
The vast majority of staks were created as public staks, although most (52%) remained the domain of a single member, the stak creator. Thus 48% of staks were shared with at least one other user and, on average, these staks attracted 3.6 members. Another way to look at this is as depicted in Fig. 18.11(b): 70% of users make the effort to share or join staks (*sociable* users); and only 30% of users created staks just for their own personal use and declined to join staks created by others (*solitary* users).

At its core HeyStaks is motivated by the idea that web search is an inherently social or collaborative activity. And even though mainstream search engines do not support this, searchers do find alternative collaboration channels (e.g., email, IM, etc.) with which to partially, albeit inefficiently, share their search experiences; see for example [57]. One of the most important early questions to ask about HeyStaks users concerns the extent to which their natural search activity serves to create a community of collaborating searchers. As users search, tag, and vote they are effectively producing and consuming community search knowledge. A user might be the first to select or tag a given result for a stak and, in this context, they have *produced* new search knowledge. Later, if this result is promoted to another user and then re-selected (or tagged or voted on), then this other user is said to have *consumed* that search knowledge; of course they have also produced search knowledge as their selection, tag, or vote is added to the stak.

We have found that 85% of users have engaged in search collaborations. The majority have consumed results that were produced by at least one other user, and on average these users have consumed results from 7.45 other users. In contrast 50% of users have produced knowledge that has been consumed by at least one other user, and in this case each of these producers has created search knowledge that is consumed by more than 12 other users on average.

One question we might ask is to what *degree* individual users tend to be producers or consumers of search knowledge. Are some searchers *net producers* of search knowledge, in the sense that they are more inclined to create search knowledge that

(a) Producers vs Consumers



(b) Peer vs Self Promotions

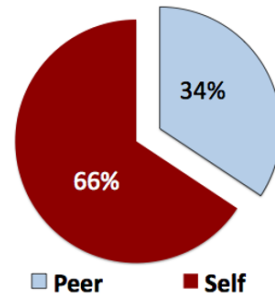


Fig. 18.12: (a) Net producers vs. consumers. (b) Promotion sources (self vs. peer).

is useful to others? Are other users *net consumers*, in the sense that they are more inclined to consume search knowledge that others have created? This data is presented in Fig. 18.12(a). To be clear, a net producer is defined as a user who has helped more other users than they themselves have been helped by, whereas a net consumer is defined as a user who has been helped by more users than they themselves have helped. The chart shows that 47% of users are net producers. Remember that, above, we noted how 50% of users have produced at least *some* search knowledge that has been consumed by some other user. It seems that the vast majority of *these* users, 94% of them in fact, are actually helping more people than they are helped by in return.

So, we have found that lots of users are helping other users, and lots of users are helped by other users. Perhaps this altruism is limited to a small number of searches? Perhaps, most of the time, at the level of individual searches, users are helping themselves? A variation on the above analysis can help shed light on this question by looking at the source of promotions that users judge to be relevant enough to select during their searches. Overall, the beta users selected more than 11,000 promotions during their searches. Some of these promotions will have been derived from the searcher's own past history; we call these *self* promotions. Others will have been derived from the search activities of other users who co-share staks with the searcher; we call these *peer* promotions. The intuition here is that the selection of self promotions corresponds to examples of HeyStaks helping users to *recover* results they have previously found, whereas the selection of promotions from peers corresponds to *discovery* tasks, where the user is benefiting from focused new content that might otherwise have been missed, or have been difficult to find; see [61, 50]. Thus Fig. 18.12(b) compares the percentage of peer and self promotions and shows that two-thirds of selected promotions are generated from the searcher's own past search activities; most of the time HeyStaks is helping searchers to recover previously found results. However, 33% of the time peer promotions are selected (and we already know that these come from many different users), helping the searcher to discover new information that others have found.

The bias towards self promotions is perhaps not surprising, especially given the habits of searchers, and especially during the early stages of stak development. The growth of most staks is initially led by a single user, usually the creator, and so inevitably most of the promotions are generated in response to the creator's own search queries. And most of these promotions will be self promotions, derived from the leader's own search activities. Many staks are not shared and so are only capable of making self promotions. As staks are shared, however, and more users join, the pool of searchers becomes more diverse. More results are added by the actions of peers and more peer promotions are generated and selected. It is an interesting task for future work to explore the evolution of a search stak and to investigate how stak content and promotions are effected as more and more users participate. Are there well-defined stages in stak evolution, for example, as self promotions give way to peer promotions? For now it is satisfying to see that even in the early stages of stak evolution, where the average stak has between 3 and 4 members, that 34% of the time members are benefiting from promotions that are derived from the activities of their peers.

18.5.4 Discussion

Compared to the first case-study, HeyStaks promotes a much more explicit form of search collaboration — search staks are explicitly created and shared by users — and the result is the formation of *micro* search communities in which small groups of searchers collaborate on particular search themes or topics. Of course this does not preclude the formation of larger groups of collaborating searchers, and it is entirely likely that certain types of search stak will evolve to become search communities in a manner that fits well with those contemplated by the previous case-study.

Once again, there are many questions left unanswered by this case-study as it provides a fertile ground for further research. For example, the potential proliferation of search staks leads to entirely new recommendation opportunities as users may benefit from suggestions about which staks to join, for example. Moreover, it may be interesting to consider the merging and/or splitting of staks in certain circumstances, allowing users to create staks by combining existing staks, for instance.

18.6 Conclusions

Web search engines are, and no doubt will continue to be, the primary tools that we will use to discover and explore online information. For all of the success of mainstream search engines like Google, the web search problem is far from being solved and research into a new generation of web search technologies is maturing. In the future it is likely that mainstream search engines will evolve to offer users

greater support when it comes to finding the right information at the right time, and recommendation technologies are set to play an important part of this future.

Already, for example, researchers are exploring how to make search engines more responsive to our particular, individual needs and preferences by combining user profiling and recommendation technologies to deliver a more personalized user experience, whether through the generation of targeted result-lists or improved query recommendation, for example. Another strand of research seeks to take advantage of the inherently collaborative nature of many web search tasks by providing searchers with new tools to foster and promote search collaboration between small groups and even large communities of searchers.

In this chapter we have provided a snapshot of these interesting areas of independent research by surveying a number of representative systems and techniques. In turn we have highlighted how these complementary approaches to collaborative and personalized web search are beginning to come together to offer users improved personalization as a side-effect of collaboration, with recommender systems playing a central role in a new type of social search service. In this regard we have presented two separate case-studies of these social search systems to show how mainstream search engines like Google may be enhanced by such approaches in practice.

In the future it is likely that mainstream search engines will evolve to accommodate many elements of these approaches, as recommendation technologies play an increasing role in web search. Where today the burden of web search is very much on the individual searcher, we believe that the introduction of recommendation technologies will provide search engines with the opportunity to be a lot more proactive as they work to anticipate, rather than respond to, a user's information needs. This in turn will lead to many new research opportunities, especially at the level of the search interface, as we look for new ways to incorporate recommendation techniques into the very fabric of web search. Indeed, already we are seeing some early examples of this as, for instance, search engines like Google and Yahoo, incorporate query recommendation techniques in to their regular search boxes. But this is just the beginning and as researchers address the challenges of profiling, privacy, and recommendation head-on, search engines will provide a unique platform for the next generation of recommendation technologies. And just as the e-commerce sites have served as an early platform for recommender systems, search engines will help to introduce a new era of recommendation technologies to a much wider audience.

Acknowledgements

This work is supported by Science Foundation Ireland under grant 07/CE/I1147.

References

1. Saleema Amershi and Meredith Ringel Morris. Cosearch: a system for co-located collaborative web search. In *Proceedings of the annual SIGCHI conference on Human factors in computing systems (CHI)*, pages 1647–1656, 2008.
2. Fabio A. Asnicar and Carlo Tasso. Ifweb: a prototype of user model-based intelligent agent for document filtering and navigation in the world wide web. In *Proceedings of the Workshop on Adaptive Systems and User Modeling on the World Wide Web at the Sixth International Conference on User Modeling*, pages 3–11, 1997.
3. Ricardo A. Baeza-Yates, Carlos A. Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Current Trends in Database Technology - EDBT 2004 Workshops*, pages 588–596, 2004.
4. Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.
5. Lori Baker-Eveleth, Suprateek Sarker, and Daniel M. Eveleth. Formation of an online community of practice: An inductive study unearthing key elements. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 254–256, 2005.
6. M. Balabanovic and Y. Shoham. FAB: Content-Based Collaborative Recommender. *Communications of the ACM*, 40(3):66–72, 1997.
7. Evelyn Balfe and Barry Smyth. Improving web search through collaborative query recommendation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 268–272, 2004.
8. D. Billsus and M. Pazzani. A Hybrid User Model for News Story Classification. In *Proceedings of the Seventh International Conference on User Modeling, UM '99*, 1999. Banff, Canada.
9. Daniel Billsus, Michael J. Pazzani, and James Chen. A learning agent for wireless news access. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*, pages 33–36, New York, NY, USA, 2000. ACM Press.
10. Oisín Boydell and Barry Smyth. Enhancing case-based, collaborative web search. In *Proceedings of International Conference on Case-Based Reasoning (ICCBR)*, pages 329–343, 2007.
11. John G. Breslin, Andreas Harth, Uldis Bojars, and Stefan Decker. Towards semantically-interlinked online communities. In *European Semantic Web Conference (ESWC)*, pages 500–514, 2005.
12. Peter Briggs and Barry Smyth. Provenance, trust, and sharing in peer-to-peer case-based web search. In *Proceedings of European Conference on Case-Based Reasoning (ECCBR)*, pages 89–103, 2008.
13. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.
14. Jay Budzik and Kristian J. Hammond. User interactions with everyday applications as context for just-in-time information access. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*, pages 44–51, New York, NY, USA, 2000. ACM.
15. R. Burke. The Wasabi Personal Shopper: A Case-Based Recommender System. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*. AAAI Press, 1999.
16. John M. Carroll and Mary Beth Rosson. Paradox of the active user. In John M. Carroll, editor, *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, chapter 5, pages 80–111. Bradford Books/MIT Press, 1987.
17. Soumen Chakrabarti, Byron Dom, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew Tomkins, David Gibson, and Jon M. Kleinberg. Mining the web's link structure. *IEEE Computer*, 32(8):60–67, 1999.
18. Pierre-Antoine Champin, Peter Briggs, Maurice Coyle, and Barry Smyth. Coping with noisy search experiences. In *Twenty-ninth SGAI International Conference on Artificial Intelligence (AI-2009)*. Springer-Verlag, 2009.

19. Huan Chang, David Cohn, and Andrew McCallum. Learning to create customized authority lists. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 127–134, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
20. Li Chen and Pearl Pu. Evaluating critiquing-based recommender agents. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2006.
21. Alvin Chin and Mark H. Chignell. Identifying active subgroups in online communities. In *CASCON*, pages 280–283, 2007.
22. Paul Alexandru Chirita, Wolfgang Nejdl, Raluca Paiu, and Christian Kohlschütter. Using odp metadata to personalize search. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 178–185, New York, NY, USA, 2005. ACM Press.
23. Paul-Alexandru Chirita, Daniel Olmedilla, and Wolfgang Nejdl. Pros: A personalized ranking platform for web search. In Paul De Bra and Wolfgang Nejdl, editors, *Proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH)*, volume 3137 of *Lecture Notes in Computer Science*, pages 34–43. Springer, 2004.
24. Maurice Coyle and Barry Smyth. Information recovery and discovery in collaborative web search. In *Proceedings of the European Conference on Information retrieval (ECIR)*, pages 356–367, 2007.
25. Maurice Coyle and Barry Smyth. Supporting intelligent web search. *ACM Trans. Internet Techn.*, 7(4), 2007.
26. W. Bruce Croft, Stephen Cronen-Townsend, and Victor Larvrenko. Relevance feedback and personalization: A language modeling perspective. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.
27. B.J. Dahlen, J.A. Konstan, J.L. Herlocker, N. Good, A. Borchers, and J. Riedl. Jump-starting movieLens: User benefits of starting a collaborative filtering system with "dead-data". In . University of Minnesota TR 98-017, 1998.
28. B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA, 1991.
29. R. Dolin, D. Agrawal, A. El Abbadi, and L. Dillon. Pharos: a scalable distributed architecture for locating heterogeneous information sources. In *CIKM '97: Proceedings of the sixth international conference on Information and knowledge management*, pages 348–355, New York, NY, USA, 1997. ACM.
30. Susan Feldman and Chris Sherman. The High Cost of Not Finding Information. In (*IDC White Paper*). IDC Group, 2000.
31. L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: the concept revisited. In *WWW '01: Proceedings of the 10th International Conference on the World Wide Web*, pages 406–414. ACM Press, 2001.
32. C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. Citeseer: an automatic citation indexing system. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, pages 89–98, New York, NY, USA, 1998. ACM.
33. Laura A. Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in www search. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 478–479, New York, NY, USA, 2004. ACM.
34. Luis Gravano, Héctor García-Molina, and Anthony Tomasic. Gloss: text-source discovery over the internet. *ACM Trans. Database Syst.*, 24(2):229–264, 1999.
35. Anthony Jameson and Barry Smyth. Recommendation to groups. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, pages 596–627. Springer-Verlag, 2007.
36. Bernard J. Jansen and Amanda Spink. An analysis of web searching by european alltheweb.com users. *Inf. Process. Manage.*, 41(2):361–381, 2005.
37. Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.

38. Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 271–279, New York, NY, USA, 2003. ACM.
39. Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.
40. Jon M. Kleinberg. Hubs, authorities, and communities. *ACM Comput. Surv.*, 31(4):5, 1999.
41. J.A. Konstan, B.N Miller, D. Maltz, J.L. Herlocker, L.R. Gorgan, and J. Riedl. GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.
42. Yehuda Koren. Tutorial on recent progress in collaborative filtering. In *Proceedings of the International Conference on Recommender Systems (RecSys)*, pages 333–334, 2008.
43. G. Koutrika and Y. Ioannidis. A unified user-profile framework for query disambiguation and personalization. In *Proc. of the Workshop on New Technologies for Personalized Information Access*, pages 44–53, 2005.
44. A. Kruger, C. L. Giles, F. M. Coetzee, E. Glover, G. W. Flake, S. Lawrence, and C. Omlin. Deadliner: building a new niche search engine. In *CIKM '00: Proceedings of the ninth international conference on Information and knowledge management*, pages 272–281, New York, NY, USA, 2000. ACM.
45. S. Lawrence and C. Lee Giles. Accessibility of Information on the Web. *Nature*, 400(6740):107–109, 1999.
46. Greg Linden, Brent Smith, and Jeremy York. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Distributed Systems Online*, 4(1), 2003.
47. Fang Liu, Clement Yu, and Weiyi Meng. Personalized web search by mapping user queries to categories. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 558–565, New York, NY, USA, 2002. ACM Press.
48. Zhongming Ma, Gautam Pant, and Olivia R. Liu Sheng. Interest-based personalized search. *ACM Trans. Inf. Syst.*, 25(1):5, 2007.
49. Christos Makris, Yannis Panagis, Evangelos Sakkopoulos, and Athanasios Tsakalidis. Category ranking for personalized search. *Data Knowl. Eng.*, 60(1):109–125, 2007.
50. Gary Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.
51. Kevin McCarthy, James Reilly, Lorraine McGinty, and Barry Smyth. Experiments in dynamic critiquing. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, pages 175–182, 2005.
52. Kevin McCarthy, Maria Salamó, Lorcan Coyle, Lorraine McGinty, Barry Smyth, and Paddy Nixon. Cats: A synchronous approach to collaborative group recommendation. In *Proceedings of the International FLAIRS Conference*, pages 86–91, 2006.
53. L. McGinty and B. Smyth. Comparison-Based Recommendation. In Susan Craw, editor, *Proceedings of the Sixth European Conference on Case-Based Reasoning (ECCBR 2002)*, pages 575–589. Springer, 2002. Aberdeen, Scotland.
54. Ryan J. Meuth, Paul Robinette, and Donald C. Wunsch. Computational intelligence meets the netflix prize. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 686–691, 2008.
55. Alessandro Micarelli and Filippo Sciarrone. Anatomy and empirical evaluation of an adaptive web-based information filtering system. *User Modeling and User-Adapted Interaction*, 14(2-3):159–200, 2004.
56. M. Mitra, A. Singhal, and C. Buckley. Improving Automatic Query Expansion. In *Proceedings of ACM SIGIR*, pages 206–214. ACM Press, 1998.
57. Meredith Ringel Morris. A survey of collaborative web search practices. In *Proceedings of the annual SIGCHI conference on Human factors in computing systems (CHI)*, pages 1657–1660, 2008.
58. Meredith Ringel Morris and Eric Horvitz. S³: Storable, shareable search. In *INTERACT (1)*, pages 120–123, 2007.

59. Meredith Ringel Morris and Eric Horvitz. Searchtogether: an interface for collaborative web search. In *UIST*, pages 3–12, 2007.
60. Sridhar P. Nerur, Riyaz Sikora, George Mangalaraj, and Venugopal Balijepally. Assessing the relative influence of journals in a citation network. *Commun. ACM*, 48(11):71–74, 2005.
61. Vicki L. O’Day and Robin Jeffries. Orienteering in an information landscape: how information seekers get from here to there. In *CHI ’93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 438–445, New York, NY, USA, 1993. ACM Press.
62. John O’Donovan, Vesile Evrim, and Barry Smyth. Personalizing trust in online auctions. In *Proceedings of the European Starting AI Researcher Symposium (STAIRS)*, Trento, Italy, 2006.
63. John O’Donovan and Barry Smyth. Eliciting trust values from recommendation errors. In *Proceedings of the International FLAIRS Conference*, pages 289–294, 2005.
64. John O’Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, pages 167–174, 2005.
65. John O’Donovan and Barry Smyth. Trust no one: Evaluating trust-based filtering for recommenders. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1663–1665, 2005.
66. John O’Donovan and Barry Smyth. Is trust robust?: an analysis of trust-based recommendation. In *Intelligent User Interfaces*, pages 101–108, 2006.
67. John O’Donovan and Barry Smyth. Mining trust values from recommendation errors. *International Journal on Artificial Intelligence Tools*, 15(6):945–962, 2006.
68. Jeremy Pickens, Gene Golovchinsky, Chirag Shah, Pernilla Qvarfordt, and Maribeth Back. Algorithmic mediation for collaborative exploratory search. In *SIGIR*, pages 315–322, 2008.
69. Alexander Pretschner and Susan Gauch. Ontology based personalized search. In *ICTAI ’99: Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, page 391, Washington, DC, USA, 1999. IEEE Computer Society.
70. Madhu C. Reddy and Paul Dourish. A finger on the pulse: temporal rhythms and information seeking in medical work. In *CSCW*, pages 344–353, 2002.
71. Madhu C. Reddy, Paul Dourish, and Wanda Pratt. Coordinating heterogeneous work: Information and representation in medical care. In *ECSCW*, pages 239–258, 2001.
72. Madhu C. Reddy and Bernard J. Jansen. A model for understanding collaborative information behavior in context: A study of two healthcare teams. *Inf. Process. Manage.*, 44(1):256–273, 2008.
73. Madhu C. Reddy and Patricia Ruma Spence. Collaborative information seeking: A field study of a multidisciplinary patient care team. *Inf. Process. Manage.*, 44(1):242–255, 2008.
74. J. Reilly, K. McCarthy, L. McGinty, and B. Smyth. Dynamic Critiquing. In Peter Funk and Pedro A. Gonzalez Calero, editors, *Proceedings of the 7th European Conference on Case-Based Reasoning*, pages 763–777. Springer-Verlag, 2004.
75. James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Incremental critiquing. *Knowl.-Based Syst.*, 18(4-5):143–151, 2005.
76. James Reilly, Barry Smyth, Lorraine McGinty, and Kevin McCarthy. Critiquing with confidence. In *Proceedings of International Conference on Case-Based Reasoning (ICCBR)*, pages 436–450, 2005.
77. Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, 1997.
78. J. Rocchio. *Relevance Feedback in Information Retrieval*. G. Salton (editor), The SMART Retrieval System: Experiments in Automatic Document Processing. Prentice–Hall, Inc., Englewood Cliffs, NJ, 1971.
79. U. Rohini and Vasudeva Varma. A novel approach for re-ranking of search results using collaborative filtering. In *Proceedings of the International Conference on Computing: Theory and Applications*, volume 00, pages 491–496, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
80. Lior Rokach, Genetic algorithm-based feature set partitioning for classification problems. *Pattern Recognition*, 41(5):1676–1700, 2008.

81. Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the International World-Wide Web Conference*, pages 377–386, 2006.
82. Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. In *Readings in information retrieval*, pages 355–364. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
83. J. Ben Schafer, Joseph Konstan, and John Riedi. Recommender systems in e-commerce. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, New York, NY, USA, 1999. ACM Press.
84. U. Shardanand and P. Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '95)*, pages 210–217. ACM Press, 1995. New York, USA.
85. U. Shardanand and P. Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 210–217, 1995.
86. Xuehua Shen, Bin Tan, and ChengXiang Zhai. Implicit User Modeling for Personalized Search. In *Proceedings of the Fourteenth ACM Conference on Information and Knowledge Management (CIKM 05)*, 2005.
87. Alan F. Smeaton, Colum Foley, Daragh Byrne, and Gareth J. F. Jones. ibingo mobile collaborative search. In *CIVR*, pages 547–548, 2008.
88. Alan F. Smeaton, Hyowon Lee, Colum Foley, and Sinéad McGivney. Collaborative video searching on a tabletop. *Multimedia Syst.*, 12(4-5):375–391, 2007.
89. Barry Smyth. Case-based recommendation. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, pages 342–376. Springer-Verlag, 2007.
90. Barry Smyth. A community-based approach to personalizing web search. *IEEE Computer*, 40(8):42–50, 2007.
91. Barry Smyth, Evelyn Balfe, Oisín Boydell, Keith Bradley, Peter Briggs, Maurice Coyle, and Jill Freyne. A Live-user Evaluation of Collaborative Web Search. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*, pages 1419–1424. Morgan Kaufmann, 2005. Edinburgh, Scotland.
92. Barry Smyth, Evelyn Balfe, Jill Freyne, Peter Briggs, Maurice Coyle, and Oisín Boydell. Exploiting query repetition and regularity in an adaptive community-based web search engine. *User Model. User-Adapt. Interact.*, 14(5):383–423, 2004.
93. Barry Smyth, Peter Briggs, Maurice Coyle, and Michael P O'Mahony. A case-based perspective on social web search. In *Proceedings of International Conference on Case-Based Reasoning (ICCBR)*, 2009.
94. Barry Smyth, Peter Briggs, Maurice Coyle, and Michael P. O'Mahony. Google. shared! a case-study in social search. In *Proceedings of the International Conference on User Modeling, Adaptation and Personalization (UMAP)*, pages 494–508. Springer-Verlag, 2009.
95. Barry Smyth and Pierre-Antoine Champin. The experience web: A case-based reasoning perspective. In *Workshop on Grand Challenges for Reasoning from Experiences (IJCAI 2009)*, 2009.
96. Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. In *Document retrieval systems*, pages 132–142. Taylor Graham Publishing, London, UK, UK, 1988.
97. Micro Speretta and Susan Gauch. Personalized search based on user search histories. In *WI '05: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 622–628, Washington, DC, USA, 2005. IEEE Computer Society.
98. Amanda Spink, Judy Bateman, and Major Bernard Jansen. Searching heterogeneous collections on the web: behaviour of excite users. *Information Research: An Electronic Journal*, 4(2), 1998.
99. Amanda Spink and Bernard J. Jansen. A study of web search trends. *Webology*, 1(2):4, 2004.

100. Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. Searching the Web: the Public and their Queries. *Journal of the American Society for Information Science*, 52(3):226–234, 2001.
101. Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 675–684, New York, NY, USA, 2004. ACM Press.
102. Jian-Tao Sun, Hua-Jun Zeng, Huan Liu, Yuchang Lu, and Zheng Chen. Cubesvd: a novel approach to personalized web search. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 382–390, New York, NY, USA, 2005. ACM Press.
103. Bin Tan, Xuehua Shen, and ChengXiang Zhai. Mining long-term search history to improve search accuracy. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 718–723, New York, NY, USA, 2006. ACM Press.
104. C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.
105. Xin Wang and Ata Kabán. State aggregation in higher order markov chains for finding online communities. In *IDEAL*, pages 1023–1030, 2006.
106. Xin Wang and Ata Kabán. A dynamic bibliometric model for identifying online communities. *Data Min. Knowl. Discov.*, 16(1):67–107, 2008.
107. Yiming Yang and Christopher G. Chute. An example-based mapping method for text categorization and retrieval. *ACM Trans. Inf. Syst.*, 12(3):252–277, 1994.
108. Baoyao Zhou, Siu Cheung Hui, and Alvis C. M. Fong. An effective approach for periodic web personalization. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 284–292, Washington, DC, USA, 2006. IEEE Computer Society.

Chapter 19

Social Tagging Recommender Systems

Leandro Balby Marinho, Alexandros Nanopoulos, Lars Schmidt-Thieme, Robert Jäschke, Andreas Hotho, Gerd Stumme and Panagiotis Symeonidis

Abstract The new generation of Web applications known as (STS) is successfully established and poised for continued growth. STS are open and inherently social; features that have been proven to encourage participation. But while STS bring new opportunities, they revive old problems, such as information overload. Recommender Systems are well known applications for increasing the level of relevant content over the “noise” that continuously grows as more and more content becomes available online. In STS however, we face new challenges. Users are interested in finding not only content, but also tags and even other users. Moreover, while traditional recommender systems usually operate over 2-way data arrays, STS data is represented as a third-order tensor or a hypergraph with hyperedges denoting (user, resource, tag) triples. In this chapter, we survey the most recent and state-of-the-art work about a whole new generation of recommender systems built to serve STS. We describe (a) novel facets of recommenders for STS, such as user, resource, and tag recommenders, (b) new approaches and algorithms for dealing with the ternary nature of STS data, and (c) recommender systems deployed in real world STS. Moreover, a concise comparison between existing works is presented, through which we identify and point out new research directions.

Leandro Balby Marinho · Alexandros Nanopoulos · Lars Schmidt-Thieme
Information Systems and Machine Learning Lab (ISMLL), University of Hildesheim, Marienburger Platz 22, 31141 Hildesheim, Germany, <http://www.ismll.uni-hildesheim.de>, e-mail: {marinho,nanopoulos,schmidt-thieme}@ismll.uni-hildesheim.de

Robert Jäschke · Andreas Hotho · Gerd Stumme
Knowledge & Data Engineering Group (KDE), University of Kassel, Wilhelmshöher Allee 73, 34121 Kassel, Germany, <http://www.kde.cs.uni-kassel.de>, e-mail: {jaeschke,hotho,stumme}@cs.uni-kassel.de

Panagiotis Symeonidis
Department of Informatics, Aristotle University, 54124 Thessaloniki, Greece, e-mail: symeon@csd.auth.gr

19.1 Introduction

With the advent of affordable domestic high-speed communication facilities, inexpensive digitization devices, and the open access nature of the Web, a new and exciting family of Web applications known as Web 2.0 has been born. The underlying idea is to decentralize and cheapen content creation, thus leading the Web into a more open, connected, and democratic environment. In this chapter we will focus on a particular family of Web 2.0 applications known as Social Tagging Systems (STS for short). STS assign a major role to the ordinary user, who is not only allowed to publish and edit resources, but also and more importantly, to create and share lightweight metadata in the form of freely chosen keywords called *tags*. The exposure of users to both tags and resources creates a fundamental trigger for communication and sharing, thus lowering the barriers to cooperation and contributing to the creation of collaborative lightweight knowledge structures known as *folksonomies*¹. Some notable examples of STS are sites like Delicious², BibSonomy³, and Last.fm⁴, where Delicious allows the sharing of bookmarks, BibSonomy the sharing of bookmarks and lists of literature, and Last.fm the sharing of music. These systems are characterized by being easy to use and free to anyone willing to participate. Once a user is logged in, he can add a resource to the system, and assign arbitrary tags to it.

If on the one hand this new family of applications brings new opportunities, it revives old problems on the other, namely the problem of information overload. Millions of individual users and independent providers are flooding STS with content and tags in an uncontrolled way, thereby lowering the potential for content retrieval and information sharing. One of the most successful approaches for increasing the level of relevant content over the “noise” that continuously grows as more and more content becomes available online lies on Recommender Systems (RS for short). In STS however, we face several new challenges. Users are interested in finding not only content, but also tags, and even other users. Moreover, while traditional RS usually operate over 2-way data arrays, folksonomy data is represented as a third-order tensor or a hypergraph with hyperedges denoting (user, resource, tag) triples. Furthermore, while there is an extensive literature for rating prediction based on explicit user feedback, i.e., a numerical value denoting the degree of preference of a user for a given item, in folksonomies there are usually no ratings. Thus, before arguing why not to simply use an old solution to a recurrent problem, we need to investigate to which extent the traditional RS paradigm and approaches apply to STS.

Social tagging recommender systems is a young research area that has attracted significant attention recently, which is expressed by the increasing number of publications (e.g., [15, 11, 37, 35, 31]) and is poised for continued growth. Furthermore,

¹ The term folksonomy refers to a blend of the two words folk and taxonomy, i.e., a collaborative classification system created and maintained by ordinary users.

² <http://delicious.com/>

³ <http://www.bibsonomy.org/>

⁴ <http://www.last.fm/>

real and large scale STS, such as Delicious, BibSonomy, and Last.fm, for example, already offer some recommender services to their users, which implies an increasing commercial interest in the area. In this chapter we survey in a concise manner, the most recent and state-of-the-art work about a whole new generation of RS built to serve STS. We describe: (a) novel facets of RS for STS, such as user, resource, and tag recommenders, (b) the challenges for deploying RS in real-world STS, (c) new approaches and algorithms for dealing with the inherent ternary relational data of folksonomies, and (d) approaches for tag acquisition. Emphasis is given on presenting a concise comparison between existing works, through which we identify and point out new research directions.

The chapter is structured as follows. In Section 19.2 we characterize the data structure of folksonomies and point out some of the differences between the traditional RS paradigm and social tagging RS. In Section 19.3 we discuss the challenges of deploying RS in real world STS and present the BibSonomy system as a study case. Section 19.4 presents several families of social tagging RS, such as graph/content-based algorithms for recommending users, resources or tags. Section 19.5 provides comparisons and discussions about the algorithms presented in Section 19.4; and finally Section 19.6 closes the chapter pointing out new directions of research in this area.

19.2 Social Tagging Recommenders Systems

Folksonomies are the underlying structures of STS and result from the practice of collaboratively creating tags to annotate and categorize content. Tags, in general, are a way of grouping content by category to make them easy to view by topic. This is a grassroots approach to organize a site and help users find content they are interested in. Note that with the introduction of tags, the usual binary relation between users and resources, which is largely exploited by traditional RS, turns into a ternary relation between users, resources, and tags.

Since tags are voluntarily and freely provided by ordinary users, problems such as unwillingness to tag and diverging vocabulary can easily arise. As we will see in the course of this chapter, a possible way to address these problems is through tag RS. Tags also represent additional and personalized information about resources, which if properly exploited, can eventually boost the performance of resource RS. But before we delve into how RS can deal and benefit from the additional information provided by tags, we need to formally define folksonomies and its data structures, elaborate on the differences between traditional RS and social tagging RS, and the challenges involved in deploying RS in real world STS; topics which are covered in the following sections.

19.2.1 Folksonomy

Formally, a *folksonomy* is a tuple $\mathbb{F} := (U, T, R, Y)$ where

- U , T , and R are non-empty finite sets, whose elements are called *users*, *tags*, and *resources*, resp., and
- Y is a ternary relation between them, i. e., $Y \subseteq U \times T \times R$, whose elements are called tag assignments.⁵

Users are typically described by their user ID, and tags may be arbitrary strings. What is considered a resource depends on the type of system. For instance, in Delicious, the resources are URLs, in BibSonomy URLs or publication references, and in Last.fm, the resources can be artists, song tracks or albums.

Folksonomy data can be represented in different ways, and as we will see in Section 19.4, each representation can lead to different recommendation algorithms.

Folksonomies as Tensors The set of triples in Y can be represented as third-order *tensors* (3-dimensional arrays) $\mathcal{A} = (a_{u,t,r}) \in \mathbb{R}^{|U| \times |T| \times |R|}$. There are different ways to represent Y as a tensor (see left-hand side of Figure 19.1). Symeonidis et al. [35], for example, proposed to interpret Y as a sparse tensor in which 1 indicates positive feedback and 0 missing values:

$$a_{u,t,r} = \begin{cases} 1, & (u,t,r) \in Y \\ 0, & \text{else} \end{cases}$$

Rendle *et al.* [26], on the other hand, distinguish between positive/negative examples and missing values in order to learn personalized ranking of tags (see Section 19.4). The idea is that positive and negative examples are only generated from observed tag assignments. Observed tag assignments are interpreted as positive feedback, whereas the non observed tag assignments of an already tagged resource are negative evidences. All other entries are assumed to be missing values (see right-hand side of Figure 19.1).

Note that in folksonomies, differently from typical RS, there are usually no numerical ratings indicating the explicit preference of a user for a given resource/tag.

Folksonomies as Hypergraphs An equivalent, but maybe more intuitive representation of a folksonomy, is a tripartite (undirected) hypergraph $G := (V, E)$, where $V := U \cup T \cup R$ is the set of nodes, and $E := \{\{u,t,r\} \mid (u,t,r) \in Y\}$ is the set of hyperedges (see Figure 19.2).

⁵ In the original definition [12], it is introduced additionally a subtag/supertag relation, which we omit here. The version used here is known in Formal Concept Analysis [7] as a *triadic context* [21, 34].

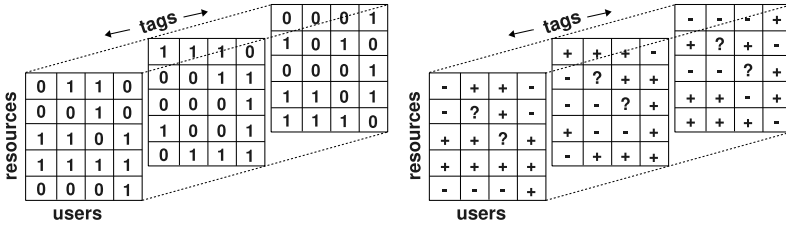


Fig. 19.1: *Left* [35]: 0/1 sparse tensor representation where positive feedback is interpreted as 1 and the remaining data as 0. *Right* [26]: Non observed tag assignments for a given already tagged resource are negative examples. All other entries are missing values.

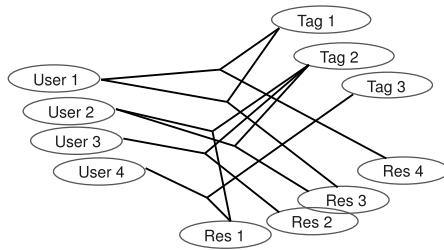


Fig. 19.2: Tripartite undirected hypergraph representation of a folksonomy.

19.2.2 The Traditional Recommender Systems Paradigm

Recommender systems are software applications that aim at predicting the user interest for a particular resource based on a collection of user profiles, e.g., the user’s history of purchase/resources’ ratings, click-stream data, demographic information, and so forth. Usually RS predict ratings of resources or suggest a list of new resources that the user hopefully will like the most. Traditionally, for m users and n resources, the user profiles are represented in a sparse user-resource matrix $\mathbf{X} \in \mathbb{R}^{m \times n} \cup \{.\}$, where $\{.\}$ denote missing values. The matrix can be decomposed into row vectors:

$$\mathbf{X} := [\mathbf{x}_1, \dots, \mathbf{x}_m]^T \text{ with } \mathbf{x}_u := [x_{u,1}, \dots, x_{u,n}], \text{ for } u := 1, \dots, m,$$

where $x_{u,r}$ indicates that user u rated resource r by $x_{u,r} \in \mathbb{R}$. Each row vector \mathbf{x}_u corresponds thus to a user profile representing the resource’s ratings of a particular user. This decomposition usually leads to algorithms that leverage user-user similarities, such as the well known user-based collaborative filtering (CF) [27]. The matrix can alternatively be represented by its column vectors:

$$\mathbf{X} := [\mathbf{x}_1, \dots, \mathbf{x}_n] \text{ with } \mathbf{x}_r := [x_{1,r}, \dots, x_{m,r}]^T, \text{ for } r := 1, \dots, n,$$

in which each column vector \mathbf{x}_r corresponds to a specific resource's ratings by all m users. This representation usually leverages item-item similarities and leads to item-based CF algorithms [3]. For a survey on neighborhood-based recommendation methods, such as CF, see Chapter 4.

Note that because of the ternary relational nature of folksonomies, traditional RS cannot be applied directly. Therefore, in order to develop RS for folksonomies, one needs to either (i) reduce the ternary relation Y to a lower dimensional space (usually second-order tensors) where traditional RS can be applied, or develop new algorithms that operate over third-order tensors or tripartite undirected hypergraphs. Note that if one follows (i), care must be taken during the dimensionality reduction since important information can be discarded, which can lower the overall accuracy of the recommendations. In Section 19.4 we present and discuss both families of algorithms.

19.2.3 Multi-mode Recommendations

Differently from the traditional RS paradigm, where one is usually concerned only with rating prediction or resource recommendations, STS users may be interested in finding resources/tags, or even other users, and therefore recommendations can be provided for any of these entity types.

The recommendation of tags is used in several systems, like Delicious and BibSonomy, for example. It usually involves the recommendation of tags to users, based on the tags other users have provided for the same resources. Tag recommendations can expose different facets of an information item and relieve users from the obnoxious task of coming up with a good set of tags. Moreover, tag recommendation can reduce the problem of tag sparsity, which results from the unwillingness of users to tag. Figure 19.5 illustrates tag recommendations in BibSonomy.

It is important to note that differently from traditional RS, where there is usually no repeat-buying, i.e., the user usually does not buy the same book, movie, CD, etc. twice, re-occurring tags are a common feature of STS. A tag that has already been used to annotate a resource can be reused to annotate other different resources. This means that while traditional RS usually only recommend items that the user has not yet bought or rated, tag recommenders can eventually recommend tags that the user has already used for other resources.

The recommendation of resources is largely used in e-commerce and advertising, like in Amazon for example. With the actual trend towards STS, the current resource recommendation services will also be able exploit the tags to boost the recommendation quality, for example, by recommending resources to users based on the tags they have in common with other similar users. The movie recommendation website movielens⁶, where users rate the movies they like and receive recommendations about other movies in which they might be interested, is a notable example.

⁶ <http://www.movielens.org>

It started as a traditional recommender service operating over the typical user-rating binary matrix, and just recently added social tagging features, whereby new tag-aware algorithms are being developed and deployed [30].

A third type of recommendation concerns recommending interesting users to a target user, which can help to connect people with common interests and encourage them to contribute and share more content. With the term *interesting* users, we mean those users who have similar profile to the target user. If a set of tags is frequently used by many users, for example, then these users implicitly form a group of users with common interests, even though they may not have any physical or online connections. The tags represent the common interests to this user group.

Each mode of recommendation, i.e., tag, resource, or user, is useful, depending of course on the context of the particular application. Algorithms that are able to provide integrated multi-mode recommendations are very appealing, as one can spare the effort of implementing and maintaining several mode-specific recommender systems.

19.3 Real World Social Tagging Recommender Systems

19.3.1 What are the Challenges?

For a recommender system to be successful in a real world application, it must approach several challenges. First, the provided recommendations must match the situation, i.e., tags should describe the annotated resource, products should awake the interest of the user, suggested resources should be interesting and relevant. Second, the suggestions should be traceable such that one easily understands why he got the items suggested. Third, they must be delivered timely without delay and they must be easy to access (i.e., by allowing the user to click on them or to use tab-completion when entering tags). Furthermore, the system must ensure that recommendations do not impede the normal usage of the system.

In this section we focus on tag recommendations as example of recommenders in STS. Most STS contain a tag recommender which suggests tags to the user when she is annotating a resource. Recommending tags can serve various purposes, such as: increasing the chances of getting a resource annotated, reminding a user what a resource is about, and consolidating the vocabulary across the users. Furthermore, as Sood et al. [33] point out, tag recommendations “fundamentally change the tagging process from generation to recognition” which requires less cognitive effort and time.

More formally, given a user u and a resource r , the task of a tag recommender is to predict the tags $\text{tags}(u, r)$ the user will assign to the resource. We will depict the (ordered!) set of recommended tags by $\hat{T}(u, r)$. Although we do not take the order of tags as the user entered them into account, the order of tags as given by the recommender plays an important role for the evaluation.

REST web services

Good intro to the REST "architecture"

to [web service tutorial guidelines api rest](#) by [hotho](#) and [3 other people](#) on 2006-04-04 16:11:47 [copy](#)

Fig. 19.3: detail showing a single bookmark post

19.3.2 BibSonomy as Study Case

19.3.2.1 System Description

BibSonomy started as a students project at the Knowledge and Data Engineering Group of the University of Kassel⁷ in spring 2005. The goal was to implement a system for organizing BIB_TE_X [25] entries in a way similar to bookmarks in Delicious – which was at that time becoming more and more popular. BIB_TE_X is a popular literature management system for L^AT_EX [20], which many researchers use for writing scientific papers. After integrating bookmarks as a second type of resource into the system and upon the progress made, BibSonomy was opened for public access at the end of 2005 – first announced to colleagues only, later in 2006 to the public.

A detailed view of one bookmark post in BibSonomy can be seen in Figure 19.3. The first line shows in bold the title of the bookmark which has the URL of the bookmark as underlying hyperlink. The second line shows an optional description the user can assign to every post. The last two lines belong together and show detailed information: first, all the tags the user has assigned to this post (*web*, *service*, *tutorial*, *guidelines* and *api*), second, the user name of that user (*hotho*) followed by a note, how many users tagged that specific resource. These parts have underlying hyperlinks, leading to the corresponding tag pages of the user, the users page and a page showing all four posts (i. e., the one of user *hotho* and those of the three other people) of this resource. The structure of a publication post is very similar, as seen in Figure 19.4.

19.3.2.2 Recommendations in BibSonomy

To support the user during the tagging process and to facilitate the tagging, BibSonomy includes a tag recommender (see Figure 19.5). When a user finds an interesting web page (or publication) and posts it to BibSonomy, the system offers up to ten recommended tags on the posting page.

⁷ <http://www.kde.cs.uni-kassel.de/>

Semantic Network Analysis of Ontologies

Bettina Hoser and Andreas Hotho and Robert Jäschke and Christoph Schmitz and Gerd Stumme. *Proceedings of the 3rd European Semantic Web Conference* (accepted for publication) (2006)

to web 2006 social ontology myown semantic analysis network sna by hotho and 1 other person on 2006-04-06 21:32:23 pick copy URL BibTeX

Fig. 19.4: detail showing a single publication post

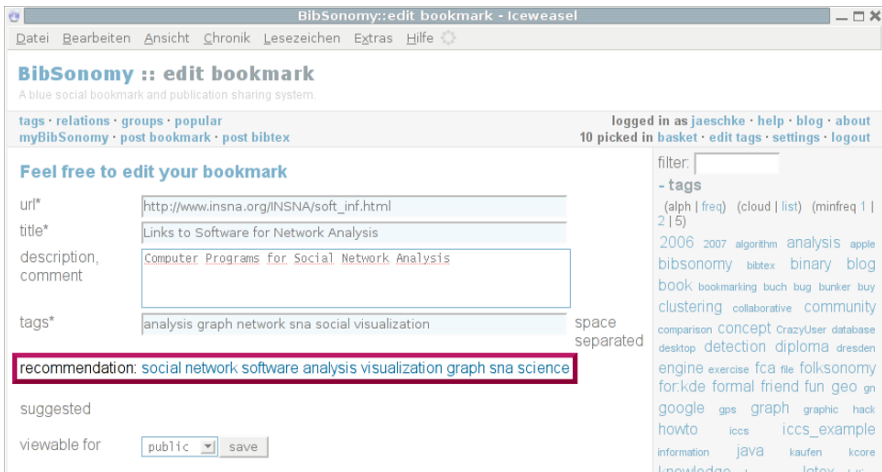


Fig. 19.5: Tag recommendations in BibSonomy during annotation of a bookmark.

19.3.2.3 Technological and Infrastructure Requirements

Implementing a recommendation service for BibSonomy required to tackle several problems, some of them we describe here.

First, having enough data available for recommendation algorithms to produce helpful recommendations is an important requirement one must address already in the design phase. The recommender needs access to the systems database and to what the user is currently posting (which could be accomplished, e.g., by (re)-loading recommendations using techniques like AJAX). Further data – like the full text of documents – could be supplied to tackle the cold-start problem (e.g., for content-based recommenders). The system must be able to handle large amounts of data, to quickly select relevant subsets and provide methods for preprocessing.

The available hardware and expected amount of data limits the choice of recommendation algorithms which can be used. Although some methods allow (partial) precomputation of recommendations, this needs extra memory and might not yield the same good results as online computation. Both hardware and network infrastructure must ensure short response times to deliver the recommendations to the user without too much delay. Together with a simple and non-intrusive user interface this ensures usability.

Further aspects which should be taken into account include implementation of logging of user events (e.g., clicking, key presses, etc.) to allow for efficient evaluation of the used recommendation methods in an online setting. Together with a live evaluation this also allows to tune the result selection strategies to dynamically choose the (currently) best recommendation algorithm for the user or resource at hand. The multiplexing of several available algorithms together with the simple inclusion of external recommendation services (by providing an open recommendation interface) is one of the recent developments in BibSonomy.

19.3.3 Tag Acquisition

The quality of tags can directly affect the recommendation performance of social tagging RS. Although folksonomies represent the “wisdom of crowds”, social tagging can present problems, such as tag sparsity (users tend to provide a constrained number of tags), polysemy (tags are subject to multiple interpretations), or tag idiosyncrasy (tags used for personal organization like “to read”, for example). All these problems can harm the quality of recommendations. For this reason, we consider alternative ways of acquiring tags. This will help us to better characterize the advantages and disadvantages of the social tagging process. We then examine the following tag acquisition methods:

- **Expert Tagging:** This approach usually relies on a small number of domain experts, who annotate resources using, mainly, structured vocabularies. Experts provide tags that are objective and cover multiple aspects. Pandora⁸ is a notable example of a system that uses experts for tagging music resources. The main advantage of using experts is the resulting well agreed tag vocabulary. This comes, of course, at the cost of manual work, which is both time consuming and expensive.
- **Tagging based on annotation games:** Games with a purpose (GWAP) [39], like the ESPGame⁹, is a breakthrough idea to use a game to employ humans for the purpose of annotation. Two players observe simultaneously the same image and are asked to enter tags until they both enter the same tag. Following the success of ESPGame, several others appeared (e.g., ListenGame¹⁰) in the

⁸ <http://www.pandora.com/>

⁹ <http://www.gwap.com/gwap/gamesPreview/>

¹⁰ <http://www.listengame.org/>

domain of music. Like social tagging, games exploit the “computational power of humans”. By partnering two or more people, the resulting set of tags has the potential of being highly accurate. The problem with games is that players, opting for higher scores, may sacrifice the quality of tags. For example, they may enter more general tags in favor of more specific, just to increase the probability of match.

- **Content-based Tagging:** Resources like URLs, sound tracks, etc., contain a rich content. By crawling associated information from the Web and by converting this data into a suitable representation, tags can be generated using data mining algorithms. In the tag recommendation task of the ECML PKDD Discovery Challenge 2008¹¹, for example, some of the tags to be predicted in the test set never appeared in the training set, which forced the participants (e.g., [23]) to use the textual content of the resources to come up with new tags. In the music domain, this approach is called *auto-tagging* and has been proposed to avoid the cold-start problem [5]. The advantage of content-based tags is that no humans must be directly involved during the tagging process. The disadvantages are that these tags can be noisy, their computation is intensive, and users are forced to agree with the tags generated by the algorithms.

Compared to the alternative methods, social tagging has the advantage of producing large-scale tag collections. The quality of tags generally improves with a large number of taggers. Nevertheless, social tagging is prone to the cold-start problem, as new resources are seldom tagged. In Table 19.1 we summarize the main advantages and disadvantages of the described approaches.

Table 19.1: Characterization of tag collection methods.

Method	Advantages	Disadvantages
Social tagging	scalable, “wisdom of crowds”	idiosyncrasy, polysemy, cold-start
Experts	accurate tags	expensive process, non scalable
Games	“wisdom of crowds”, potentially scalable	cold-start, manipulation prone
Mined tags	automation, avoids cold-start	noisy, computationally intensive

Although social tagging is prone to idiosyncrasy, sparsity, and cold-start problems, the quality of tags generally improves with a large number of taggers. Furthermore, social tagging systems (as well as annotation games) represent a human computation paradigm with enormous potential to address problems that content-based mechanisms for tag acquisition cannot yet tackle on their own. But unlike computers, humans require some incentive to become part of the “collective computation”, which can be provided, for example, through the recommendation of tags.

¹¹ <http://www.kde.cs.uni-kassel.de/ws/rsdc08/>.

19.4 Recommendation Algorithms for Social Tagging Systems

As we pointed out in Section 19.2, there are some particularities in folksonomies that one needs to take into account before designing RS, such as:

- Folksonomy data is represented as tensors or tripartite undirected hypergraphs and thus one needs to either transform the data in order to apply traditional recommender algorithms or extend the existent methods to operate over tensors or hypergraphs.
- Folksonomy users might be interested in multi-mode recommendations, so algorithms that serve all modes with minor or no changes during mode switching are ideally desired.
- Folksonomies allow multi-media resources, and thereby content-based algorithms should be able to efficiently incorporate content information in the folksonomy data structure.

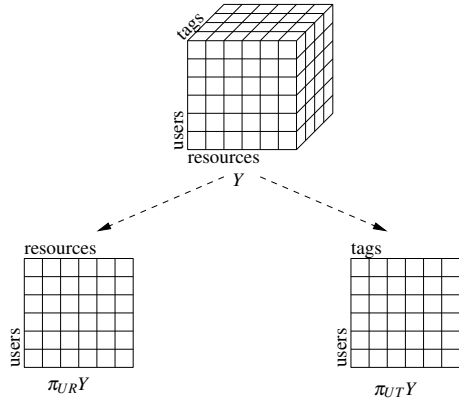
In this section we survey some of the most recent and prominent methods about social tagging RS, showing and discussing how they address the aforementioned issues.

19.4.1 Collaborative Filtering

Collaborative Filtering is one of the most used and successfully applied methods for personalized RS, for which a large and continuously active literature exists (see Chapters 4 and 5). Basically, it is an algorithm for matching people with similar interests for the purpose of making recommendations. As pointed out in Section 19.2.2, traditional recommender systems typically operate on second-order tensors representing a binary relation between users and resources. Thus, because of the ternary relational nature of folksonomies, traditional CF cannot be applied directly, unless the ternary relation Y is reduced to a lower dimensional space [24]. To this end, in the case of user-based CF, we consider as matrix \mathbf{X} alternatively the two 2-dimensional projections $\pi_{UR}Y \in \{0, 1\}^{|U| \times |R|}$ with $(\pi_{UR}Y)_{u,r} := 1$ if there exists $t \in T$ s. t. $(u, t, r) \in Y$ and 0 else, and $\pi_{UT}Y \in \{0, 1\}^{|U| \times |T|}$ with $(\pi_{UT}Y)_{u,t} := 1$ if there exists $r \in R$ s. t. $(u, t, r) \in Y$ and 0 else (Figure 19.6). One could eventually also consider the resource-tag projection matrix, what would lead to unpersonalized content-based models.

The projections preserve the user information, and lead to RS based on occurrence or non-occurrence of resources or tags, resp., with the users. Notice that we have here two possible setups in which the k -neighborhood N_u^k of a user u can be formed, by considering either the resources or the tags as objects. Having defined matrix \mathbf{X} , and having decided whether to use $\pi_{UR}Y$ or $\pi_{UT}Y$ for computing user neighborhoods, we have the required setup to apply CF. We first compute, based on the row decomposed version of \mathbf{X} and for a given k , the set N_u^k of the k users that are most similar to user u :

Fig. 19.6 Projections of Y into the user’s resource and user’s tag spaces.



$$N_u^k := \arg \max_{v \in U \setminus \{u\}}^k \text{sim}(\mathbf{x}_u, \mathbf{x}_v) \tag{19.1}$$

where the superscript in the $\arg \max$ function indicates the number $k \in \mathbb{N}$ of neighbors to be returned, and sim is any well defined similarity measure, such as, for example, the usual cosine similarity measure, i. e., $\text{sim}(\mathbf{x}_u, \mathbf{x}_v) := \frac{\langle \mathbf{x}_u, \mathbf{x}_v \rangle}{\|\mathbf{x}_u\| \|\mathbf{x}_v\|}$.

Multi-mode Recommendations Having the neighborhood computed, we can extract the set $\hat{T}(u, r)$ of n recommended tags for a given user u , a given resource r , and some $n \in \mathbb{N}$, as follows:

$$\hat{T}(u, r) := \arg \max_{t \in T}^n \sum_{v \in N_u^k} \text{sim}(\mathbf{x}_u, \mathbf{x}_v) \delta(v, t, r) \tag{19.2}$$

where $\delta(v, t, r) := 1$ if $(v, t, r) \in Y$ and 0 else.

If one wants to recommend resources instead, the same principle used for tags can be applied. Note that if we use only the $\pi_{UR}Y$ projection, we would end up at the standard user-based CF algorithm (see Eq. 19.3). But since tags can provide additional information about user interests, they can eventually boost the recommendation quality and thereby should be exploited. A trivial tag-aware recommender method is to compute the user neighborhood based on the $\pi_{UT}Y$ projection matrix and aggregate the resources of the neighborhood to generate the recommendation list. A similar idea is presented in [6], where first the user-tag projection matrix $\pi_{UT}Y$ is used to compute a ranked list of tags, whereby the recommendation list of resources is extracted. But by using only $\pi_{UT}Y$ alone, one discards the resource information, which in this case, is the key mode of interest. In this sense, one needs to find a way to accommodate all the three modes of the folksonomy in a 2-way data structure so that standard CF can be applied. Tso-Sutter et al. [37] proposed an approach for doing that by extending the typical user-resource matrix with tags as pseudo users and pseudo resources (see Figure 19.7). Note that in this way, the user/resource profile is automatically enriched with tags. A fusion algorithm is then proposed for combining user-based CF (*ucf*) and item-based CF (*icf*) predictions

over the extended matrix. Recall that in the standard user-based CF for the resource prediction problem, the interestingness score of a given user u for a particular resource r is computed as the averaged number of neighbors that co-occur with resource r , i.e.,

$$p^{\text{ucf}}(x_{u,r} = 1) := \frac{|\{v \in N_u \mid x_{v,r} = 1\}|}{|N_u|}, \quad (19.3)$$

For item-based CF applied to the resource prediction problem, the algorithm suggested by [3] computes the interestingness score of a given user u for a particular resource r as the averaged sum of similarities between r and its neighboring resources N_r that co-occur with u , i.e.,

$$p^{\text{icf}}(x_{u,r} = 1) := \sum_{\{r' \in N_r \mid x_{u,r'} = 1\}} \text{sim}(r, r') \quad (19.4)$$

The fusion of these two scores is then computed by

$$\begin{aligned} p^{\text{iucf}}(x_{u,r} = 1) := & \lambda \cdot \frac{p^{\text{ucf}}(x_{u,r} = 1)}{\sum_r p^{\text{ucf}}(x_{u,r} = 1)} \\ & + (1 - \lambda) \cdot \frac{p^{\text{icf}}(x_{u,r} = 1)}{\sum_r p^{\text{icf}}(x_{u,r} = 1)} \end{aligned} \quad (19.5)$$

where λ is just a parameter controlling the influence of *ucf* or *icf*. Note that since the values of the prediction lists computed by *ucf* and *icf* have different units (user-based being the frequency of items and item-based the similarity of items), the predicted scores are normalized to unity. For some $n \in \mathbb{N}$, the top- n recommendation list is then generated by

$$\arg \max_r^n p^{\text{iucf}}(x_{u,r} = 1) \quad (19.6)$$

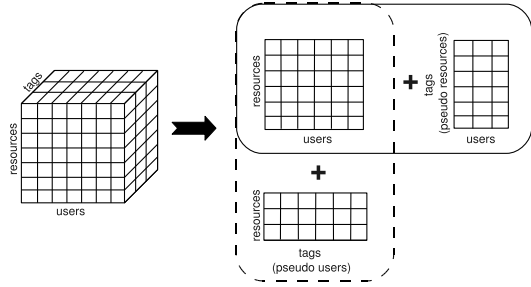
A similar idea was proposed by Wetzker et al. [41], where the probabilistic latent semantic analysis (PLSA) model [10] is extended with tags for the recommendation of resources. In the standard PLSA, the probability that a resource co-occurs with a given user can be computed by

$$P(r|u) := \sum_z P(r|z)P(z|u), \quad (19.7)$$

where $Z := \{z_1, \dots, z_q\}$ is a hidden topic variable and is assumed to be the origin of observed co-occurrence distributions between users and resources. The same hidden topics are then assumed to be the origin of resource/tag co-occurrences, i.e.,

$$P(r|t) := \sum_z P(r|z)P(z|t). \quad (19.8)$$

Fig. 19.7 Extending the user-resource matrix horizontally by including tags as pseudo resources and vertically by including tags as pseudo users.



Both models are then combined on the common factor $P(r|z)$ by maximizing the log-likelihood function

$$L := \sum_r \left[\lambda \sum_u f(r,u) \log P(r|u) + (1 - \lambda) \sum_t f(r,t) \log P(r|t) \right], \quad (19.9)$$

where $f(r,u)$ and $f(r,t)$ correspond to the co-occurrence counts between resources and users, and resources and tags respectively. Here λ is a predefined weight balancing the influence of each model. The usual Expectation-Maximization (EM) algorithm is then applied for performing maximum likelihood estimation for the model. Resources for a given user u are then weighted by the probability $P(r|u)$ (see Eq. 19.7), ranked, and the top ranked resources are finally recommended.

For recommending users, one can either recommend a neighborhood based on $\pi_{UT}Y$ or $\pi_{UR}Y$. In order to recommend a neighborhood that takes into account the three modes of the folksonomy, one could, for example, either use the matrix extensions proposed by [37] (see Figure 19.7) or compute a linear combination of the user similarities based on the user-resource and user-tag projection matrices.

Remarks on Complexity CF usually suffers from scalability problems, given that the whole input matrix needs to be kept in memory. In STS, one may have to eventually keep more than one matrix in memory, depending on which kind of projections one wants to operate upon. To compute recommendations we usually need three steps:

1. Computation of projections: In order to compose the projections, we need to determine the (u,r) , (u,t) and/or (r,t) co-occurrences. For that, we just need to do a linear scan in Y .
2. Neighborhood computation: In traditional user-based CF algorithms, the computation of the neighborhood N_u is usually linear on the number of users as one needs to compute the similarity of a given test user with all the other users in the database. In addition, we need to sort the similarities in order to determine the k -nearest neighbors.
3. Recommendations: For predicting the top- n tag/resource recommendations for a given test user, we need to: (i) count the tags/resources co-occurrences with the nearest neighbors N_u , (ii) weight each co-occurrence by the corresponding

neighbor similarity, and (iii) sort the tags/resources based on their weights (e.g., Eq. 19.2).

19.4.2 Recommendation based on Ranking

In the following we present recommendation algorithms that, inspired from Web ranking, base their recommendations on a ranking score. Their common characteristic is that the score is computed according to spectral attributes extracted from the underlying folksonomy data structure. However, the different ways to represent a folksonomy (see Section 19.2.1) can result in different ranking-based algorithms.

19.4.2.1 Ranking based on Tensor Factorization

By representing Y as a tensor, one is able to exploit the underlying latent semantic structure in \mathcal{A} formed by multi-way correlations between users, tags, and resources. This can be attained using recommendation algorithms that are based on tensor factorization, as the ones proposed in [26, 35, 43]. With such algorithms, multi-way correlations can be effectively detected, leading to improved performance. Chapter 5 presents several state-of-the-art methods for matrix factorization (i.e., second-order tensor factorization) for the problem of rating prediction.

The factorization of \mathcal{A} is expressed as follows (see Figure 19.8):

$$\hat{\mathcal{A}} := \hat{\mathcal{C}} \times_u \hat{U} \times_r \hat{T} \times_t \hat{R} \quad (19.10)$$

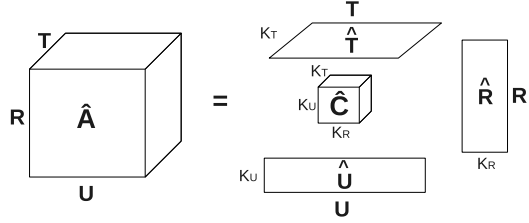
where $\hat{U} \in \mathbb{R}^{|U| \times k_U}$, $\hat{T} \in \mathbb{R}^{|T| \times k_T}$, $\hat{R} \in \mathbb{R}^{|R| \times k_R}$ are low-rank feature matrices representing an entity, i.e., user, resources, and tags resp., in terms of its small number of latent dimensions k_U , k_R , k_T , and $\hat{\mathcal{C}} \in \mathbb{R}^{k_U \times k_R \times k_T}$ is a tensor representing interactions between the latent factors called *core tensor*. The symbol \times_i denotes the i -mode multiplication between a tensor and a matrix. The model parameters to be learned are then the quadruple $\hat{\theta} := (\hat{\mathcal{C}}, \hat{U}, \hat{R}, \hat{T})$. This decomposition refers to a general factorization model known as Tucker decomposition [18]. After the parameters are learned, predictions can be done as follows:

$$\hat{a}_{u,r,t} = \sum_{\tilde{u}} \sum_{\tilde{r}} \sum_{\tilde{t}} \hat{c}_{\tilde{u},\tilde{r},\tilde{t}} \cdot \hat{u}_{u,\tilde{u}} \cdot \hat{r}_{r,\tilde{r}} \cdot \hat{t}_{t,\tilde{t}} \quad (19.11)$$

where indices over the feature dimension of a feature matrix are marked with a tilde, and elements of a feature matrix are marked with a hat (e.g. $\hat{t}_{t,\tilde{t}}$).

Symeonidis et al. [35] proposed to factorize \mathcal{A} , using the 0/1 interpretation scheme (see left-hand side of Figure 19.1), through higher order SVD (HOSVD), which is the multi-dimensional analog of SVD for tensors; see [18] for a recent survey. The basic idea is to minimize an element-wise loss on the elements of $\hat{\mathcal{A}}$ by optimizing the square loss, i.e.,

Fig. 19.8 Tensor Factorization.



$$\operatorname{argmin}_{\hat{\theta}} \sum_{(u,t,r) \in U \times T \times R} (\hat{a}_{u,t,r} - a_{u,t,r})^2$$

Rendle *et al.* [26], on the other hand, propose RTF (Ranking with Tensor Factorization), a method for learning an optimal factorization of \mathcal{A} for the specific problem of tag recommendations. First, the observed tag assignments are divided in positive, negative, and missing values as described in Section 19.2.1 (see right-hand side of Figure 19.1). Let $P_{\mathcal{A}} := \{(u, r) | \exists t \in T : (u, t, r) \in Y\}$ be the set of all distinct user/resource combinations in Y , the sets of positive and negative tags of a particular $(u, r) \in P_{\mathcal{A}}$ are then defined as:

$$T_{u,r}^+ := \{t | (u, r) \in P_{\mathcal{A}} \wedge (u, t, r) \in Y\}$$

$$T_{u,r}^- := \{t | (u, r) \in P_{\mathcal{A}} \wedge (u, t, r) \notin Y\}$$

From this, pairwise tag ranking constraints can be defined for the values of $\hat{\mathcal{A}}$:

$$a_{u,t_1,r} > a_{u,t_2,r} \Leftrightarrow (u, t_1, r) \in T_{u,r}^+ \wedge (u, t_2, r) \in T_{u,r}^- \quad (19.12)$$

Thus, instead of minimizing the least-squares as in the HOSVD-based methods, an optimization criterion that maximizes the ranking statistic AUC (area under the ROC-curve) is proposed. The AUC measure for a particular $(u, r) \in P_{\mathcal{A}}$ is defined as:

$$\operatorname{AUC}(\hat{\theta}, u, r) := \frac{1}{|T_{u,r}^+| |T_{u,r}^-|} \sum_{t^+ \in T_{u,r}^+} \sum_{t^- \in T_{u,r}^-} H_{0.5}(\hat{a}_{u,t^+,r} - \hat{a}_{u,t^-,r}) \quad (19.13)$$

where H_{α} is the Heaviside function:

$$H_{\alpha} := \begin{cases} 0, & x < 0 \\ \alpha, & x = 0 \\ 1, & x > 0 \end{cases} \quad (19.14)$$

The overall optimization task with respect to the ranking statistic AUC and the observed data is then:

$$\operatorname{argmax}_{\hat{\theta}} \sum_{(u,r) \in P_A} \text{AUC}(\hat{\theta}, u, r) \quad (19.15)$$

The optimization problem is then solved by means of gradient descent [26]. Note that with this optimization criterion missing values are also taken into account since the maximization is only done on the observed tag assignments.

Multi-mode Recommendations Once $\hat{\mathcal{A}}$ is computed, the list with the n highest scoring tags for a given user u and a given resource r can be calculated by:

$$\text{Top}(u, r, n) := \operatorname{argmax}_{t \in T}^n \hat{a}_{u,t,r} \quad (19.16)$$

Recommending n resources/users to a given user u for a particular t can be done in a similar manner (see [36]). Thus, tensor modeling may allow multi-mode recommendations in an easy way. However, for the RTF method described above, in which the factorization is learned by solving a specific tag ranking optimization problem, it might be necessary to define a specific optimization function for each mode of interest.

Remarks on Complexity A major benefit of a factorization model like RTF or HOSVD is that after a model is built, predictions only depend on the smaller factorization dimensions. HOSVD can be performed efficiently following the approach of Sun and Kolda [19]. Other approaches to improve the scalability to large data sets is through slicing [38] or approximation [4].

19.4.2.2 FolkRank

The web search algorithm PageRank [2] reflects the idea that a web page is important if there are many pages linking to it, and if those pages are important themselves.¹² In [12], Hotho et al. employed the same underlying principle for Google-like search and ranking in folksonomies. The key idea of the FolkRank algorithm is that a resource which is tagged with important tags by important users becomes important itself. The same holds, symmetrically, for tags and users. We have thus a graph of vertices which are mutually reinforcing each other by spreading their weights. In this section we briefly recall the principles of the FolkRank algorithm, and explain how it can be used for generating tag recommendations.

Because of the different nature of folksonomies compared to the web graph (undirected triadic hyperedges instead of directed binary edges), PageRank cannot be applied directly on folksonomies. In order to employ a weight-spreading ranking scheme on folksonomies, we overcome this problem in two steps. First, we transform the hypergraph into an undirected graph. Then we apply a differential ranking approach that deals with the skewed structure of the network and the undirectedness of folksonomies, and which allows for topic-specific rankings.

¹² This idea was extended in a similar fashion to bipartite subgraphs of the web in HITS [17] and to n-ary directed graphs in [42].

Folksonomy-Adapted PageRank First we convert the folksonomy $\mathbb{F} = (U, T, R, Y)$ into an *undirected* tri-partite graph $G_{\mathbb{F}} = (V, E)$. The set V of nodes of the graph consists of the disjoint union of the sets of tags, users and resources (i. e., $V = U \dot{\cup} T \dot{\cup} R$). All co-occurrences of tags and users, users and resources, tags and resources become edges between the respective nodes. I. e., each triple (u, t, r) in Y gives rise to the three undirected edges $\{u, t\}$, $\{u, r\}$, and $\{t, r\}$ in E .

Like PageRank, we employ the random surfer model, that is based on the idea that an idealized random web surfer normally follows links (e. g., from a resource page to a tag or a user page), but from time to time jumps to a new node without following a link. This results in the following definition.

The rank of the vertices of the graph is computed (like in PageRank) with the weight spreading computation

$$\mathbf{w}_{t+1} \leftarrow dA^T \mathbf{w}_t + (1-d)\mathbf{p}, \quad (19.17)$$

where \mathbf{w} is a weight vector with one entry for each node in V , A is the row-stochastic version of the adjacency matrix¹³ of the graph $G_{\mathbb{F}}$ defined above, \mathbf{p} is the random surfer vector – which we use as preference vector in our setting, and $d \in [0, 1]$ is determining the strength of the influence of \mathbf{p} . By normalization of the vector \mathbf{p} , we enforce the equality $\|\mathbf{w}\|_1 = \|\mathbf{p}\|_1$. This¹⁴ ensures that the weight in the system will remain constant. The rank of each node is its value in the limit $\mathbf{w} := \lim_{t \rightarrow \infty} \mathbf{w}_t$ of the iteration process.

For a global ranking, one will choose $\mathbf{p} = \mathbf{1}$, i. e., the vector composed by 1's. In order to generate recommendations, however, \mathbf{p} can be tuned by giving a higher weight to the user node and to the resource node for which one currently wants to generate a recommendation. The recommendation $\hat{T}(u, r)$ is then the set of the top n nodes in the ranking, restricted to tags.

As the graph $G_{\mathbb{F}}$ is undirected, most of the weight that went through an edge at moment t will flow back at $t + 1$. The results are thus rather similar (but not identical, due to the random surfer) to a ranking that is simply based on edge degrees. In the experiments presented below, we will see that this version performs reasonable, but not exceptional. This is in line with our observation in [12] which showed that the topic-specific rankings are biased by the global graph structure. As a consequence, we developed in [12] the following differential approach.

FolkRank – Topic-Specific Ranking The undirectedness of graph $G_{\mathbb{F}}$ makes it very difficult for other nodes than those with high edge degree to become highly ranked, no matter what the preference vector is. This problem is solved by the *differential* approach in FolkRank, which computes a topic-specific ranking of the elements in a folksonomy. In our case, the topic is determined by the user/resource pair (u, r) for which we intend to compute the tag recommendation.

1. Let $\mathbf{w}^{(0)}$ be the fixed point from Equation (19.17) with $\mathbf{p} = \mathbf{1}$.

¹³ $a_{ij} := \frac{1}{\text{degree}(i)}$ if $\{i, j\} \in E$ and 0 else

¹⁴ ... together with the condition that there are no rank sinks – which holds trivially in the undirected graph $G_{\mathbb{F}}$.

2. Let $\mathbf{w}^{(1)}$ be the fixed point from Equation (19.17) with $\mathbf{p} = \mathbf{1}$, but $\mathbf{p}[u] = 1 + |U|$ and $\mathbf{p}[r] = 1 + |R|$.
3. $\mathbf{w} := \mathbf{w}^{(1)} - \mathbf{w}^{(0)}$ is the final weight vector.

Thus, we compute the winners and losers of the mutual reinforcement of nodes when a user/resource pair is given, compared to the baseline without a preference vector. We call the resulting weight $\mathbf{w}[x]$ of an element x of the folksonomy the *FolkRank* of x .¹⁵

Multi-mode Recommendations For generating tag recommendations for a given user/resource pair (u, r) , we compute the ranking as described and then restrict the result set $\hat{T}(u, r)$ to the top n tag nodes. Similarly, one can compute recommendations for users (or resources) by giving preference to a certain user (or resource). Since FolkRank computes a ranking on all three dimensions of the folksonomy, this produces the most relevant tags, users, and resources for the given user (or resource).

Remarks on Complexity One iteration of the adapted PageRank requires the computation of $dA\mathbf{w} + (d-1)\mathbf{p}$, with $A \in \mathbb{R}^{s \times s}$ where $s := |U| + |T| + |R|$. If t marks the number of iterations, the complexity would therefore be $(s^2 + s)t \in \mathcal{O}(s^2t)$. However, since A is sparse, it is more efficient to go linearly over all *tag assignments* in Y to compute the product $A\mathbf{w}$. After rank computation we have to sort the weights of the tags to collect the top n tags.

19.4.3 Content-Based Social Tagging RS

All the algorithms described so far do not exploit the content of resources, and hence can be applied to any folksonomy regardless the type of resource supported. Nonetheless, the content of resources is a valuable source of information, specially in cold-start scenarios where there is scarcity of explicit user feedback. In the following, we shortly discuss recommenders that make explicit use of resources' content.

19.4.3.1 Text-Based

Song et al. [32] proposed an approach based on graph clustering for tagging textual resources, like web pages or other kinds of documents. It does not perform personalized recommendations, as it does not examine users individually. In particular, it considers the relationship among documents, tags, and words contained in resources. These relationships are represented in two bipartite graphs. The approach is divided in two stages:

¹⁵ In [12] we showed that \mathbf{w} provides indeed valuable results on a large-scale real-world dataset while $\mathbf{w}^{(1)}$ provides an unstructured mix of topic-relevant elements with elements having high edge degree. In [13], we applied this approach for detecting trends over time in folksonomies.

- In the offline stage, it efficiently performs low rank approximation for the weighted adjacency matrix of the two bipartite graphs, using the Lanczos algorithm [8] for symmetrically partitioning the graphs into multi-class clusters. Moreover, a novel node ranking scheme is proposed to rank the nodes corresponding to tags within each cluster. Next, it applies a Poisson mixture model to learn the document distributions for each class.
- In the online stage, given a document vector, based on the joint probabilities of the tags and the document, tags are recommended for this document based on their within-cluster ranking.

As explained in [32], this two-stage framework can be interpreted as an unsupervised-supervised learning procedure. During the offline stage, nodes are partitioned into clusters (unsupervised learning) and cluster labels are assigned to document nodes, acting as “class” labels. Moreover, tag nodes are given ranks in each cluster. A mixture model is then built based on the distribution of document and word nodes. In the online stage, a document is classified (supervised learning) into predefined clusters acquired in the first stage by naive Bayes, so that tags can be recommended in the descending orders of their ranks.

Song et al. [32] emphasize the efficiency of the approach, which is guaranteed by the Poisson mixture modeling that allows recommendations in linear-time. Experimental results with two large data sets crawled from CiteULike (9,623 papers and 6,527 tags) and Delicious (22,656 URLs and 28,457 tags) show that recommendations can be provided within one second.

Different content-based methods to suggest tags, given a resource, have also been investigated recently by Illig et al. [14].

19.4.3.2 Image-Based

Abbasi et al. [1] proposed to use tags as high level features, along with low level image features to train an image classifier on Flickr. Even though this method is not directly applied for RS, it gives some interesting insights about how one can combine tags with low level image features, which could eventually serve as input for a RS. The idea is to first create a vector space from tagging information of images, and then a low level feature space of images using the wavelet transform. These two feature spaces are then joined and used to train a One Class Support Vector Machine (SVM) classifier on the combined feature space.

For creating a feature vector of images based on its tags, a bag-of-words model is used to represent tags as features. Then, the feature vectors are normalized using Term Frequency normalization. Note that this gives low weights to tags in images having a lot of tags. The tag feature vector is then represented as

$$\mathbf{f}_t := (tf(t_1, r), tf(t_2, r), \dots, tf(t_{|T|}, r))^T,$$

where $tf(t, r)$ represents the normalized term frequency value of tag t co-occurring with resource r .

RGB colors are used for the low level feature extraction. Each image r is represented as a four-dimensional vector

$$\mathbf{f}_r := (c_{1,r}, c_{2,r}, c_{3,r}, c_{4,r})^T,$$

where the first component $c_{1,r}$ is the mean pixel value in the image r and the remaining components represent the red, green, and blue channel respectively.

The feature vectors are then combined, i.e.,

$$\mathbf{f}_{t,r} := (tf(t_1, r), tf(t_2, r), tf(t_{|T|}, r), c_{1,r}, c_{2,r}, c_{3,r}, c_{4,r})^T.$$

This combined feature vectors are then used as input for training a One Class SVM classifier. Experiments were done in real data collected for Flickr and it was shown that the classifier trained with the combined feature vectors performed considerably better than if trained only with the tag feature vectors or low level image feature vectors alone.

19.4.3.3 Audio-Based

Eck et al. [5] proposed a method for predicting social tags directly from MP3 files. These tags are called automatic tags (shortly, autotags), because they are directly generated from the musical content. Autotags help in the case where there exist several songs in a collection that are either untagged or poorly tagged. Thus, autotags help to address the “cold-start problem” in music RS. Nevertheless, autotags can be used to smooth the tag space by providing a set of comparable baseline tags for all tracks in a music RS.

Autotags are generated with a machine learning model which uses the meta-learning algorithm AdaBoost to predict tags from audio features, such as: Mel-Frequency Cepstral Coefficients (MFCCs), auto-correlation coefficients computed for selected tags inside songs, and spectrogram coefficients sampled by constant-Q frequency. The audio features are calculated over short sliding windows within the song, leading to an overwhelming total number of features. To reduce this number, “aggregated” features were created by computing individual means and standard deviations (i.e., independent Gaussians) of the previous audio features over short (e.g., 5 seconds) windows of feature data.

Feature selection is performed as follows. The model selects features based on a features ability to minimize empirical error. Therefore, it discards features when weak learners associated with those features are being selected too late by AdaBoost.

Due to the high skewness in the frequency of tags, the prediction task is treated as a classification problem. For each tag, prediction is about whether a particular artist has “none”, “some” or “a lot” of a particular tag relative to other tags. The quantification of the boundaries between the 3 classes is done by summing the normalized tag counts of all artists, generating a 100-bin histogram for each tag and moving the category boundaries such that an equal number of artists fall into each of the categories.

To generate autotags, the classes have to be transformed into a bag of words to be associated with an artist. Based on the semantics of the 3 classes, this is done by subtracting the value of the “none” bin from the value of the “a lot” bin, because “none” is the opposite of “a lot” (thus the “some” class serves just to make the classifier more selective in predicting “none” and “a lot”).

Experimental evaluation of autotag generation was done with 89,924 songs for 1,277 artists, which resulted in more than 1 million 5 seconds aggregated features. Focus was given on the 60 most popular tags from the social online radio station Last.fm. These tags included genres such as “Rock”, and tags related to mood like “chillout”. The classification errors were significantly lower than the random errors. As described by the authors [5], performance should be compared against other classifiers, like SVM or neural networks, in order to better assess the merit of the approach.

19.4.4 Evaluation Protocols and Metrics

In the following we present some of the protocols and metrics used for the evaluation of the different recommendation modes. For a more comprehensive survey on evaluating recommender systems refer to Chapter 8.

Resource Recommendations For evaluating tag-aware resource recommenders, the usual protocols and metrics used for traditional RS can be directly used [9] (c.f. Chapter 8).

Tag Recommendations For evaluating tag recommenders, there are two possible scenarios that can eventually lead to two different evaluation protocols:

- ‘New post’: A user selects a resource that he has not tagged yet and the system tries to suggest a personalized list of n tags for this resource to the user. This protocol was first used in [24, 16] and was called *LeaveOnePostOut* protocol. For each user u , a separate training set is generated by removing all tags of one of his resources r_u , which has been selected randomly. The training set is then the folksonomy (U, T, R, Y') with $Y' := Y \setminus (\{u\} \times \text{tags}(u, r_u) \times \{r_u\})$ where $\text{tags}(u, r_u)$ is the set of tags used by user u for r_u . The task is then to generate a prediction that is close to $\text{tags}(u, r_u)$. This reproduces the scenario in which a user has an untagged resource for which the system tries to generate useful recommendations.
- ‘Tagging refinement’: A user selects a resource that he has already tagged in the past and the system suggests a personalized list of n additional tags that the user might want to use for improving his tagging for the resource. This protocol was first introduced in [35]. The idea is to split the tag assignments into past (training set) and future tag assignments (test set). This reflects the scenario where users gradually tag items and receive recommendations before they provide all their tags.

For both protocols the usual precision, recall and f1-measure metrics are commonly used [15, 35, 11, 26]:

$$\text{Recall}(\hat{T}(u, r_u)) = \frac{|\text{tags}(u, r_u) \cap \hat{T}(u, r_u)|}{|\text{tags}(u, r_u)|} \quad (19.18)$$

$$\text{Precision}(\hat{T}(u, r_u)) = \frac{|\text{tags}(u, r_u) \cap \hat{T}(u, r_u)|}{|\hat{T}(u, r_u)|} \quad (19.19)$$

$$\text{F1-measure}(\hat{T}(u, r_u)) = \frac{2 \cdot \text{Recall}(\hat{T}(u, r_u)) \cdot \text{Precision}(\hat{T}(u, r_u))}{\text{Recall}(\hat{T}(u, r_u)) + \text{Precision}(\hat{T}(u, r_u))} \quad (19.20)$$

Furthermore, both scenarios can be applied in an online setting, where the recommendations are computed in real time and shown to the user during annotation of a resource.¹⁶ One can then record if the user clicked on one of the recommended tags or otherwise used the recommendation (e.g., with autocompletion mechanisms). This setting is probably the most realistic one and gives a good measure on how the user liked the recommendation. However, it is pretty laborious to set up and needs a system with active users. There are also some users which don't click on recommendations or use autocompletion which would affect this evaluation.

User Recommendations For the task of user recommendation, the system suggests to the target user a personalized list of n users, which form his neighborhood. Some systems, like Last.fm, provide information about links between users (in Last.fm socially connected users are called *neighbors*). If such information is available, then it can serve as a ground truth for the evaluation of user recommendation. In cases where such ground truth is not available (like in Bibsonomy), the evaluation of user recommendation can be performed along the lines of evaluating user communities, most notably by calculating the item similarities within them as described in [22]. In particular, in a social bookmarking system (like BibSonomy), for each web resource, its first (home) page can be crawled and preprocessed to create a vector of terms. This way, between any two web resources, their cosine similarity can be computed as follows. For each test user's neighborhood (i.e., most similar users), the Average Cosine Similarity (ACS) of all web resource pairs inside the neighborhood can be computed. ACS corresponds to the intra-neighborhood similarity. Moreover, from a selected number of random neighborhood pairs among all test users' neighborhoods, the average pairwise web resource similarity between every two neighborhoods can be computed. This measure corresponds to the inter-neighborhood similarity. Therefore, the quality of recommended user-neighborhood is judged according both to its intra-neighborhood similarity (the higher the better) and to its inter-neighborhood-similarity (the lower the better).

¹⁶ This was one of the tasks of the ECML PKDD Discovery Challenge 2009 – see <http://www.kde.cs.uni-kassel.de/ws/dc09/>.

19.5 Comparison of Algorithms

In this section, we briefly discuss the main advantages and disadvantages of the algorithms presented in Section 19.4. Note that we just consider the non content-based algorithms since they can be compared under a common basis.

We saw in Section 19.4.1 that in order to apply standard CF-based algorithms to folksonomies, some data transformation must be performed. Such transformations lead to information loss, which can lower the recommendation quality. Another well known problem with CF-based methods is that large projection matrices must be kept in memory, which can be time/space consuming thus compromising the ability to perform real-time recommendations. Another problem is that for each different mode to be recommended, the algorithm must be eventually changed, demanding an additional effort for offering multi-mode recommendations.

FolkRank builds on PageRank and proved to give significantly better tag recommendations than CF [15]. This method also allows for mode switching with no change in the algorithm. Moreover, as well as CF-based algorithms, FolkRank is robust against online updates since it does not need to be trained every time a new user, resource or tag enters the system. However, FolkRank is computationally expensive and not trivially scalable, making it more suitable for systems where real-time recommendations is not a requirement.

Similarly to FolkRank, tensor factorization methods work directly on the ternary relations of folksonomies. Although the learning phase can be costly, it can be performed offline. After the model is learned, the recommendations can be done fast, making these algorithms suitable for real-time recommendations. A potential disadvantage of tensor factorization methods is that easy mode switching can only be achieved if one consider that the different recommendation problems, i.e., user/resource/tag, can be addressed by minimizing the same error function. If one chooses HOSVD, for example, the model can be used for multi-mode recommendations with trivial mode switching, but at the cost of evtl. solving the wrong problem: HOSVD minimizes a least-square error function while social tagging RS are more related to ranking. If one tries to optimally reconstruct the tensor w.r.t. an error function targeted to a specific recommendation mode, on the other hand, accuracy is eventually improved for the targetted mode, but at the cost of making mode switching more involved. Figure 19.9 shows a comparison between some of the aforementioned algorithms in a snapshot of the BibSonomy dataset for the tag recommendation problem [26]. Note that the best method is RTF followed by FolkRank and HOSVD.

Table 19.2 summarizes this discussion. Note that the absence of a “X” in Table 19.2 indicates that the corresponding property is not trivially achieved by the algorithm being considered.

Fig. 19.9 F-Scores for Top-1, Top-2 to Top-10 lists on a snapshot of the BibSonomy dataset. FolkRank, PageRank and HOSVD are compared to RTF with an increasing number of dimensions under the *LeaveOnePostOut* protocol [26].

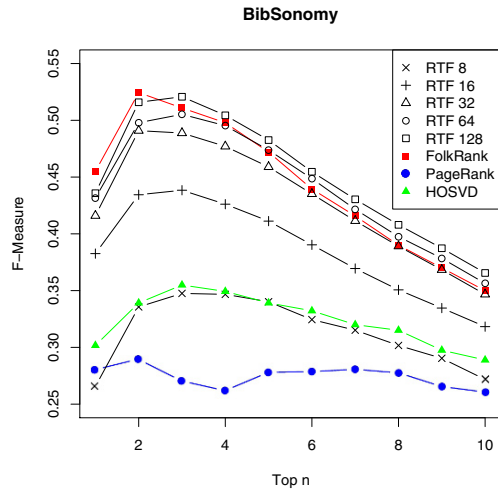


Table 19.2: Summary of advantages and disadvantages of the presented algorithms.

Method	Scalable	Multi-mode recommendation	Keeps Ternary Relation	Online Update
CF-based				X
Folkrank		X	X	X
HOSVD	X	X	X	
RTF	X		X	

19.6 Conclusions and Research Directions

The Web 2.0 represents a shift of paradigm from the Web-as-information-source to a Web-as-participation-platform where users can upload content, exercise control over that content, and therefore add value to the application as they use it. Among the most prominent family of Web 2.0 applications are the Social Tagging Systems, which promote the sharing of user’s tags/resources by exposing them to other users. Due to the increasing popularity of these systems, issues like information overload rapidly become a problem. RS proved to be well suited for this kind of problem in the past and are thus a prominent solution for tackling the information overload in the Web’s next generation. In this chapter we presented:

- The data structures of folksonomies, stressing the differences in comparison to the ones used by traditional RS.
- The different modes that can be recommended in STS.
- RS deployed in real STS, stressing the challenges and requirements for doing so.
- Different ways of acquiring tags and how they can affect recommender algorithms.

- Algorithms that:
 - reduce the data dimensionality in order to apply standard CF algorithms,
 - operate directly on the ternary relational data of folksonomies,
 - exploit the content of resources.
- Evaluation protocols and metrics.
- Comparison of the algorithms in terms of pros and cons.

Although the methods that transform the original folksonomy data allow the direct application of standard CF-based algorithms, the transformation inevitably cause some loss of information, which can lower the overall recommendation quality. Some methods try to overcome this problem by doing some sort of ensemble over the different data projections resulting from the transformation, which adds additional free parameters to the problem in order to control the influence of each component. A more natural solution is to operate over the original ternary relation of a folksonomy, which requires the development of new RS algorithms such as FolkRank, that explores the folksonomy hypergraph, or the ones based on tensor factorization. Although FolkRank is known for its high predictive quality, it suffers from scalability problems, and so an interesting research direction is to investigate ways of making it scale.

Tensor factorization for social tagging RS is a recent and prominent field. The research work on this topic has just started to uncover the benefits that those methods have to offer. A particularly appealing research direction concerns investigating tensor factorization models that feature both high recommendation accuracy and easy mode switching.

As pointed out before, folksonomies usually do not contain numerical ratings, but recently the GroupLens¹⁷ research group released a folksonomy dataset in which numerical ratings for the tagged resources are also given.¹⁸ This represents several research opportunities on how to exploit the resource's rating information in order to improve recommendations. In this case, a single data structure for all the modes, such as tensors or hypergraphs, would evtl. fail since the ratings are only related to user-resource pairs and not to tags. Similar issues can be investigated for content-based methods. We saw that content-based methods usually disregard the user information, but past research shows that hybrid methods that combine user preferences with resource's content usually lead to better recommenders. Here, again, tensor or hypergraph representations would evtl. fail since resources' content are only related to the resources but not to the users or tags. So hybrid-based methods that perform some sort of fusion between folksonomy representations and resources' content would be a valuable contribution to the area.

Other topics that were not covered in this chapter, but are nevertheless interesting research directions, concern, for example, recommendations' novelty and serendipity [44], i.e., tags, users and/or resources that are potentially interesting but not obvious; modeling social wisdom for recommendations [40, 29], i.e., explicit friendship

¹⁷ <http://www.grouplens.org/>

¹⁸ This dataset can be downloaded at <http://www.grouplens.org/node/73>

strength and mutual trust relations among users, that are evtl. orthogonal to similarities of interests and behavior, can be modeled and used to improve the quality of RS.

References

1. Rabeeh Abbasi, Marcin Grzegorzek, and Steffen Staab. Using colors as tags in folksonomies to improve image classification. In *SAMT '08: Poster at Semantics And digital Media Technologies*, 2008.
2. Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998.
3. Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.
4. P. Drineas and M. W. Mahoney. A randomized algorithm for a tensor-based generalization of the svd. *Linear Algebra and Its Applications*, 420(2–3):553–571.
5. Douglas Eck, Paul Lamere, Thierry Bertin-Mahieux, and Stephen Green. Automatic generation of social tags for music recommendation. In *NIPS'07: Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, volume 20, 2007.
6. Claudiu S. Firan, Wolfgang Nejdl, and Raluca Paiu. The benefit of using tag-based profiles. In *LA-WEB '07: Proceedings of the 2007 Latin American Web Conference*, pages 32–41, Washington, DC, USA, 2007. IEEE Computer Society.
7. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin – Heidelberg, 1999.
8. Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.
9. Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.
10. Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, New York, NY, USA, 1999. ACM.
11. Andreas Hotho, Dominik Benz, Robert Jäschke, and Beate Krause, editors. *ECML PKDD Discovery Challenge 2008 (RSDC'08)*. Workshop at 18th Europ. Conf. on Machine Learning (ECML'08) / 11th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'08), 2008.
12. Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in folksonomies: Search and ranking. In York Sure and John Domingue, editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, pages 411–426, Heidelberg, June 2006. Springer.
13. Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Trend detection in folksonomies. In *SAMT '06: Proceedings of the first International Conference on Semantics And Digital Media Technology*, volume 4306 of *LNCS*, pages 56–70, Heidelberg, Dec 2006. Springer.
14. Jens Illig, Andreas Hotho, Robert Jäschke, and Gerd Stumme. A comparison of content-based tag recommendations in folksonomy systems. In *KPP '07: Postproceedings of the International Conference on Knowledge Processing in Practice*, 2009 (to appear).
15. Robert Jäschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in social bookmarking systems. *AI Communications*, pages 231–247, 2008.
16. Robert Jäschke, Leandro Balby Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in folksonomies. In *PKDD '07: Proceedings of the 11th*

- European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 4702 of *Lecture Notes in Computer Science*, pages 506–514, Berlin, Heidelberg, 2007. Springer.
17. Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
 18. Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September 2009.
 19. Tamara G. Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In *ICDM '08: Proceedings of the 8th IEEE International Conference on Data Mining*, pages 363–372, December 2008.
 20. Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1986.
 21. F. Lehmann and R. Wille. A triadic approach to formal concept analysis. In G. Ellis, R. Levinson, W. Rich, and J. F. Sowa, editors, *Conceptual Structures: Applications, Implementation and Theory*, volume 954 of *Lecture Notes in Computer Science*, pages 32–43. Springer, 1995.
 22. Xin Li, Lei Guo, and Yihong Eric Zhao. Tag-based social interest discovery. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 675–684, New York, NY, USA, 2008. ACM.
 23. M. Srikanth M. Tatu and T. D'Silva. Tag recommendations using bookmark content. In *ECML/PKDD '08: Proceedings of the ECML PKDD Discovery Challenge at 18th Europ. Conf. on Machine Learning / 11th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases*, 2008.
 24. Leandro Balby Marinho and Lars Schmidt-Thieme. Collaborative tag recommendations. In *GFKL '07: Proceedings of the 31st Annual Conference of the Gesellschaft für Klassifikation (GfKI), Freiburg*, pages 533–540. Springer, 2007.
 25. Oren Patashnik. BibTeXing, 1988. (Included in the BibTeX distribution).
 26. Steffen Rendle, Leandro B. Marinho, Alexandros Nanopoulos, and Lars S. Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736. ACM, 2009.
 27. Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, New York, NY, USA, 1994. ACM.
 28. Lior Rokach and Oded Maimon, Theory and applications of attribute decomposition, IEEE International Conference on Data Mining, IEEE Computer Society Press, pp. 473–480, 2001.
 29. Ralf Schenkel, Tom Crecelius, Mouna Kacimi, Thomas Neumann, Josiane Parreira, Marc Spaniol, and Gerhard Weikum. Social wisdom for search and recommendation, June 2008. Accepted for publication.
 30. Shilad Sen, Jesse Vig, and John Riedl. Tagommenders: connecting users to items through tags. In *WWW '09: Proceedings of the 18th international conference on World Wide Web*, pages 671–680, New York, NY, USA, 2009. ACM.
 31. Andriy Shepitsen, Jonathan Gemmell, Bamshad Mobasher, and Robin Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 259–266, New York, NY, USA, 2008. ACM.
 32. Yang Song, Ziming Zhuang, Huajing Li, Qiankun Zhao, Jia Li, Wang-Chien Lee, and C. Lee Giles. Real-time automatic tag recommendation. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 515–522, New York, NY, USA, 2008. ACM.
 33. Sanjay Sood, Sara Owsley, Kristian Hammond, and Larry Birnbaum. Tagassist: Automatic tag suggestion for blog posts. In *ICWSM '07: Proceedings of the International Conference on Weblogs and Social Media*, 2007.
 34. Gerd Stumme. A finite state model for on-line analytical processing in triadic contexts. In *ICFCA*, pages 315–328, 2005.

35. Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. Tag recommendations based on tensor dimensionality reduction. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 43–50, New York, NY, USA, 2008. ACM.
36. Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis. *IEEE Transactions on Knowledge and Data Engineering*, 22(2), 2010.
37. Karen H. L. Tso-Sutter, Leandro Balby Marinho, and Lars Schmidt-Thieme. Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1995–1999, New York, NY, USA, 2008. ACM.
38. P. Turney. Empirical evaluation of four tensor decomposition algorithms. *Technical Report (NRC/ERB-1152)*, 2007.
39. Luis von Ahn and Laura Dabbish. Designing games with a purpose. *Commun. ACM*, 51(8):58–67, 2008.
40. Frank E. Walter, Stefano Battiston, and Frank Schweitzer. Personalised and dynamic trust in social networks. In *RecSys '09: Proceedings of the 2009 ACM conference on Recommender systems*, New York, NY, USA, 2009. ACM. to appear.
41. Robert Wetzker, Winfried Umbrath, and Alan Said. A hybrid approach to item recommendation in folksonomies. In *ESAIR '09: Proceedings of the WSDM '09 Workshop on Exploiting Semantic Annotations in Information Retrieval*, pages 25–29. ACM, 2009.
42. Wensi Xi, Benyu Zhang, Zheng Chen, Yizhou Lu, Shuicheng Yan, Wei-Ying Ma, and Edward Allan Fox. Link fusion: a unified link analysis framework for multi-type interrelated data objects. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 319–327, New York, NY, USA, 2004. ACM.
43. Yanfei Xu, Liang Zhang, and Wei Liu. Cubic analysis of social bookmarking for personalized recommendation. pages 733–738. 2006.
44. Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, pages 22–32, New York, NY, USA, 2005. ACM.

Chapter 20

Trust and Recommendations

Patricia Victor, Martine De Cock, and Chris Cornelis

Abstract Recommendation technologies and trust metrics constitute the two pillars of trust-enhanced recommender systems. We discuss and illustrate the basic trust concepts such as trust and distrust modeling, propagation and aggregation. These concepts are needed to fully grasp the rationale behind the trust-enhanced recommender techniques that are discussed in the central part of the chapter, which focuses on the application of trust metrics and their operators in recommender systems. We explain the benefits of using trust in recommender algorithms and give an overview of state-of-the-art approaches for trust-enhanced recommender systems. Furthermore, we explain the details of three well-known trust-based systems and provide a comparative analysis of their performance. We conclude with a discussion of some recent developments and open challenges, such as visualizing trust relationships in a recommender system, alleviating the cold start problem in a trust network of a recommender system, studying the effect of involving distrust in the recommendation process, and investigating the potential of other types of social relationships.

20.1 Introduction

Collaboration, interaction and information sharing are the main driving forces of the current generation of web applications referred to as ‘Web 2.0’ [48]. Well-known examples of this emerging trend include weblogs (online diaries or journals for sharing ideas instantly), Friend-Of-A-Friend¹ (FOAF) files (machine-readable documents

Patricia Victor · Chris Cornelis

Dept. of Applied Mathematics and Computer Science, Ghent University, Krijgslaan 281 (S9), 9000 Gent, Belgium e-mail: Patricia.Victor,Chris.Cornelis@ugent.be

Martine De Cock

Institute of Technology, University of Washington Tacoma, 1900 Pacific Ave, Tacoma, WA, USA (on leave from Ghent University) e-mail: mdecock@u.washington.edu

¹ See www.foaf-project.org

describing basic properties of a person, including links between the person and objects/people they interact with), wikis (web applications such as Wikipedia² that allow people to add and edit content collectively) and social networking sites (virtual communities where people with common interests can interact, such as Facebook³, dating sites, car addict forums, etc.). In this chapter, we focus on one specific set of Web 2.0 applications, namely *social recommender systems*. These recommender systems generate predictions (recommendations) that are based on information about users' profiles and relationships between users. Nowadays, such online relationships can be found virtually everywhere, think for instance of the very popular social networking sites Facebook, LinkedIn and MSN⁴.

Research has pointed out that people tend to rely more on recommendations from people they trust (friends) than on online recommender systems which generate recommendations based on anonymous people similar to them [57]. This observation, combined with the growing popularity of open social networks and the trend to integrate e-commerce applications with recommender systems, has generated a rising interest in *trust-enhanced recommendation systems*. The recommendations generated by these systems are based on information coming from an (online) *trust network*, a social network which expresses how much the members of the community trust each other. A typical example is Golbeck's FilmTrust [16], an online social network combined with a movie rating and review system in which users are asked to evaluate their acquaintances' movie tastes on a scale from 1 to 10. Another example is the e-commerce site Epinions.com, which maintains a trust network by asking its users to indicate which members they trust (i.e., their personal 'web of trust') or distrust ('block list').

Trust-enhanced recommender systems use the knowledge that originates from such trust networks to generate more personalized recommendations: users receive recommendations for items rated highly by people in their web of trust (WOT), or even by people who are trusted by these WOT members, etc. (see e.g. [7, 16, 46, 61]). The main strength of most of these systems is their use of *trust propagation* and *trust aggregation* operators; mechanisms to estimate the trust transitively by computing how much trust a user a has in another user c , given the value of trust for a trusted third party b by a and c by b (propagation), and by combining several trust estimates into one final trust value (aggregation). Propagation and aggregation are the two key building blocks of *trust metrics*, which aim to estimate the trust between two unknown users in the network.

Apart from trust, in a large group of users (each with their own intentions, tastes and opinions) it is only natural that also *distrust* occurs. For example, Epinions first provided the possibility to include users in a personal WOT (based on their quality as a reviewer), and later on also introduced the concept of a personal 'block list', reflecting the members that are distrusted by a particular user. The information in

² See www.wikipedia.org

³ See www.facebook.com

⁴ See www.linkedin.com, or www.msn.com

the WOT and block list is then used to make the ordered list of presented reviews more personalized. From a research perspective, too, it is generally acknowledged that distrust can play an important role [21, 62, 68], but much ground remains to be covered in this domain.

Recommendation technologies and trust metrics constitute the two pillars of trust-enhanced recommender systems. Since the former are covered in much detail in other chapters of this handbook, we will restrict ourselves to the essentials. On the other hand, we do not assume that most readers are familiar with the trust research area. Therefore, in the following section, we start with a discussion and illustration of the basic trust concepts, namely trust and distrust modeling, propagation and aggregation; concepts that are needed to fully grasp the rationale behind the trust-enhanced recommender techniques as they are discussed in Section 3. This is the central part of the chapter, and focuses on the application of trust metrics and their operators in recommender systems. We explain the benefits of using trust in recommender algorithms and give an overview of state-of-the-art approaches for trust-enhanced recommender systems. Furthermore, we explain the details of three well-known trust-based systems and provide a comparative analysis of their performance. After this overview of classical trust-enhanced research, in Section 4, we focus on some recent developments and open challenges, such as visualizing trust relationships in a recommender system, alleviating the cold start problem in a trust network of a recommender system, studying the effect of involving distrust in the recommendation process, and investigating the potential of other types of social relationships. The chapter is concluded in Section 5.

20.2 Computational Trust

In this section we provide the reader with a basic introduction to the field of computational interpersonal trust, i.e., trust that can be computed among two individuals in a social trust network. This implies that we cover trust models (how to represent trust and how to deal with distrust; Section 2.1), trust propagation operators (how to estimate the trust between two individuals by using information coming from users that are on the connecting path between them; Section 2.2.1), and trust aggregation (how to combine trust values generated by multiple propagation paths; Section 2.2.2). We illustrate these concepts by classical and recent examples.

Note that this overview is inexhaustive; for instance, we do not cover trust updating or trust bootstrapping. Our primary goal is to familiarize the reader with the main concepts of the trust computation area, and as such to lay the foundation for an easy understanding of the rationale and details of the trust-enhanced recommendation techniques presented in Section 3.

20.2.1 Trust Representation

Trust models come in many flavours and can be classified in several ways. In this chapter we focus on two such classifications, namely probabilistic versus gradual approaches, and representations of trust versus representations of both trust and distrust. Table 20.1 shows some representative references for each class.

A *probabilistic* approach deals with a single trust value in a black or white fashion — an agent or source can either be trusted or not — and computes a probability that the agent can be trusted. In such a setting, a higher suggested trust value corresponds to a higher probability that an agent can be trusted. Examples can, among others, be found in [66] in which Zaihrayeu et al. present an extension of an inference infrastructure that takes into account the trust between users and between users and provenance elements in the system, in [55] where the focus is on computing trust for applications containing semantic information such as a bibliography server, or in contributions like [32] in which a trust system is designed to make community blogs more attack-resistant. Trust is also often based on the number of positive and negative transactions between agents in a virtual network, such as in Kamvar et al.’s Eigentrust for peer-to-peer (P2P) networks [28], or Noh’s formal model based on feedbacks in a social network [44]. Both [25] and [51] use a subjective logic framework (discussed later on in this section) to represent trust values; the former for quantifying and reasoning about trust in IT equipment, and the latter for determining the trustworthiness of agents in a P2P system.

On the other hand, a *gradual* approach is concerned with the estimation of trust values when the outcome of an action can be positive to some extent, e.g. when provided information can be right or wrong to some degree, as opposed to being either right or wrong (e.g. [1, 11, 15, 21, 35, 59, 68]). In a gradual setting, trust values are not interpreted as probabilities: a higher trust value corresponds to a higher trust in an agent, which makes the ordering of trust values a very important factor in such scenarios. Note that in real life, too, trust is often interpreted as a gradual phenomenon: humans do not merely reason in terms of ‘trusting’ and ‘not trusting’, but rather trusting someone ‘very much’ or ‘more or less’. Fuzzy logic [29, 65] is very well-suited to represent such natural language labels which represent vague intervals rather than exact values. For instance, in [59] and [31], fuzzy linguistic terms are used to specify the trust in agents in a P2P network, and in a social network, respectively. A classical example of trust as a gradual notion can be found in [1], in which a four-value scale is used to determine the trustworthiness of agents, viz. very trustworthy - trustworthy - untrustworthy - very untrustworthy.

The last years have witnessed a rapid increase of gradual trust approaches, ranging from socio-cognitive models (for example implemented by fuzzy cognitive maps in [12]), over management mechanisms for selecting good interaction partners on the web [59] or for pervasive computing environments (Almenáez et al.’s PTM [3]), to representations for use in recommender systems [15, 35], and general models tailored to semantic web applications [68].

Table 20.1: Classification of trust models

	<i>trust only</i>	<i>trust and distrust</i>
<i>probabilistic</i>	Kamvar et al. [28] Richardson et al. [55] Zaihrayeu et al. [66]	Jøsang et al. [25]
<i>gradual</i>	Abdul-Rahman et al. [1] Falcone et al. [12] Golbeck [15] Massa et al. [35]	Victor et al. [62] Guha et al. [21]

While trust is increasingly getting established, the use and modeling of *distrust* remains relatively unexplored. Most approaches completely ignore distrust (see for example [31, 32, 43, 55, 66]), or consider trust and distrust as opposite ends of the same continuous scale (see e.g. [1, 19, 59]). However, in agent network theory there is a growing body of opinion that distrust cannot be seen as the equivalent of lack of trust [10, 13, 34]. Moreover, work in the psychology area has repeatedly asked for a re-examination of the assumption that positive- and negative-valent feelings are not separable [8, 50, 52], and some researchers even claim that trust and distrust are not opposite, but related dimensions that can occur simultaneously [9, 33].

To the best of our knowledge, there is only one probabilistic model that considers trust and distrust simultaneously: in Jøsang's subjective logic [24, 25], an opinion includes a belief b that an agent is to be trusted, a disbelief d corresponding to a belief that an agent is not to be trusted, and an uncertainty u . The uncertainty factor leaves room for ignorance, but the requirement that the belief b , the disbelief d and the uncertainty u sum up to 1, rules out options for inconsistency even though this might arise quite naturally in large networks with contradictory sources [60].

Examples of gradual models for both trust and distrust can be found in [11, 21, 62, 68]. Guha et al. use a couple (t, d) with a trust degree t and a distrust degree d , both in $[0, 1]$. To obtain the final suggested trust value, they subtract d from t [21]. However, as explained in [62], potentially important information is lost when the trust and distrust scales are merged into one. For example, the scenario $(0.2, 0)$ in which there is partial trust collapses to 0.2, but so does the scenario $(0.6, 0.4)$ that exhibits both partial trust *and* partial distrust. To deal with the issues in Guha's and Jøsang's approach, Victor et al. proposed an extension of [11] in which trust and distrust values are drawn from a bilattice [14]. Such a bilattice structure is able to solve trust problems caused by presence of distrust or lack of knowledge, and provides insight into knowledge problems caused by having too little or too much, i.e. contradictory, information [62].

Trust and trust models have been used in many fields of computer science, and also in a wide range of applications; a nice overview can be found in [6] in which

Artz and Gil classify trust research in four major areas: models that use policies to establish trust (enforcing access policies, managing credentials, etc.), general trust models such as [12] and [68], models for trust in information sources such as [66], and reputation-based trust models. The latter category includes, among others, research that uses the history of an agent's actions or behaviour (see e.g. [28, 46]), and work that computes trust over social networks, such as [21, 36]. In fact, the trust-enhanced recommender techniques that we will describe in Section 20.3 all belong to this class.

20.2.2 Trust Computation

In online trust networks, most other users are typically unknown to a specific user. Still there are cases in which it is useful to be able to derive some information on whether or not an unknown user can be trusted, and if so, to what degree. In the context of recommender systems for instance, this is important if none of the known users has rated a specific item that the user is interested in, but there are some ratings available by unknown users (who are a member of the trust network). For instance, the number of people that users have in their web of trust in Epinions is estimated to be around 1.7 on average. The total number of users of Epinions on the other hand well exceeds 700 000 [61]. In other words, the WOT of a user only contains a very tiny fraction of the user community. Hence, it would be very useful to be able to tap into the knowledge of a larger subset of the user population to generate recommendations.

Trust metrics compute an estimate of how much a user should trust another user, based on the existing trust relations between other users in the network. Various types of trust metrics exist in the literature; we refer to [68] for a good overview. In that paper, Ziegler and Lausen classify trust metrics along three dimensions: group versus scalar metrics, centralized versus distributed approaches, and global versus local metrics. The first dimension refers to the way trust relations are evaluated, while the second classification is based on the place where the trust estimations are computed. The last dimension refers to the network perspective: trust metrics can take into account all users and trust relationships between them when computing a trust estimation (see e.g. [28, 44, 55]), or only rely on a part of the trust network, hence taking into account personal bias (e.g. [15, 21, 35]). The trust-enhanced techniques of Section 3 belong to the latter type.

20.2.2.1 Propagation

Trust metrics usually incorporate techniques that are based on the assumption that trust is somehow transitive. We call these techniques trust propagation strategies. Let us illustrate this with Figure 20.1: if user *a* trusts user *b* (whom we call a trusted third party, or TTP for short), and TTP *b* trusts user *c*, then it is reasonable to assume

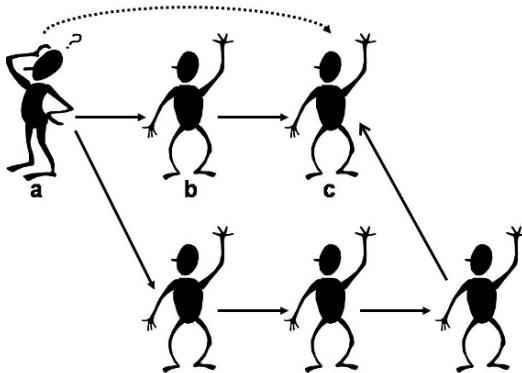


Fig. 20.1: Propagation example

that a should trust c to a certain degree. This basic propagation strategy is known as atomic direct propagation, and is the type that we will focus on in the remainder of this chapter⁵. However, trust is not always transitive. For instance, if Jane trusts Alice to give her a good-looking haircut and Alice trusts John to fix her bicycle, this does not imply that Jane trusts John to fix bicycles, nor to give a nice haircut.

But, in the same context/scope, and under certain conditions, trust can be transitive [26]. Suppose e.g. that Jane is new in town and wants to have a haircut. Jane trusts that Alice can find a good hairdresser, while Alice trusts Mariah to be a good hairdresser. Hence, Jane can trust Mariah to be a good hairdresser. This example also shows us that a distinction must be made between trust in a user's competence to assess the trustworthiness of a user (functional trust, Alice trusting Mariah), or trust in a user's competence to recommend/evaluate a good recommender agent (referral trust, Jane trusting Alice) [1, 26]. As explained in [26], it is the referral part that allows trust to become transitive. A propagation path can then be seen as a transitive chain of referral trust parts, which ends with one functional trust scope.

When dealing with trust only, in a probabilistic setting, multiplication is very often used as the standard propagation operator, see for instance [55]. This is also the case in gradual settings [3, 15, 21], but there is a wider spectrum of propagation operators available, dependent on the goal or the spirit of the application. This is illustrated by the following example.

⁵ For a discussion of other trust propagation strategies, such as cocitation, transpose trust, or coupling, we refer to [21].

Example 20.1. Suppose that, on a scale from 0 to 1, user a trusts user b to the degree 0.5, and that b trusts user c to the degree 0.7. Then, in a probabilistic setting (using standard multiplication), trust propagation yields 0.35. In a fuzzy logic approach however, the final trust estimate depends on the choice of the operator: for instance, the rationale that a propagation chain is only as strong as its weakest link leads to the use of the minimum as propagation operator, hence yielding 0.5 as the propagated trust estimate. The use of the Łukasiewicz conjunction operator on the other hand, i.e. $\max(t_1 + t_2 - 1, 0)$, will yield 0.2. Like with multiplication, this propagated trust value reflects the individual influences of both composing links, as opposed to only the weakest link.

Other trust propagation work includes techniques based on fuzzy if-then rules [31, 59], on the theory of spreading activation models (Ziegler and Lausen's Appleseed [68]), or on the semantic distance between a TTP's trust and a user's perception of the TTP's trust [1].

Of course, not all propagation paths have the same length. In Figure 20.1 e.g., there are two paths leading from the source user a to the target user c . If we suppose that all trust links in the network denote complete trust, then intuitively we feel that the estimated trust of the second propagation path should be lower than that of the first path, since we are heading further away from the source user. This idea of 'trust decay' [20] is often implemented in propagation strategies. For instance, in Ziegler's approach this is incorporated through a spreading factor [68], Golbeck only takes into account shortest paths and ignores all others [15], and in applications that only work with binary trust (instead of gradual), Massa determines the propagated trust based on a user's distance from a fixed propagation horizon [35].

In the case of atomic direct propagation, if a trusts b and b trusts c , a might trust c to a certain degree. Analogously, if a trusts b and b distrusts c , it seems clear that a should somehow distrust c . However, the picture gets more complicated when we also allow distrust as the first link in a propagation chain. For example, if a distrusts b and b distrusts c , there are several options for the trust estimation of a in c : a possible reaction is to infer that a should trust c , since a might think that distrusted acquaintances of users he distrusts are best to be trusted ('the enemy of your enemy is your friend'). Or a should distrust c because a thinks that someone that is distrusted by a user that he distrusts certainly must be distrusted. Yet another interpretation of distrust propagation is to ignore information coming from a distrusted user b , because a might decide not to take into account anything that a distrusted user says.

Guha et al. call the second strategy additive distrust propagation, and the first multiplicative distrust propagation [21]. They discuss the negative side effects of multiplicative propagation (also see [68]), but conclude that it cannot be ignored because it has some philosophical defensibility. Besides Guha et al., other researchers also proposed operators that adhere to the first strategy, such as Victor et al.'s approach using fuzzy logic concepts [62] or Jøsang et al.'s opposite belief favouring discount operator [27]. Examples of the last strategy can be found in [21, 27, 62].

Example 20.2. Like with trust propagation, approaches to distrust propagation are intimately linked to the representations of trust and distrust at hand. Let us assume the use of a couple (t, d) with a trust degree t and a distrust degree d , both in $[0, 1]$. In this representation, $(1, 0)$ corresponds to full trust, $(0, 1)$ corresponds to full distrust, and $(0, 0)$ corresponds to full ignorance, or full lack of knowledge. Gradual values such as in $(0.5, 0.2)$ denote partial trust 0.5, partial distrust 0.2 and partial lack of knowledge $1 - 0.5 - 0.2 = 0.3$. Assume that the trust score of user a in user b is (t_1, d_1) and, likewise, that the trust score of user b in user c is (t_2, d_2) . The trust score (t_3, d_3) of user a in user c can then be calculated as follows [62]:

$$(t_3, d_3) = (t_1 \times t_2, t_1 \times d_2)$$

This propagation strategy reflects the attitude of listening to whom you trust and not deriving any knowledge through a distrusted or unknown third party. Below are some examples of propagated trust scores. Each row correspond to a possible trust score of a in b , each column to a trust score of b in c , and the corresponding table entry contains the propagated trust score of a in c .

	(0.0,0.0)	(0.0,1.0)	(1.0,0.0)	(0.5,0.2)
(0.0,0.0)	(0.0, 0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
(0.0,1.0)	(0.0, 0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
(1.0,0.0)	(0.0,0.0)	(0.0,1.0)	(1.0,0.0)	(0.5,0.2)
(0.5,0.2)	(0.0,0.0)	(0.0, 0.5)	(0.5,0.0)	(0.25,0.1)

In [25] the same propagation technique is used to combine pairs of beliefs and dis-beliefs. Furthermore, subtracting the distrust degree from the trust degree, the propagated trust score collapses to $t_1 \times (t_2 - d_2)$, a propagation scheme proposed in [21].

Example 20.3. Alternatively, the trust score (t_3, d_3) of user a in user c can be calculated as [62]:

$$(t_3, d_3) = (t_1 \times t_2 + d_1 \times d_2 - t_1 \times t_2 \times d_1 \times d_2, t_1 \times d_2 + d_1 \times t_2 - t_1 \times d_2 \times d_1 \times t_2)$$

In this propagation strategy, t_3 is computed as the probabilistic sum of $t_1 \times t_2$ and $d_1 \times d_2$, while d_3 is the probabilistic sum of $t_1 \times d_2$ and $d_1 \times t_2$. The underlying assumption is that a distrusted user is giving the wrong information on purpose. Hence user a trusts user c if a trusted third party tells him to trust c , or, if a distrusted third party tells him to distrust c (i.e. the enemy of your enemy is your friend). Subtracting the distrust degree from the trust degree yields $(t_1 - d_1) \times (t_2 - d_2)$, a distrust propagation scheme put forward in [21]. Below are some examples of propagated trust scores.

	(0.0,0.0)	(0.0,1.0)	(1.0,0.0)	(0.5,0.2)
(0.0,0.0)	(0.0, 0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
(0.0,1.0)	(0.0, 0.0)	(1.0,0.0)	(0.0,0.1)	(0.2,0.5)
(1.0,0.0)	(0.0,0.0)	(0.0,1.0)	(1.0,0.0)	(0.5,0.2)
(0.5,0.2)	(0.0,0.0)	(0.2, 0.5)	(0.5,0.2)	(0.28,0.2)

All these approaches illustrate the fact that, so far, no consensus has yet been reached on how to propagate distrust. Different operators yield different results depending on the interpretation, thus revealing part of the complex problem of choosing an appropriate propagation scheme for the application at hand.

20.2.2.2 Aggregation

Besides propagation, a trust metric must also include an aggregation strategy. After all, in large networks it will often be the case that not one, but several paths lead to the user for whom we want to obtain a trust estimate. When this is the case, the trust estimates that are generated through the different propagation paths must be combined into one aggregated estimation; see for instance the situation depicted in Figure 20.2.

Metrics that only work with trust mostly use classical aggregation operators such as the minimum, maximum, weighted sum, average, or weighted average [1, 3, 19, 28, 43, 44, 55]. The main benefit of weighted operators is that they give us the opportunity to consider some sources (TTPs or propagation paths) as more important than others. In other words, weighted operators provide a way to model the aggregation process more flexibly.

Aggregation of both trust and distrust has not received much attention so far. Only Jøsang et al. have proposed three aggregation operators (called consensus operators) for the subjective logic framework [27]; however, they assume equally important users.

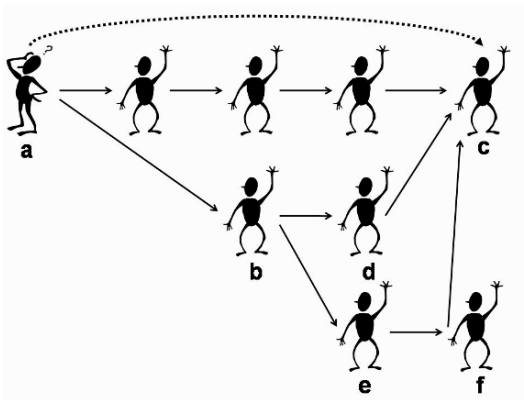


Fig. 20.2: Aggregation example

Note that propagation and aggregation very often must be combined together, and that the final trust estimation might depend on the way this is implemented. Let us take a look at Figure 20.2. There are two ways for user a to obtain a trust estimate about user c from user b . The first possibility is to propagate trust to agent c , i.e., to apply a propagation operator on the trust from b to d and from d to c , and to apply one from b to e , from e to f , and from f to c , and then to aggregate the two propagated trust results. In this scenario, trust is first propagated, and afterwards aggregated (i.e., first propagate then aggregate, or FPTA). A second possibility is to follow the opposite process, i.e., first aggregate and then propagate (FATP). In this scenario, the TTP b must aggregate the estimates that he receives via d and e , and pass on the new estimate to a . It is easy to see that in the latter case the agents/users in the network receive much more responsibility than in the former scenario, and that the trust computation can be done in a distributed manner, without agents having to expose their personal trust and/or distrust information.

Example 20.4. In Figure 20.2 there are three different paths from a to c . Assume that all trust weights on the upper chain are 1, except for the last link which has a trust weight of 0.9. Hence, using multiplication as propagation operator, the propagated trust value resulting from that chain is 0.9. Now, suppose that a trusts b to degree 1, and that b trusts d to the degree 0.5 and e to the degree 0.8. That means that the propagated trust value over the two chains from a to c through b are $1 \times 0.5 \times 0.4 = 0.2$ and $1 \times 0.8 \times 0.6 \times 0.7 \approx 0.34$ respectively. Using the classical average as aggregation operator, FPTA yields a final trust estimate of $(0.9 + 0.2 + 0.34)/3 = 0.48$. On the other hand, if we would allow b to first aggregate the information coming from his trust network, then b would pass the value $(0.2 + 0.34)/2 = 0.27$ on to a . In a FATP strategy, this would then be combined with the information derived through the upper chain in Figure 20.2, leading to an overall final trust estimate of $(0.9 + 0.27)/2 \approx 0.59$.

20.3 Trust-Enhanced Recommender Systems

The second pillar of trust-enhanced recommendation research is the recommender system technology. Recommender systems are often used to accurately estimate the degree to which a particular user (from now on termed the target user) will like a particular item (the target item). These algorithms come in many flavours [2, 54]. Most widely used methods for making recommendations are either content-based (see Chapter 3) or collaborative filtering methods (see Chapter 5). Content-based methods suggest items similar to the ones that the user previously indicated a liking for [56]. Hence, these methods tend to have their scope of recommendations limited to the immediate neighbourhood of the user's past purchase history or rating record for items. For instance, if a customer of a DVD rental service so far has only ordered romantic movies, the system will only be able to recommend related items, and not explore other interests of the user. Recommender systems can be improved

significantly by (additionally) using collaborative filtering, which typically works by identifying users whose tastes are similar to those of the target user (i.e., neighbours) and by computing predictions that are based on the ratings of these neighbours [53].

In the following section, we discuss the weaknesses of such classical recommender systems and illustrate how they can be alleviated by incorporating a trust network among the users of the system. These advanced, trust-based recommendation techniques adhere closest to the collaborative filtering paradigm, in the sense that a recommendation for a target item is based on ratings by other users for that item, rather than on an analysis of the content of the item. A good overview of classic and novel contributions in the field of trust systems, and trust-aware recommender systems in particular, can be found in the book edited by Golbeck [17].

20.3.1 Motivation

Despite significant improvements on recommendation approaches, some important problems still remain. In [37], Massa and Avesani discuss some of the weaknesses of collaborative filtering systems. For instance, users typically rate or experience only a small fraction of the available items, which makes the rating matrix very sparse (since a recommender system often deals with millions of items). For instance, a particular data set from Epinions contains over 1 500 000 reviews that received about 25 000 000 ratings by more than 160 000 different users [61]. Due to this data sparsity, a collaborative filtering algorithm experiences a lot of difficulties when trying to identify good neighbours in the system. Consequently, the quality of the generated recommendations might suffer from this. Moreover, it is also very challenging to generate good recommendations for users that are new to the system (i.e., cold start users), as they have not rated a significant number of items and hence cannot properly be linked with similar users. Thirdly, because recommender systems are widely used in the realm of e-commerce, there is a natural motivation for producers of items (manufacturers, publishers, etc.) to abuse them so that their items are recommended to users more often [67]. For instance, a common ‘copy-profile’ attack consists in copying the ratings of the target user, which results in the system thinking that the adversary is most similar to the target. Finally, Sinha and Swearingen [57, 58] have shown that users prefer more transparent systems, and that people tend to rely more on recommendations from people they trust (‘friends’) than on online recommender systems which generate recommendations based on anonymous people similar to them.

In real life, a person who wants to avoid a bad deal may ask a friend (i.e., someone he trusts) what he thinks about a certain item i . If this friend does not have an opinion about i , he can ask a friend of his, and so on until someone with an opinion about i (i.e., a recommender) has been found. Trust-enhanced recommender systems try to simulate this behaviour, as depicted in Figure 20.3: once a path to a recommender is found, the system can combine that recommender’s judgment with available trust

information (through trust propagation and aggregation) to obtain a personalized recommendation. In this way, a trust network allows to reach more users and more items. In the collaborative filtering setting in Figure 20.4, users a and b will be linked together because they have given similar ratings to certain items (among which i_1), and analogously, b and c can be linked together. Consequently, a prediction of a 's interest in i_2 can be made. But in this scenario there is no link between a (or c) and i_3 or, in other words, there is no way to find out whether i_3 would be a good recommendation for agent a . This situation might change when a trust network has been established among the users of the recommender system.

The solid lines in Figure 20.4 denote trust relations between user a and user b , and between b and user c . While in a scenario without a trust network a collaborative filtering system is not able to generate a prediction about i_3 for user a , this could be solved in the trust-enhanced situation: if a expresses a certain level of trust in b , and b in c , by propagation an indication of a 's trust in c can be obtained. If the outcome would indicate that agent a should highly trust c , then i_3 might become a good recommendation for a , and will be highly ranked among the other recommended items. This simple example illustrates that augmenting a recommender system by including trust relations can help solving the sparsity problem. Moreover, a trust-enhanced system also alleviates the cold start problem: it has been shown that by issuing a few trust statements, compared to a same amount of rating information, the system can generate more, and more accurate, recommendations [35]. Moreover, a web of trust can be used to produce an indication about the trustworthiness of users and as such make the system less vulnerable to malicious insiders: a simple copy-profile attack will only be possible when the target user, or someone who is trusted by the target user, has explicitly indicated that he trusts the adversary to a certain degree. Finally, the functioning of a trust-enhanced system (e.g. the concept of trust propagation) is intuitively more understandable for the users than the classical 'black box' approaches. A nice example is Golbeck's FilmTrust system [16] which asks its users to evaluate their acquaintances based on their movie taste, and accordingly uses that information to generate personalized predictions.

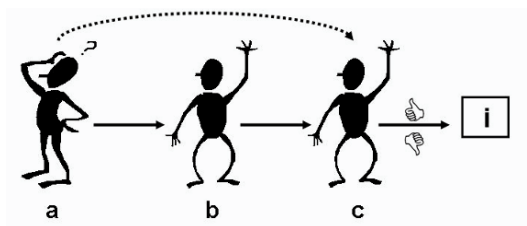


Fig. 20.3: Recommending an item

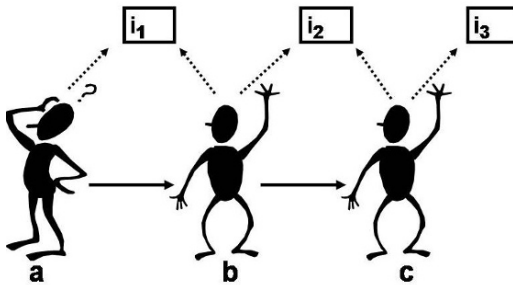


Fig. 20.4: Trust relations in recommender systems

20.3.2 State of the Art

All these examples illustrate that establishing a trust network among the users of a recommender system may contribute to its success. Hence, unsurprisingly, some attempts in this direction have already been made, see for example [15, 23, 30, 37, 46, 49, 51]. Trust-enhanced recommender systems can roughly be divided into two classes, according to the way the trust values are obtained. The first group uses information coming from a trust network that is generated by the direct input of the users, i.e., by explicitly issuing trust statements. Examples can be found in [16, 23, 37]. Such a strategy allows to use trust propagation and aggregation in the network to infer the final trust values that are needed in the recommender algorithm. On the other hand, the second group does not require the user to estimate the trust in his acquaintances. Instead, trust values are computed automatically, for instance based on a user's history of making reliable recommendations [30, 46], or based on transitivity rules for user-to-user similarity [49].

In the behavioral literature, the concept of trust is well defined; see for example Mayer et al.'s framework in which ability, benevolence, integrity and propensity to trust are determined as its key factors [40], or McAllister's work that distinguishes between cognition-based and affect-based trust [41]. However, in the recommendation research area, trust is often used as an umbrella term for a wide range of relationships between people, especially when dealing with automatic computation of trust values. In these cases, trust is being used to denote a variety of concepts, ranging from perceived similarity of tastes, over reputation, to the assessment of a user's competence.

In Section 20.4 we further discuss this in more detail ; in this section, we focus on the basics of both strategies (i.e., mining a trust network and automatic computation of trust values), and illustrate the techniques with representative work in each class.

20.3.2.1 Mining a Trust Network

The most common trust-enhanced recommender strategies ask their users to explicitly issue trust statements about other users. Take for instance Moleskiing [7], a ski mountaineering community site which uses FOAF-files that contain trust information on a scale from 1 to 9 [19], or the e-commerce site Epinions.com which orders reviews based on a trust network that it maintains by asking its users to indicate which members they trust (i.e., their personal web of trust) or distrust (block list). Another well-known example is Golbeck's FilmTrust [16], an online social network combined with a movie rating and review system in which users are asked to evaluate their acquaintances' movie tastes on a scale from 1 to 10.

All these systems exploit the relations in the trust network to determine which opinions or ratings should weigh more or less in the recommendation process. In other words, this group of algorithms uses the trust estimates (obtained by propagation and aggregation) as weights in the decision process. This weighting can be done in several ways. In this section, we focus on the two most commonly used strategies, namely classical weighted average and adaptations of the collaborative filtering mechanism, and illustrate each of them with one well-known state-of-the-art implementation.

Trust-based weighted mean In a recommender system without a trust network, a simple recommendation algorithm that needs to estimate how well a target user will like a target item i can compute the average rating for i by taking into account the ratings $r_{u,i}$ from all the system's users u who are already familiar with i . This baseline recommendation strategy can be refined by computing a *trust-based weighted mean*. In particular, by including trust values $t_{a,u}$ that reflect the degree to which the raters u are trusted, the algorithm allows to differentiate between the sources. In fact, it is only natural to assign more weight to ratings of highly trusted users. The formula is given by Equation (20.1), in which $p_{a,i}$ denotes the predicted rating of target item i for target user a , and R^T represents the set of users who evaluated i and for which the trust value $t_{a,u}$ exceeds a given threshold.

$$p_{a,i} = \frac{\sum_{u \in R^T} t_{a,u} r_{u,i}}{\sum_{u \in R^T} t_{a,u}} \quad (20.1)$$

TidalTrust This formula is at the heart of Golbeck et al.'s recommendation algorithm [15]. The novelty of this algorithm mainly lies in the way the trust estimates $t_{a,u}$ are inferred; a trust metric that they have called *TidalTrust*. In [18], the authors give an overview of the observations that have lead to the development of TidalTrust. In each experiment, they ignored an existing trust relation from a user a to a user c , and focused on all paths that connect a to c . In short, by comparing the propagated trust results from these paths with the original, hidden, trust value, they

noticed that (1) shorter propagation paths yield more accurate trust estimates, and that (2) paths containing higher trust values yield better results too.

Hence, taking into account the first observation, only allowing shorter paths should yield the best results. However, in some cases only a few users will be reachable if a limit is set on the path length. This trade-off is incorporated through a variable path length limit: the shortest path length that is needed to connect the target user with a user u that has rated the item (i.e., a rater) becomes the path depth of the algorithm. Like this, the depth of the breadth-first search varies from one computation to another.

One way of addressing the second observation (higher trust values on the path yield better trust estimates) is to limit the information such that it only comes from the most trusted users. However, every user has its own behaviour for issuing trust values (one user may give the maximum value quite often while another one never does), and in addition, it will often be the case that only a few paths contain the same high trust value. This is why Golbeck et al. opted to incorporate a value that represents the path strength (i.e., the minimum trust rating on a path), and to compute the maximum path strength over all paths leading to the raters. This maximum (max) is then chosen as the minimum trust threshold for participation in the process.

The TidalTrust formula is given by Equation (20.2), in which $WOT^+(a)$ represents the set of users for whom a 's trust statement exceeds the given threshold max . This means that each user in the process computes its trust in another user as a weighted mean, and only takes into account information from users that he has rated at least as high as max .

$$t_{a,u} = \frac{\sum_{v \in WOT^+(a)} t_{a,v} t_{v,u}}{\sum_{v \in WOT^+(a)} t_{a,v}} \quad (20.2)$$

TidalTrust is a recursive algorithm; the trust value $t_{a,u}$ is recursively computed as the weighted mean of trust values $t_{v,u}$ for all TTPs v that are the first link on the shortest path from a to u . The users assure that the maximum path depth is not exceeded by keeping track of the current path length. Note that this algorithm belongs to the class of gradual trust approaches and is an example of a local trust metric.

Golbeck et al. have shown that using trust-based weighted mean in combination with TidalTrust does not necessarily offer a general benefit over computing the average or applying collaborative filtering, but that it does yield significantly more accurate recommendations for users who disagree with the average rating for a specific item (see e.g. [15, 18]).

Trust-based collaborative filtering Whereas Golbeck's approach is an example of a weighted average implementation, another class of trust-enhanced systems is tied more closely to the *collaborative filtering* algorithm. In collaborative filtering, a rating of target item i for target user a can be predicted using a combination of the

ratings of the neighbours of a (similar users) that are already familiar with item i [53]. The classical formula is given by Equation (20.3).

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in R^+} w_{a,u}(r_{u,i} - \bar{r}_u)}{\sum_{u \in R^+} w_{a,u}} \quad (20.3)$$

The unknown rating $p_{a,i}$ for item i and target user a is predicted based on the mean \bar{r}_a of ratings by a for other items, as well as on the ratings $r_{u,i}$ by other users u for i . The formula also takes into account the similarity $w_{a,u}$ between users a and u , usually calculated as Pearson's Correlation Coefficient (PCC) [22]. In practice, most often only users with a positive correlation $w_{a,u}$ who have rated i are considered. We denote this set by R^+ . However, instead of a PCC-based computation of the weights, one can also infer the weights through the relations of the target user in the trust network (again through propagation and aggregation); see Formula (20.4) which adapts Formula (20.3) by replacing the PCC weights $w_{a,u}$ by the trust values $t_{a,u}$. This strategy is also supported by the fact that trust and similarity are correlated, as shown in [69].

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in R^T} t_{a,u}(r_{u,i} - \bar{r}_u)}{\sum_{u \in R^T} t_{a,u}} \quad (20.4)$$

We call this alternative *trust-based collaborative filtering*. Note that, because the weights are not equal to the PCC, this procedure can produce out of bounds results. When this is the case, $p_{a,i}$ is rounded to the nearest possible rating.

MoleTrust Formula (20.4) is at the basis of Massa et al.'s recommendation algorithm which incorporates a new trust metric, called *MoleTrust* [38]. This metric consists of two phases. In the first stage, cycles in the trust network are removed, while the second stage includes the actual trust computation. Since it is often the case that a large number of trust propagations must be executed in trust experiments (think e.g. of the large test sets from Epinions.com), it is much more efficient to remove trust cycles beforehand, so that every user only needs to be visited once for obtaining a trust prediction.

The removing of the cycles transforms the original trust network into a directed acyclic graph, and hence the trust prediction for $t_{a,u}$ can be obtained by performing a simple graph walk: first the trust of the users at distance 1 is computed (i.e., direct trust information), then the trust of the users at distance 2, etc. Note that because of the acyclic nature of the graph, the trust value of a user at distance x only depends on the already computed trust values of the users at distance $x - 1$.

The trust of the users at distance 2 or more is calculated in a way similar to Golbeck et al.'s algorithm, i.e. formula (20.2). However, the details of the breadth-first implementation differ significantly. In TidalTrust, a user u is added to $WOT^+(a)$ only if he is on a shortest path from target user a to target item i . On the other hand,

Table 20.2: Characteristic features of two state-of-the-art recommendation approaches that mine a trust network to predict a rating for target user a and target item i , based on ratings of other users u for i

	TidalTrust	MoleTrust
propagation	multiplication	multiplication
aggregation	trust-based weighted mean (20.2)	trust-based weighted mean (20.2)
maximum length of propagation path	dynamic (shortest path)	static (horizon)
trust threshold	dynamic (strongest chain)	static
entry requirement for TTP v in propagation process	v is on a shortest path from a to u	v is on a path from a to u within the horizon
prediction of rating	trust-based weighted mean (20.1)	trust-based collaborative filtering (20.4)

in MoleTrust, $WOT^+(a)$ includes all users who have rated the target item and that can be reached through a direct or propagated trust relation. But trust is not computed for all eternity: before the computation begins, one must assign a value d to the ‘propagation horizon’ parameter. Like this, only users who are reachable within distance d are taken into account. Another important input parameter of MoleTrust is the trust threshold for participation in the process (unlike the dynamic max value in TidalTrust), which is for example set to 0.6 (on a scale from 0 to 1) in the experiments reported in [38].

Note that, analogous to TidalTrust, MoleTrust belongs to the class of gradual local trust metrics. In their experiments, Massa and Avesani have illustrated that MoleTrust provides better trust estimates than global trust metrics such as eBay’s⁶, especially when it comes down to estimating the trust in controversial users (who are trusted by one group and distrusted by another) [38]. They also showed that MoleTrust yields more accurate predictions for cold start users, compared to a classical collaborative filtering system [35, 36].

Golbeck’s and Massa’s approach are two typical examples of trust-enhanced recommender techniques that use explicit trust information. Table 20.2 summarizes their most prominent characteristics. Other recommendation approaches that also mine a trust network can be found in, among others, [23, 63].

20.3.2.2 Automatic Trust Generation

The algorithms discussed in the previous section require explicit trust input from the users. As a consequence, the applications that use such an algorithm must provide

⁶ www.ebay.com

a means to obtain the necessary information; think e.g. of FilmTrust or Moleskiing. However, this might not always be possible or feasible. In such cases, methods that automatically infer trust estimates, without needing explicit trust information, might be a better solution. An example of such a system can be found in [47].

Most commonly, these approaches base their trust generation mechanism on the past rating behaviour of the users in the system. More specifically, deciding to what degree a particular user should participate in the recommendation process is influenced by his history of delivering accurate recommendations. Let us exemplify this with the well-known approach of O’Donovan et al. [46].

Profile- and item-level trust Our intuition tells us that a user who has made a lot of good recommendations in the past can be viewed as more trustworthy than other users who performed less well. To be able to select the most trustworthy users in the system, O’Donovan introduced two trust metrics, viz. *profile-level* and *item-level trust*, reflecting the general trustworthiness of a particular user u , and the trustworthiness of a user u with respect to a particular item i , respectively. Both trust metrics need to compute the correctness of u ’s recommendations for the target user a . In particular, a prediction $p_{a,i}$ that is generated only by information coming from u (hence u is the sole recommender) is considered correct if $p_{a,i}$ is within ϵ of a ’s actual rating $r_{a,i}$.

The profile-level trust t_u^P for u is then defined as the percentage of correct recommendations that u contributed. Remark that this is a very general trust measure; in practice it will often occur that u performs better in recommending a set of specific items. To this aim, O’Donovan also proposed the more fine-grained item-level trust t_u^i , which measures the percentage of recommendations for item i that were correct. Hence, in such automated approaches, trust values are not generated via trust propagation and aggregation, but are based on the ratings that were given in the past. Remark that O’Donovan’s methods are global trust metrics. The way the values are obtained can be seen as probabilistic.

Trust-based filtering Similar to other trust-enhanced techniques, the values that are obtained through the trust metric are used as weights in the recommendation process. Just like Massa, O’Donovan et al. focus on trust-based adaptations of collaborative filtering. In [46] they investigate several options, such as combining the obtained trust values with PCC information. An alternative to this scheme is to use trust values as a filter, so that only the most trustworthy neighbours participate in the recommendation process. This strategy is called *trust-based filtering*, see Formula (20.5) in which $w_{a,u}$ denotes the PCC and $R^{T+} = R^T \cap R^+$.

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in R^{T+}} w_{a,u} (r_{u,i} - \bar{r}_u)}{\sum_{u \in R^{T+}} w_{a,u}} \tag{20.5}$$

In other words, only users whose item/profile-level trust exceeds a certain threshold, and that have a positive correlation with a , are taken into account.

In [46], O’Donovan and Smyth showed that trust-based filtering achieves better accuracy than collaborative filtering in terms of average errors. Moreover, the algorithm based on profile-level trust yields lower errors than collaborative filtering in nearly 70% of all prediction cases.

O’Donovan’s method is a representative example in the group of strategies that use automatic trust generation. A related approach can be found in [30], which works with an utilitarian measure instead of a binary correctness function.

20.3.3 Empirical Comparison

One question that stands out is which of the state-of-the-art approaches discussed above performs best in practice. Basically, so far, researchers in the trust-based recommender field introduced their own new algorithms and evaluated these on their own applications and/or data sets, without including a comparison of other trust-enhanced approaches based on the same data set/application. Therefore, in the remainder of this section, we provide a head-to-head comparison of the performance that the previously discussed trust-enhanced techniques can achieve on one and the same data set. We focus on Golbeck’s trust-based weighted mean with TidalTrust (Eq. (20.1)), Massa’s trust-based collaborative filtering with MoleTrust (Eq. (20.4)), and O’Donovan’s trust-based filtering (Eq. (20.5)). Since our goal is to compare all techniques on the same data sets and to investigate the influence of trust propagation, we have chosen not to implement O’Donovan’s automatic trust generation strategy, but to mine the same trust network as the other two strategies. Although O’Donovan et al. do not use trust propagation in their experiments [46], it is of course possible to do so. Since there is no explicit use of trust values in (20.5), we only need to specify how propagation enlarges R^{T+} (see below).

20.3.3.1 Data Sets

The data sets we use in our experiments are obtained from Epinions.com, a popular e-commerce site where users can write reviews about consumer products and assign a rating to the products and the reviews. Two Epinions data sets are often used for experimenting with trust-enhanced recommender systems. The first one was collected by Massa and Bhattacharjee [39] in a 5-week crawl and contains 139 738 products that are rated by 49 290 users in total; the consumer products are rated on a scale from 1 to 5. The second data set was compiled by Guha et al. [21]: this large data set contains 1 560 144 reviews that received 25 170 637 ratings by 163 634 different users. The reviews are evaluated by assigning a helpfulness rating which ranges from ‘not helpful’ (1/5) to ‘most helpful’ (5/5). This data set does not contain any information about consumer products and product ratings, but works with reviews and review ratings instead; in other words, for this data set, we discuss and evaluate

a ‘review recommender system’. Hence, in the context of Guha’s set, an item denotes a review of consumer goods, whereas for the crawled data set an item denotes a consumer product.

In our experiments we focus on the number of recommendations/predictions that can be generated by the systems and on the prediction errors, for random items as well as controversial items. The latter are the most challenging items for a recommender system, since it is much harder to predict a score for an item that has received a variety of high and low scores, reflecting disagreement about the item. More than in any other case, a recommendation for a user needs to be truly personalized when the target item under consideration is controversial; i.e., when an item has both ‘ardent supporters’ and ‘motivated adversaries’, with no clear majority in either group. In [63], Victor et al. explain why classical standard deviation is not sufficient to detect the true controversial items in a data set, and propose a new measure to define the controversiality level of a particular item. Their methodology leads to 1 416 controversial items in Guha’s data set, and 266 in Massa’s data set. We refer to [63] for more details about the controversiality computation. To compare the performance achieved for controversial items (CIs) with the performance that can be obtained in general, we also present the average coverage and accuracy for 1 416 and 266 randomly selected ‘popular’ items (RIs) (that have been evaluated at least 20 times, analogous to the controversial items).

Epinions allows users to evaluate other users based on the quality of their reviews, and to provide trust and distrust evaluations in addition to ratings. The fact that both data sets contain explicit trust information from the users makes them very appropriate to study issues in trust-enhanced recommender systems. Users can evaluate other users by including them in their WOT (i.e. a list of reviewers whose reviews and ratings were consistently found to be valuable⁷), or by putting them in their block list (a list of authors whose reviews were consistently found to be offensive, inaccurate or low quality⁷, thus indicating distrust). In Guha’s data set, the trust evaluations make up an Epinions WOT graph consisting of 114 222 users and 717 129 non self-referring trust relations. Massa’s data set contains information on 49 288 users who issued or received 487 003 trust statements in total.

Note that the data sets only contain binary trust values, hence in our experiments $t_{a,u}$ in (20.1), (20.4) and (20.5) can take on the values 0 (absence of trust) and 1 (full presence) only. This limitation leads to alterations of some of the trust-based algorithms; e.g., Formula (20.1) reduces to the classical average. For simplicity, we only consider one-step propagation in this paper. This means that for the propagated versions of (20.4) and (20.5), we consider chains of length 1 and 2, whereas for (20.2) we only consider chains of length 2 when there are no shorter chains available. These two simplifications put a restriction on our empirical comparison, because we cannot analyse the algorithms exactly as they were meant/designed to be.

⁷ See www.epinions.com/help/faq/

20.3.3.2 Coverage

Coverage refers to the number of target user - target item pairs for which a prediction can be generated. A classical way to measure the coverage of a recommender system is by using the leave-one-out method, which consists of hiding a rating and trying to predict its hidden value. The coverage of a specific algorithm then refers to the amount of computable predictions $p_{a,i}$ versus the number of leave-one-out experiments to perform (i.e., the number of ratings available in the data set). For Formula (20.3) we call $p_{a,i}$ computable if there is at least one user u for which the PCC $w_{a,u}$ can be calculated, while for Formulas (20.1) and (20.4) a computable $p_{a,i}$ means that there is at least one user u for which the (propagated) trust estimate $t_{a,u}$ can be calculated. Finally, for Formula (20.5), predictions are possible when at least one user u is found for which the PCC can be computed and $t_{a,u}$ is 1.

Table 20.3 shows the coverage (% COV) for controversial items (CIs) and randomly selected items (RIs) in Guha's and Massa's data sets. The first four rows cover baseline strategies (B1)–(B4). The first baseline strategy is a system that always predicts 5/5 (B1), since this is the predominant score for items in Epinions. The second system computes the average received rating for the target item (B2), while the third one yields the average rating given by target user a (B3). The latter method will score well in a system where the users have a rating behaviour with little variation. Finally, the last baseline returns a random helpfulness score between 1 and 5 (B4).

In general, baselines (B1), (B2) and (B4) achieve maximal coverage for both controversial and randomly selected items: (B1) and (B4) do not rely on any additional (trust or PCC) information, and since the items in our experiments are evaluated at least 20 times, it is always possible to compute (B2). With (B3), in those cases in which the target user rated only one item, his average rating is lacking, so a prediction cannot be generated.

For the other algorithms in Table 20.3, the numbers in the first column refer to the corresponding recommendation formulas given above. For the trust-enhanced approaches, we distinguish between experiments that did not use propagated trust information (higher rows) and those that did (bottom rows). We only consider one-step propagation: for (P1) and (P4), we maintained the propagation strategy used in TidalTrust and MoleTrust⁸ respectively, while for (P5) we added a user to R^T if he belongs to the WOT of the target user a , or is directly trusted by a WOT member of a .

Without propagation, it is clear that the coverage of the collaborative filtering algorithm is superior to that of the others, and approaches the maximal value. This is due to the fact that PCC information is, in general, more readily available than direct trust information: there are normally more users for which a positive correlation with the target user a can be computed than users in a 's WOT. On the other hand, trust-based filtering (20.5), which also uses PCC weights, is the most demanding strategy because it requires users in a 's WOT who have already rated two other

⁸ Note that we incorporate Massa et al.'s horizon-based strategy for binary trust settings [35].

Table 20.3: Performance trust-based recommender algorithms

ALGORITHM	Giuba et al.'s data set				Massa et al.'s data set				
	Controversial items (CIs)		Randomly selected items (RIs)		Controversial items (CIs)		Randomly selected items (RIs)		
	% COV	MAE	RMSE	% COV	MAE	RMSE	% COV	MAE	RMSE
(B1) Base: score 5	100	1.45	1.96	100	0.16	0.51	100	1.94	2.46
(B2) Base: average score for item	100	1.25	1.34	100	0.18	0.40	100	1.35	1.51
(B3) Base: average score of user	99	1.23	1.58	100	0.36	0.50	98	1.43	1.78
(B4) Base: random score	100	1.61	2.02	100	1.92	2.37	100	1.66	2.08
(20.3) Collaborative filtering	94	0.96	1.13	98	0.19	0.38	81	1.34	1.58
(20.1) Trust-based weighted mean	63	0.86	1.20	89	0.13	0.35	41	1.33	1.70
(20.4) Trust-based collaborative filtering	63	0.87	1.16	89	0.17	0.35	40	1.32	1.65
(20.5) Trust-based filtering	60	0.86	1.16	86	0.16	0.36	25	1.35	1.71
(P1) Propagated Trust-based weighted mean	88	0.91	1.22	97	0.15	0.38	76	1.37	1.69
(P4) Propagated Trust-based collaborative filtering	88	0.99	1.16	97	0.19	0.37	76	1.32	1.56
(P5) Propagated Trust-based filtering	84	0.94	1.13	96	0.18	0.36	57	1.36	1.64

items in common with a (otherwise the PCC can not be computed). In between these extremes, the coverage for TidalTrust (20.1) is a bit higher than that of MoleTrust (20.4) because the latter can only generate predictions for target users who have rated at least two items, otherwise the average rating for the target user can not be computed).

This ranking of approaches in terms of coverage still applies when propagated trust information is taken into account, but note that the difference with collaborative filtering has shrunk considerably. In particular, thanks to trust propagation, the coverage increases with about 25% (10%) for controversial (randomly selected) items in the first set, and more than 30% in the second set.

For Guha's data set, the coverage results for controversial items are significantly lower than those for randomly selected items. This is due to the fact that, on average, controversial items in this data set receive less ratings than randomly selected items, which yields less leave-one-out experiments per item, but also a smaller chance that such an item was rated by a user with whom the target user a has a positive PCC, or by a user that a trusts. This also explains the lower coverage results for the nontrivial recommendation strategies. The same observations cannot be made for Massa's data set: on average, the CIs receive more ratings than the RIs (21 131 vs. 12 741). This explains the somewhat lower coverage performance of the algorithms on the random item set.

Also remark that the coverage results for Massa's data set are significantly lower in general than those for Guha's; (20.1), (20.4) and (20.5) achieve a coverage that is at least 20% worse. Users in Guha's data set rate much more items than users in Massa's data set, which yields less users who have rated the same items, i.e., neighbours (through trust or PCC) that are needed in the computation.

20.3.3.3 Accuracy

As with coverage, the accuracy of a recommender system is typically assessed by using the leave-one-out method, more in particular by determining the deviation between the hiding ratings and the predicted ratings. In particular, we use two well-known measures, viz. mean absolute error (MAE) and root mean squared error (RMSE) [22]. The first measure considers every error of equal value, while the latter one emphasizes larger errors. Since reviews and products are rated on a scale from 1 to 5, the extreme values that MAE and RMSE can reach are 0 and 4. Even small improvements in RMSE are considered valuable in the context of recommender systems. For example the Netflix prize competition⁹ offers a \$1 000 000 reward for a reduction of the RMSE by 10%.

The MAE and RMSE reported in Table 20.3 is overall higher for the controversial items than for the randomly selected items. In other words, generating good predictions for controversial items is much harder than for randomly chosen items.

⁹ See <http://www.netflixprize.com/>

This applies to all the algorithms, but most clearly to the baseline strategies (except (B4)). While in Massa's data set all algorithms adjust themselves in more or less the same way, in Guha's data set (B1) and (B2) clearly experience more difficulties when generating predictions for controversial items: whereas for random items they are competitive with collaborative filtering and the trust-enhanced approaches, their MAE and RMSE on the controversial item set increase with more than 1 on the rating scale from 1 to 5.

Also note that it is more difficult to generate good recommendations in Massa's data set than in Guha's, for controversial as well as random items. This is due to the higher inherent controversiality level of the former data set.

When focusing on the MAE of the non-baseline approaches for controversial items, we notice that, without propagation, trust-enhanced approaches all yield better results than collaborative filtering¹⁰ (with one exception for trust-based filtering on Massa's CIs), which is in accordance with the observations made in [15, 36]. This can be attributed to the accuracy/coverage trade-off: a coverage increase is usually at the expense of accuracy, and vice versa. It also becomes clear when taking into account trust propagation: as the coverage of the trust-enhanced algorithms nears that of the collaborative filtering algorithm, so do the MAEs.

However, the RMSEs give us a different picture. On the controversial item sets, the RMSE of the trust-enhanced approaches is generally higher than that of collaborative filtering, which does not always occur on the random sets; recall that a higher RMSE means that more large prediction errors occur. One possible explanation for this is the fact that, for controversial items, the set R^T of trusted acquaintances that have rated the target item is too small (e.g., contains only 1 user), and in particular smaller than R^+ . This hypothesis is also supported by the fact that with trust propagation (which enlarges R^T) RMSEs rise at a slower rate than the corresponding MAEs. Moreover, it is often the case that the propagated algorithms achieve lower RMSEs than their unpropagated counterparts, see e.g. the results on controversial items in Massa's data set.

20.3.3.4 Conclusion

The experiments on both Epinions data sets, each with their own characteristics, endorse the same conclusions. For random items, intelligent strategies such as collaborative filtering and trust-based algorithms barely outperform the baselines. However, the baselines fall short in generating good recommendations for controversial items. Trust-enhanced systems perform better in this respect, although there is certainly still room for improvement; remember the higher RMSEs and the fact that trust-based approaches on Massa's CIs yield no visible improvements over collaborative filtering. These findings call for further research on improving the algorithms and

¹⁰ Note that all the MAE improvements on Guha's data set are statistically significant ($p < 0.000$).

identifying specific cases where trust approaches are effective (think e.g. of Massa et al.'s results for cold start users).

The coverage and accuracy results show no clear winner among the three state-of-the-art trust-enhanced strategies proposed by Golbeck et al., Massa et al., and O'Donovan et al. Trust-based collaborative filtering seems to score best on Massa's data set, while trust-based weighted mean and trust-based filtering achieve the best accuracy on Guha's data set; this trend is also confirmed by the results obtained by propagation.

The two data sets contain rating information and trust information, which makes them popular in trust-enhanced recommender experiments. However, they have one shortcoming: the trust values in Epinions are binary, making it impossible to investigate all aspects of the algorithms we discussed in this chapter, since a lot of the existing trust-based approaches are based on the assumption that trust is a gradual concept. Unfortunately, there are no such data sets publicly available.

20.4 Recent Developments and Open Challenges

In the previous sections we have covered the basics of trust modeling, trust metrics, and trust-enhanced recommender systems. In this section, we want to give the reader a foretaste of new directions in the research area of trust-based recommendation systems. This is certainly not meant to be a complete overview, but rather a selection of recent developments in the field. In particular, we will briefly discuss the following issues: alleviating the trust-based cold start problem, visualization of trust-enhanced recommender systems, theoretical foundations for trust-based research, and involving distrust in the recommendation process.

Massa and Avesani have shown that the user cold start problem in classical recommender systems can be alleviated by including a trust network among its users. They demonstrated that, for new users, it is more beneficial to issue a few trust statements (compared to rating some items) in order to get good recommendations from the system [35]. However, Victor et al. have shown that cold start users in the classical sense (who rated only a few items) are very often cold start users in the trust sense as well [61]. Hence, new users must be encouraged to connect to other users to expand the trust network as soon as possible, but choosing whom to connect to is often a difficult task. Given the impact this choice has on the delivered recommendations, it is critical to guide newcomers through this early stage connection process. In [61] this problem is tackled by identifying three types of key figures in the recommender system's network, viz. frequent raters, mavens and connectors. The authors show that, for a cold start user, connecting to one of the identified key figures is much more advantageous than including a randomly chosen user, with respect to coverage as well as accuracy of the generated recommendations.

Remark that these connection guidance issues link up with the broader problem

of trust bootstrapping, i.e., the problem of how to establish initial trust relations in the network. O'Donovan, too, addresses this problem, but in a very different way: he introduces PeerChooser, a new procedure to visualize a trust-based collaborative filtering recommender system [45]. More specifically, PeerChooser visualizes both information coming from the traditional similarity measure PCC, and information coming from the underlying trust-space generated from the rating data (remember O'Donovan's profile- and item-level trust [46]). One of the main features of the system is its possibility to extract trust information on the fly, directly from the user at recommendation time. This is done by moving specific icons (representing users in the system) on an interactive interface. In this way, the user can indicate his mood and preferences, thereby actively providing real-time trust information.

There are also other ways to establish trust relations when the information is not explicitly given by the users. Several sources of social data can be consulted, such as online friend and business networks (think e.g. of Facebook or LinkedIn), e-mail communication, reputation systems, etc. In the recommender system literature, they are often lumped together and collectively referred to as trust, although they map onto different concepts: behavioral theory clearly draws a distinction between homophily or cognitive similarity (similarity between people/tastes/etc.), social capital (reputation, opinion leadership), tie strength (in terms of relationship duration and interaction frequency), and trust (see e.g. [40, 41]). Potentially all these social data sources could be incorporated into a (trust-enhanced) recommender system, but so far not much research has been conducted to find out which ones will be most useful [4], and whether these sources would provide similar results as the classical trust-based recommendation approaches discussed in this chapter. In [5], Arazy et al. embark upon this problem and argue that the design of social recommenders should be grounded in theory, rather than making ad hoc design choices as is often the case in current algorithms.

Another recent research direction of a completely different nature is the investigation of the potential of distrust in trust-based recommender systems. Whereas in the trust modeling domain only a few attempts have been made to incorporate distrust, in the recommender domain this is even less so. This is due to several reasons, the most important ones being that very few data sets containing distrust information are available, and that there is no general consensus yet about how to propagate it and to use it for recommendation purposes. A first experimental evaluation of the effects of involving distrust in the recommendation process is reported in [64]. In this paper, three distrust strategies are investigated, viz. distrust as an indicator to reverse deviations, distrust as a filter for neighbour selection, and distrust as a debugger of a web of trust. The first two strategies are based on the rationale that trust can be used to select similar users (neighbours) in collaborative filtering systems, while the latter strategy has been suggested by various researchers in the field, see e.g. [20, 68]. The results indicate that the first technique is not the line to take. Distrust as a filter and/or debugger looks more promising, but it is clear that much work

remains to be done in this nascent research area before one can come to a more precise conclusion.

20.5 Conclusions

In this chapter we have given an introduction to the research area of trust modeling, and illustrated how trust networks can improve the performance of classical recommender systems. We discussed several state-of-the-art implementations of these so-called trust-enhanced recommender strategies, and provided an experimental evaluation of their performance on two data sets from Epinions.com. This comparison in terms of coverage and accuracy did not yield any clear winner, but did show that each of the algorithms has its own merits.

Recommender applications that maintain a social trust network among their users can benefit from trust propagation strategies that have proven to yield a surplus value, whereas in cases where it is not immediately possible to collect explicit trust statements, methods that are able to automatically compute trust values seem the most ideal solution. Of course, these strategies could not have been devised without the appropriate data sets and/or applications to experiment with.

In fact, one of the main difficulties in the trust-enhanced recommender research domain is the lack of publicly available and suitable test data. Hence, it is our hope that in the near future more such data and applications become within reach of every researcher in need of it, and we strongly believe that this will attract and inspire even more people, thereby stimulating the research in this thriving area of trust-based recommendation.

References

1. Abdul-Rahman, A., Hailes, S.: Supporting trust in virtual communities. In: Proc. of the 33rd Hawaii International Conference on System Sciences, pp. 1769-1777 (2000)
2. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17**, 734-749 (2005)
3. Almenárez, F., Marín, A., Campo, C., García, C.: PTM: A pervasive trust management model for dynamic open environments. In: Proc. of the First Workshop on Pervasive Security, Privacy and Trust, in conjunction with Mobiquitous (2004)
4. Arazy, O., Elsane, I., Shapira, B., Kumar, N.: Social relationships in recommender systems. In: Proc. of the 17th Workshop on Information Technologies & Systems (2007)
5. Arazy, O., Kumar, N., Shapira, B.: Improving social recommender systems. *IT Professional* May/June, 31-37 (2009)
6. Artz, D., Gil, Y.: A survey of trust in computer science and the semantic web. *Journal of Web Semantics* **5**, 58-71 (2007)
7. Avesani, P., Massa, P., Tiella, R.: Moleskiing.it: a trust-aware recommender system for ski mountaineering. *International Journal for Infonomics* (2005)

8. Cacioppo, J., Berntson, G.: Relationship between attitudes and evaluative space: a critical review, with emphasis on the separability of positive and negative substrates. *Psychological Bulletin* **115**, 401–423 (1994)
9. Constantinople, A.: An eriksonian measure of personality development in college students. *Development Psychology* **1**, 357–372 (1969)
10. Cofta, P.: Distrust. In: Proc. of the International Conference on Electronic Commerce, pp. 250–258 (2006)
11. De Cock, M., Pinheiro da Silva, P.: A many-valued representation and propagation of trust and distrust. In: Bloch, I., Petrosino, A., Tettamanzi, A. (eds.) *Lecture Notes in Computer Science* 3849, pp. 108–113 (2006)
12. Falcone, R., Pezzulo, G., Castelfranchi, C.: A fuzzy approach to a belief-based trust computation. In: Eder, J., Haav, H.-M., Kalja, A., Penjam, J. (eds.) *Lecture Notes in Artificial Intelligence* 2631, pp. 73–86 (2003)
13. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G.: Modeling the impact of trust and distrust in agent networks. In: Proc. of the Third Workshop on Agent-oriented Information Systems, pp. 45–58 (2001)
14. Ginsberg, M.: Multi-valued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence* **4**, 265–316 (1988)
15. Golbeck, J.: Computing and applying trust in web-based social networks. PhD thesis (2005)
16. Golbeck, J.: Generating predictive movie ratings from trust in social networks. In: Stølen, K., Winsborough, W.H., Martinelli, F., Massacci, F. (eds.) *Lecture Notes in Computer Science* 3986, pp. 93–104 (2006)
17. Golbeck, J.: *Computing with Social Trust*. Springer, London (2009)
18. Golbeck, J., Mannes, A.: Using trust and provenance for content filtering on the semantic web. In: Proc. of the WWW06 Models of Trust for the Web Workshop (2006)
19. Golbeck, J., Parsia, B., Hendler, J.: Trust networks on the semantic web. In: Klusch, M., Omicini, A., Ossowski, S., Laamanen, H. (eds.) *Lecture Notes in Artificial Intelligence* 2782, pp. 238–249 (2003)
20. Guha, R.: Open rating systems. Technical report, Stanford Knowledge Systems Laboratory (2003)
21. Guha, R., Kumar, R., Raghavan, P., Tomkins, A.: Propagation of trust and distrust. In: Proc. of the World Wide Web Conference, pp. 403–412 (2004)
22. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems* **22**, 5–53 (2004)
23. Hess, C., Schiedler, C.: Trust-based recommendations for documents. *AI Communications* **21**, 145–153 (2008)
24. Jøsang, A.: A logic for uncertain probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **9**, 279–311 (2001).
25. Jøsang, A., Knapskog, S.: A metric for trusted systems. In: Proc. of the National Computer Security Conference, pp. 16–29 (1998)
26. Jøsang, A., Gray, E., Kinatader, M.: Simplification and analysis of transitive trust networks. *Web Intelligence and Agent Systems* **4**, 139–161 (2006)
27. Jøsang, A., Marsh, S., Pope, S.: Exploring different types of trust propagation. In: Stølen, K., Winsborough, W.H., Martinelli, F., Massacci, F. (eds.) *Lecture Notes in Computer Science* 3986, pp. 179–192 (2006)
28. Kamvar, S., Schlosser, M., Garcia-Molina, H.: The eigentrust algorithm for reputation management in P2P networks. In: Proc. of the World Wide Web Conference, pp. 640–651 (2003)
29. Klir, G., Yuan, B.: *Fuzzy sets and systems: theory and applications*. Prentice Hall PTR, New Jersey (1995)
30. Lathia, N., Hailes, S., Capra, L.: Trust-based collaborative filtering. In: Karabulut, Y., Mitchell, J., Herrmann, P., Damsgaard Jensen, C. (eds.) *IFIP International Federation for Information Processing* **263**, 119–134 (2008)
31. Lesani, M., Bagheri, S.: Applying and inferring fuzzy trust in semantic web social networks. In: Kodé, M.T., Lemire, D. (eds.) *Semantic Web and Beyond* 2, pp. 23–43 (2006)

32. Levien, R.: Attack-resistant trust metrics. In: Golbeck, J. (ed.) *Computing With Social Trust*, pp. 121-132 (2009)
33. Lewicki, R., McAllister, D., Bies, R.: Trust and distrust: new relationships and realities. *Academy of Management Review* **23**, 438–458 (1998)
34. Marsh, S., Briggs, P.: Examining trust, forgiveness and regret as computational concepts. In: Golbeck, J. (ed.) *Computing With Social Trust*, pp. 9-43 (2009)
35. Massa, P., Avesani, P.: Trust-aware collaborative filtering for recommender systems. In: *Proc. of the Federated International Conference On The Move to Meaningful Internet*, pp. 492-508 (2004)
36. Massa, P., Avesani, P.: Trust-aware recommender systems. In: *Proc. of ACM Recommender Systems*, pp. 17-24 (2007)
37. Massa, P., Avesani, P.: Trust metrics in recommender systems. In: Golbeck, J. (ed.) *Computing with Social Trust*, pp. 259-285 (2009)
38. Massa, P., Avesani, P.: Trust metrics on controversial users: balancing between tyranny of the majority and echo chambers. *International Journal on Semantic Web and Information Systems* **3**, 39–64 (2007)
39. Massa, P., Bhattacharjee, B.: Using trust in recommender systems: an experimental analysis. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) *Lecture Notes in Computer Science 2995*, pp. 221-235 (2004)
40. Mayer, R., Davis, J., Schoorman, D.: An integrative model of organizational trust. *The Academy of Management Review* **20**, 709–734 (1995)
41. McAllister, D.: Affect- and cognition-based trust as foundations for interpersonal cooperation in organizations. *The Academy of Management Journal* **38**, 24–59 (1995)
42. Moskovitch, R., Elovici, Y., Rokach, L.: Detection of unknown computer worms based on behavioral classification of the host, *Computational Statistics and Data Analysis*, 52(9):4544–4566 (2008)
43. Mui, L., Mohtashemi, M., Halberstadt, A.: A computational model of trust and reputation. In: *Proc. of the 35th Hawaii International Conference on System Sciences*, pp. 2431-2439 (2002)
44. Noh, S.: Calculating trust using aggregation rules in social networks. In: Xiao, B., Yang, L., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) *Lecture Notes in Computer Science 4610*, pp. 361-371 (2007)
45. O'Donovan, J.: Capturing trust in social web applications. In: Golbeck, J. (ed.) *Computing With Social Trust*, pp. 213-257 (2009)
46. O'Donovan, J., Smyth, B.: Trust in recommender systems. In: *Proc. of the 10th International Conference on Intelligent User Interfaces*, pp. 167-174 (2005)
47. O'Donovan, J., Smyth, B.: Mining trust values from recommendation errors. *International Journal on Artificial Intelligence Tools* **15**, 945–962 (2006)
48. O'Reilly, T.: What is web 2.0. Available at <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html> (2005)
49. Papagelis, M., Plexousakis, D., Kutsuras, T.: Alleviating the sparsity problem of collaborative filtering using trust inferences. In: Herrmann, P., Issarny, V., Shiu, S. (eds.) *Lecture Notes in Computer Science 3477*, pp. 224-239 (2005)
50. Petty, R., Wegener, D., Fabrigar, L.: Attitudes and attitude change. *Annual Review of Psychology* **48**, 609–647 (1997)
51. Pitsilis, G., Marshall, L.: A trust-enabled P2P recommender system. In: *Proc. of the 15th Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pp. 59-64 (2006)
52. Priester, J., Petty, R.: The gradual threshold model of ambivalence: relating the positive and negative bases of attitudes to subjective ambivalence. *Journal of Personality and Social Psychology* **71**, 431–449 (1996)
53. Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., Riedl, J.: Grouplens: An open architecture for collaborative filtering of netnews. In: *Proc. of Computer Supported Cooperative Work*, pp. 175-186 (1994)

54. Resnick, P., Varian, H.R.: Recommender systems. *Communications of the ACM* **40**, 56–58 (1997)
55. Richardson, M., Agrawal, R., Domingos, P.: Trust management for the semantic web. In: *Proc. of the Second International Semantic Web Conference*, pp. 351–368 (2003)
56. Schafer, J., Konstan, J., Riedl, J.: E-commerce recommendation applications. *Data Mining and Knowledge Discovery* **5**, 115–153 (2001)
57. Sinha, R., Swearingen, K.: Comparing recommendations made by online systems and friends. *Proc. of the DELOS-NSF Workshop on Personalisation and Recommender Systems in Digital Libraries* (2001)
58. Swearingen, K., Sinha, R.: Beyond algorithms: an HCI perspective on recommender systems. *Proc. of SIGIR Workshop on Recommender Systems* (2001)
59. Tang, W., Ma, Y., Chen, Z.: Managing trust in peer-to-peer networks. *Journal of Digital Information Management* **3**, 58–63 (2005)
60. Victor, P., De Cock, M., Cornelis, C., Pinheiro da Silva, P.: Towards a provenance-preserving trust model in agent networks. In: *Proc. of Models of Trust for the Web WWW2006 Workshop* (2006)
61. Victor, P., Cornelis, C., De Cock, M., Teredesai, A.M.: Key figure impact in trust-enhanced recommender systems. *AI Communications* **21**, 127–143 (2008)
62. Victor, P., Cornelis, C., De Cock, M., Pinheiro da Silva, P.: Gradual trust and distrust in recommender systems. *Fuzzy Sets and Systems* **160** 1367–1382 (2009)
63. Victor, P., Cornelis, C., De Cock, M., Teredesai, A.M.: A comparative analysis of trust-enhanced recommenders for controversial items. In: *Proc. of the International AAI Conference on Weblogs and Social Media*, pp. 342–345 (2009)
64. Victor, P., Cornelis, C., De Cock, M., Teredesai, A.M.: Trust- and distrust-based recommendations for controversial reviews. *IEEE Intelligent Systems*, in press.
65. Zadeh, L.A.: Fuzzy sets. *Information and Control* **8**, 338–353 (1965)
66. Zaihrayeu, I., Pinheiro da Silva, P., McGuinness, D.: IWTrust: Improving user trust in answers from the web. In: *Proc. of the Third International Conference On Trust Management*, pp. 384–392 (2005)
67. Zhang, S., Ouyang, Y., Ford, J., Makedon, F.: Analysis of a low-dimensional linear model under recommendation attacks. In: *Proc. of the International ACM SIGIR Conference*, pp. 517–524 (2006)
68. Ziegler, C., Lausen, G.: Propagation models for trust and distrust in social networks. *Information System Frontiers* **7**, 337–358 (2005)
69. Ziegler, C., Golbeck, J.: Investigating correlations of trust and interest similarity - Do birds of a feather really flock together?. *Decision Support Systems* **43**, 460–475 (2007)

Chapter 21

Group Recommender Systems: Combining Individual Models

Judith Masthoff

Abstract This chapter shows how a system can recommend to a group of users by aggregating information from individual user models and modelling the users affective state. It summarizes results from previous research in this area. It also shows how group recommendation techniques can be applied when recommending to individuals, in particular for solving the cold-start problem and dealing with multiple criteria.

21.1 Introduction

Most work on recommender systems to date focuses on recommending items to individual users. For instance, they may select a book for a particular user to read based on a model of that user's preferences in the past. The challenge recommender system designers traditionally faced is how to decide what would be optimal for an individual user. A lot of progress has been made on this, as evidenced by other chapters in this handbook (e.g. Chapters 2,3, 4,5 and 6).

In this chapter, we go one-step further. There are many situations when it would be good if we could recommend to a group of users rather than to an individual. For instance, a recommender system may select television programmes for a group to view or a sequence of songs to listen to, based on models of all group members. Recommending to groups is even more complicated than recommending to individuals. Assuming that we know perfectly what is good for individual users, the issue arises how to combine individual user models. In this chapter, we will discuss how group recommendation works, what its problems are, and what advances have been made. Interestingly, we will show that group recommendation techniques have many uses as well when recommending to individuals. So, even if you are developing recom-

Judith Masthoff
University of Aberdeen, AB24 3UE Aberdeen UK, e-mail: j.masthoff@abdn.ac.uk

mender systems aimed at individual users you may still want to read on (perhaps reading Section 21.7 first will convince you).

This chapter focusses on deciding what to recommend to a group, in particular how to aggregate individual user models. There are other issues to consider when building a group recommender system which are outside the scope of this chapter. In particular:

- *How to acquire information about individual users' preferences.* The usual recommender techniques can be used (such as explicit ratings and collaborative- and content-based filtering, see other handbook chapters). There is a complication in that it is difficult to infer an individual's preferences when a group uses the system, but inferences can be made during individual use combined with a probabilistic model when using it in company. An additional complication is that an individual's ratings may depend on the group they are in. For instance, a teenager may be very happy to watch a programme with his younger siblings, but may not want to see it when with his friends.
- *How will the system know who is present?* Different solutions exist, such as users explicitly logging in, probabilistic mechanisms using the time of day to predict who is present, the use of tokens and tags, etc [10].
- *How to present and explain group recommendations?* As seen in this handbook's chapter on explanations, there are already many considerations when presenting and explaining *individual* recommendations. The case of group recommendations is even more difficult. More discussion on explaining group recommendations is provided in [8] and under Challenges in our final section.
- *How to help users to settle on a final decision?* In some group recommenders, users are given group recommendations, and based on these recommendations negotiate what to do. In other group recommenders this is not an issue (see Section 21.2.3 on the difference between passive and active groups). An overview of how users' decisions can be aided is provided in [8].

The next section highlights usage scenarios of group recommenders, and provides a classification of group recommenders inspired by differences between the scenarios. Section 21.3 discusses strategies for aggregating models of individual users to allow for group recommendation, what strategies have been used in existing systems, and what we have learned from our experiments in this area. Section 21.4 deals with the issue of order when we want to recommend a sequence of items. Section 21.5 provides an introduction into the modelling of affective state, including how an individual's affective state can be influenced by the affective states of other group members. Section 21.6 explores how such a model of affective state can be used to build more sophisticated aggregation strategies. Section 21.7 shows how group modelling and group recommendation techniques can be used when recommending to an individual user. Section 21.8 concludes this chapter and discusses future challenges.

21.2 Usage Scenarios and Classification of Group Recommenders

There are many circumstances in which adaptation to a group is needed rather than to an individual. Below, we present two scenarios that inspired our own work in this area, discuss the scenarios underlying related work, and provide a classification of group recommenders inspired by differences between the scenarios.

21.2.1 Interactive Television

Interactive television offers the possibility of personalized viewing experiences. For instance, instead of everybody watching the same news program, it could be personalized to the viewer. For me, this could mean adding more stories about the Netherlands (where I come from), China (a country that fascinates me after having spent some holidays there) and football, but removing stories about cricket (a sport I hardly understand) and local crime. Similarly, music programs could be adapted to show music clips that I actually like.

There are two main differences between traditional recommendation as it applies to say PC-based software and the interactive TV scenarios sketched above. Firstly, in contrast to the use of PCs, television viewing is largely a family or social activity. So, instead of adapting the news to an individual viewer, the television would have to adapt it to the group of people sitting in front of it at that time. Secondly, traditional work on recommendation has often concerned recommending one particular thing to the user, so for instance, which movie the user should watch. In the scenarios sketched above, the television needs to adapt a sequence of items (news items, music clips) to the viewer. The combination of recommending to a group and recommending a sequence is very interesting, as it may allow you to keep all individuals in the group satisfied by compensating for items a particular user dislikes with other items in the sequence which they do like.

21.2.2 Ambient Intelligence

Ambient intelligence deals with designing physical environments that are sensitive and responsive to the presence of people. For instance, consider the case of a bookstore where sensors detect the presence of customers identified by some portable device (e.g. a Bluetooth-enabled mobile phone, or a fidelity card equipped with an active RFID tag). In this scenario, there are various sensors distributed among the shelves and sections of the bookstore which are able to detect the presence of individual customers. The bookstore can associate the identification of customers with their profiling information, such as preferences, buying patterns and so on.

With this infrastructure in place, the bookstore can provide customers with a responsive environment that would adapt to maximise their well-being with a view to increasing sales. For instance, the device playing the background music should take into account the preferences of the group of customers within hearing distance. Similarly, LCD displays scattered in the store show recommended books based on the customers nearby, the lights on the shop's display window (showing new titles) can be rearranged to reflect the preferences and interests of the group of customers watching it, and so on. Clearly, group adaptation is needed, as most physical environments will be used by multiple people at the same time.

21.2.3 Scenarios Underlying Related Work

In this section we discuss the scenarios underlying the best known group recommender systems:

- MUSICFX [15] chooses a radio station for background music in a fitness centre, to suit a group of people working out at a given time. This is similar to the Ambient Intelligence scenario discussed above.
- POLYLENS [17] is a group recommender extension of MOVIELENS. MOVIELENS recommends movies based on an individual's taste as inferred from ratings and social filtering. POLYLENS allows users to create groups and ask for group recommendations.
- INTRIGUE [2] recommends places to visit for tourist groups taking into account characteristics of subgroups within that group (such as children and the disabled).
- The TRAVEL DECISION FORUM [7] helps a group to agree on the desired attributes of a planned joint holiday. Users indicate their preferences on a set of features (like sport and room facilities). For each feature, the system aggregates the individual preferences, and users interact with embodied conversational agents representing other group members to reach an accepted group preference.
- The COLLABORATIVE ADVISORY TRAVEL SYSTEM (CATS) [16] also helps users to choose a joint holiday. Users consider holiday packages, and critique their features (e.g., 'like the one shown but with a swimming pool'). Based on these critiques, the system recommends other holidays to them. Users also select holidays they like for other group members to see, and these are annotated with how well they match the preferences of each group member (as induced from their critiques). The individual members' critiques results in a group preference model, and other holidays are recommended based on this model.
- YU'S TV RECOMMENDER [20] recommends a television program for a group to watch. It bases its recommendation on the individuals' preferences for program features (such as genre, actors, keywords).

21.2.4 A Classification of Group Recommenders

The scenarios provided above differ on several dimensions, which provide a way to classify group recommender systems:

- *Individual preferences are known versus developed over time.* In most scenarios, the group recommender starts with individual preferences. In contrast, in CATS, individual preferences develop over time, using a critiquing style approach. Chapter 13 discusses critiquing and its role in group recommendation.
- *Recommended items are experienced by the group versus presented as options.* In the Interactive TV scenario, the group experiences the news items. In the Ambient Intelligence and MUSICFX scenarios, they experience the music. In contrast, in the other scenarios, they are presented with a list of recommendations. For example, POLYLENS presents a list of movies the group may want to watch.
- *The group is passive versus active.* In most scenarios, the group does not interact with the way individual preferences are aggregated. However, in the TRAVEL DECISION FORUM and CATS the group negotiates the group model.
- *Recommending a single item versus a sequence.* In the scenarios of MUSICFX, POLYLENS, and YU'S TV RECOMMENDER it is sufficient to recommend individual items: people normally only see one movie per evening, radio stations can play forever, and YU'S TV RECOMMENDER chooses one TV program only. Similarly, in the TRAVEL DECISION FORUM and CATS users only go on one holiday. In contrast, in our Interactive TV scenario, a sequence of items is recommended, for example making up a complete news broadcast. Similarly, in INTRIGUE, it is quite likely that a tourist group would visit multiple attractions during their trip, so would be interested in a sequence of attractions to visit. Also, in the Ambient Environment scenario it is likely that a user will hear multiple songs, or see multiple items on in-store displays.

In this chapter, we will focus on the case where individual preferences are known, the group directly experiences the items, the group is passive, and a sequence is recommended. Recommending a sequence raises interesting questions regarding sequence order (see Section 21.4) and considering the individuals' affective state (see Sections 21.5 and 21.6). A passive group with direct experience of the items makes it even more important that the group recommendation is good.

DeCampos et al.'s classification of group recommenders also distinguishes between passive and active groups [4]. In addition, it uses two other dimensions:

- *How individual preferences are obtained.* They distinguish between content-based and collaborative filtering. Of the systems mentioned above, POLYLENS is the only one that uses collaborative filtering.
- *Whether recommendations or profiles are aggregated.* In the first case, recommendations are produced for individuals and then aggregated into a group recommendation. In the second case, individual preferences are aggregated into a group model, and this model is used to produce a group recommendation. They

mention INTRIGUE and POLYLENS as aggregating recommendations, while the others aggregate profiles.

These two dimensions are related to how the group recommender is implemented rather than being inherent to the usage scenario. In this chapter, we focus on aggregating profiles, but the same aggregation strategies apply when aggregating recommendations. The material presented in this chapter is independent of how the individual preferences are obtained.

21.3 Aggregation Strategies

The main problem group recommendation needs to solve is how to adapt to the group as a whole based on information about individual users' likes and dislikes. For instance, suppose the group contains three people: Peter, Jane and Mary. Suppose a system is aware that these three individuals are present and knows their interest in each of a set of items (e.g. music clips or advertisements). Table 21.1 gives example ratings on a scale of 1 (really hate) to 10 (really like). Which items should the system recommend, given time for four items?

Table 21.1: Example of individual ratings for ten items (A to J)

	A	B	C	D	E	F	G	H	I	J
Peter	10	4	3	6	10	9	6	8	10	8
Jane	1	9	8	9	7	9	6	9	3	8
Mary	10	5	2	7	9	8	5	6	7	6

21.3.1 Overview of Aggregation Strategies

Many strategies exist for aggregating individual ratings into a group rating (e.g. used in elections and when selecting a party leader). For example, the Least Misery Strategy uses the minimum of ratings to avoid misery for group members (Table 21.2).

Eleven aggregation strategies inspired by Social Choice Theory are summarised in Table 21.3 (see [10] for more details).

Table 21.2: Example of the Least Misery Strategy

	A	B	C	D	E	F	G	H	I	J
Peter	10	4	3	6	10	9	6	8	10	8
Jane	1	9	8	9	7	9	6	9	3	8
Mary	10	5	2	7	9	8	5	6	7	6
Group Rating	1	4	2	6	7	8	5	6	3	6

21.3.2 Aggregation Strategies Used in Related Work

Most of the related work uses one the aggregation strategies in Table 21.3 (sometimes with a small variation), and they differ in the one used:

- **INTRIGUE** uses a weighted form of the Average strategy. It bases its group recommendations on the preferences of subgroups, such as children and the disabled. It takes the average, with weights depending on the number of people in the subgroup and the subgroup’s relevance (children and disabled were given a higher relevance).
- **POLYLENS** uses the Least Misery Strategy, assuming groups of people going to watch a movie together tend to be small and that a small group tends to be as happy as its least happy member.
- **MUSICFX** uses a variant of the Average Without Misery Strategy. Users rate all radio stations, from +2 (really love this music) to -2 (really hate this music). These ratings are converted to positive numbers (by adding 2) and then squared to widen the gap between popular and less popular stations. An Average Without Misery strategy is used to generate a group list: the average of ratings is taken but only for those items with individual ratings all above a threshold. To avoid starvation and always picking the same station, a weighted random selection is made from the top stations of the list.
- **YU’S TV RECOMMENDER** uses a variant of the Average Strategy. It bases its group recommendation on individuals’ ratings of program features: -1 (dislikes the feature), +1 (likes the feature) and 0 (neutral). The feature vector for the group minimizes its distance compared to individual members’ feature vectors. This is similar to taking the average rating per feature.
- The **TRAVEL DECISION FORUM** has implemented multiple strategies, including the Average Strategy and the Median Strategy. The Median strategy (not in Table 21.3) uses the middle value of the ratings. So, in our example, this results in group ratings of 10 for A, and 9 for F. The Median Strategy was chosen because it is nonmanipulable: users cannot steer the outcome to their advantage by deliberately giving extreme ratings that do not truly reflect their opinions. In contrast, for example, with the Least Misery strategy devious users can avoid getting items they dislike slightly, by giving extremely negative ratings. The issue of manipulability is most relevant when users provide explicit ratings, used

Table 21.3: Overview of Aggregation Strategies

Strategy	How it works	Example
Plurality Voting	Uses ‘first past the post’: repetitively, the item with the most votes is chosen.	A is chosen first, as it has the highest rating for the majority of the group, followed by E (which has the highest rating for the majority when excluding A).
Average	Averages individual ratings	B’s group rating is 6, namely $(4+9+5)/3$.
Multiplicative	Multiplies individual ratings	B’s group rating is 180, namely $4*9*5$.
Borda Count	Counts points from items’ rankings in the individuals’ preference lists, with bottom item getting 0 points, next one up getting one point, etc	A’s group rating is 17, namely 0 (last for Jane) + 9 (first for Mary) + 8 (shared top 3 for Peter)
Copeland Rule	Counts how often an item beats other items (using majority vote) minus how often it loses	F’s group rating is 5, as F beats 7 items (B,C,D,G,H,I,J) and loses from 2 (A,E).
Approval Voting	Counts the individuals with ratings for the item above a approval threshold (e.g. 6)	B’s group rating is 1 and F’s is 3.
Least Misery	Takes the minimum of individual ratings	B’s group rating is 4, namely the smallest of 4,9,5.
Most Pleasure	Takes the maximum of individual ratings	B’s group rating is 9, namely the largest of 4,9,5.
Average without Misery	Averages individual ratings, after excluding items with individual ratings below a certain threshold (say 4).	J’s group rating is 7.3 (the average of 8,8,6), while A is excluded because Jane hates it.
Fairness	Items are ranked as if individuals are choosing them in turn.	Item E may be chosen first (highest for Peter), followed by F (highest for Jane) and A (highest for Mary).
Most respected person	Uses the rating of the most respected individual.	If Jane is the most respected person, then A’s group rating is 1. If Mary is most respected, then it is 10.

for group recommendation only, and are aware of others’ ratings, all of which is the case in the TRAVEL DECISION FORUM. It is less relevant when ratings are inferred from user behaviour, also used for individual recommendations, and

users are unaware of the ratings of others (or even of the aggregation strategy used).

- In CATS, users indicate through critiquing which features a holiday needs to have. For certain features, users indicate whether they are required (e.g. ice skating required). For others, they indicate quantities (e.g. at least 3 ski lifts required). The group model contains the requirements of all users, and the item which fulfils most requirements is recommended. Users can also completely discard holidays, so, the strategy has a Without Misery aspect.

It should be noted that both YU'S TV RECOMMENDER and the TRAVEL DECISION FORUM aggregate preferences for each feature without using the idea of fairness: loosing out on one feature is not compensated by getting your way on another.

Though some exploratory evaluation of MUSICFX, POLYLENS and CATS has taken place, for none of these systems it has been investigated how effective their strategy really is, and what the effect would be of using a different strategy. The experiments presented in the next section shed some light on this question.

In contrast, some evaluation of YU'S TV RECOMMENDER has taken place [20]. They found that their aggregation worked well when the group was quite homogeneous, but that results were disliked when the group was quite heterogeneous. This is as we would expect, given the Average Strategy will make individuals quite happy if they are quite similar, but will cause misery when tastes differ widely.

21.3.3 Which Strategy Performs Best

We conducted a series of experiments to investigate which strategy from Table 21.3 is best (see [10] for details).

In Experiment 1 (see Figure 21.1), we investigated how people would solve this problem, using the User as Wizard evaluation method [13]. Participants were given individual ratings identical to those in Table 21.1. These ratings were chosen to be able to distinguish between strategies. Participants were asked which items the group should watch, if there was time for one, two, ..., seven items. We compared participants' decisions and rationale with those of the aggregation strategies. We found that participants cared about fairness, and about preventing misery and starvation ("this one is for Mary, as she has had nothing she liked so far"). Participants' behaviour reflected that of several of the strategies (e.g. the Average, Least Misery, and Average Without Misery were used), while other strategies (e.g. Borda count, Copeland rule) were clearly not used.

In Experiment 2 (see Figure 21.2), participants were given item sequences chosen by the aggregation strategies as well as the individual ratings in Table 21.1. They rated how satisfied they thought the group members would be with those sequences, and explained their ratings. We found that the Multiplicative Strategy (which multiplies the individual ratings) performed best, in the sense that it was the only strategy for which *all* participants thought its sequence would keep all members of the group

satisfied. Borda count, Average, Average without Misery and Most Pleasure also performed quite well. Several strategies (such as Copeland rule, Plurality voting, Least misery) could be discarded as they clearly were judged to result in misery for group members.

We also compared the participants' judgements with predictions by simple satisfaction modelling functions. Amongst other, we found that more accurate predictions resulted from using:

- quadratic ratings, which e.g. makes the difference between a rating of 9 and 10 bigger than that between a rating of 5 and 6
- normalization, which takes into account that people rate in different ways, e.g., some always use the extremes of a scale, while others only use the middle of the scale.

21.4 Impact of Sequence Order

As mentioned in Section 21.2, we are particularly interested in recommending a *sequence* of items. For example, for a personalised news program on TV, a recommender may select seven news items to be shown to the group. To select the items, it can use an aggregation strategy (such as the Multiplicative Strategy) to combine individual preferences, and then select the seven items with the highest group ratings. Once the items have been selected, the question arises in what order to show them in the news program. For example, it could show the items in descending order of group rating, starting with the highest rated item and ending with the lowest rated one. Or, it could mix up the items, showing them in a random order.

However, the problem is actually far more complicated than that. Firstly, in responsive environments, the group membership changes continuously, so deciding on the next seven items to show based on the current members seems not a sensible

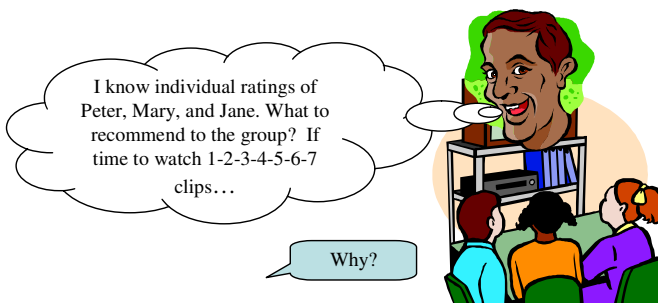


Fig. 21.1: Experiment 1: which sequence of items do people select if given the system's task

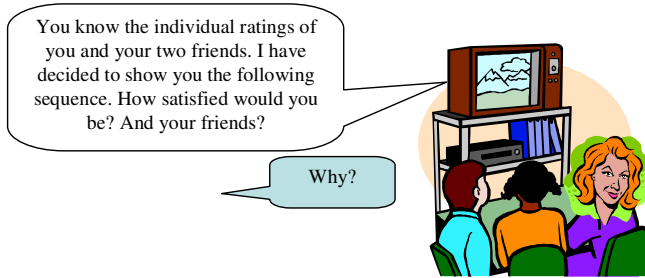


Fig. 21.2: Experiment 2: What do people like?

strategy, as in the worse case, none of these members may be present anymore when the seventh item is shown.

Secondly, overall satisfaction with a sequence may depend more on the order of the items than one would expect. For example, for optimal satisfaction, we may need to ensure that our news program has:

- *A good narrative flow.* It may be best to show topically related items together. For example, if we have two news items about Michael Jackson (say about his funeral and about a tribute tour) then it seems best if these items are presented together. Similarly, it would make sense to present all sports' items together.
- *Mood consistency.* It may be best to show items with similar moods together. For example, viewers may not like seeing a sad item (such as a soldier's death) in the middle of two happy items (such as a decrease in unemployment and a sporting victory).
- *A strong ending.* It may be best to end with a well-liked item, as viewers may remember the end of the sequence most.

Similar ordering issues arise in other recommendation domains. For example, a music programme may want to consider rhythm when sequencing items. The recommender may need additional information (such as items' mood, topics, rhythm) to optimise ordering. It is beyond the topic of this chapter to discuss how this can be done (and is very recommender domain specific). We just want to highlight that the items already shown may well influence what the best next item is. For example, suppose the top four songs in a music recommender were all Blues. It may well be that another Blues song ranked sixth may be a better next selection than a Classical Opera song ranked fifth.

In Experiment 3 (see Figure 21.3), we investigated how a previous item may influence the impact of the next item. Amongst others, we found that mood (resulting from the previous item) and topical relatedness can influence ratings for subsequent items. This means that aggregating individual profiles into a group profile should be done repeatedly, every time a decision needs to be made about the next item to display.

[Insert name of your favorite sport's club] wins important game
 Fleet of limos for Jennifer Lopez 100-metre trip
 Heart disease could be halved
 Is there room for God in Europe?
 Earthquake hits Bulgaria
 UK fire strike continues
 Main three Bulgarian players injured after Bulgaria-Spain football match

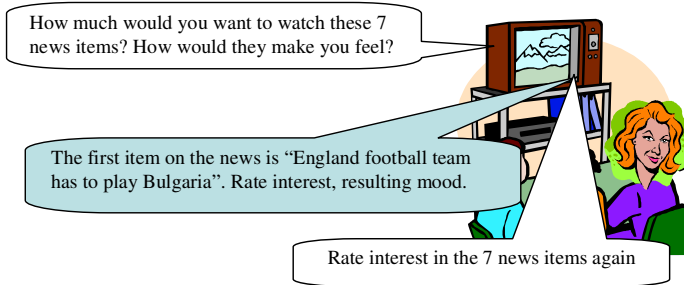


Fig. 21.3: Experiment 3: Investigating the effect of mood and topic

21.5 Modelling Affective State

When recommending to a group of people, you cannot give everybody what they like all of the time. However, you do not want anybody to get too dissatisfied. For instance, in a shop it would be bad if a customer were to leave and never come back, because they really cannot stand the background music. Many shops currently opt to play music that nobody really hates, but most people not love either. This may prevent losing customers, but would not result in increasing sales. An ideal shop would adapt the music to the customers in hearing range in such a way that they get songs they really like most of the time (increasing the likelihood of sales and returns to the shop). To achieve this, it is unavoidable that customers will occasionally get songs they hate, but this should happen at a moment when they can cope with it (e.g. when being in a good mood because they loved the previous songs). Therefore, it is important to monitor continuously how satisfied each group member is. Of course, it would put an unacceptable burden on the customers if they had to rate their satisfaction (on music, advertisements etc) all the time. Similarly, measuring this satisfaction via sensors (such as heart rate monitors or facial expression recognizers) is not yet an option, as they tend to be too intrusive, inaccurate or expensive. So, we propose to model group members' satisfaction; predicting it based on what we know about their likes and dislikes.

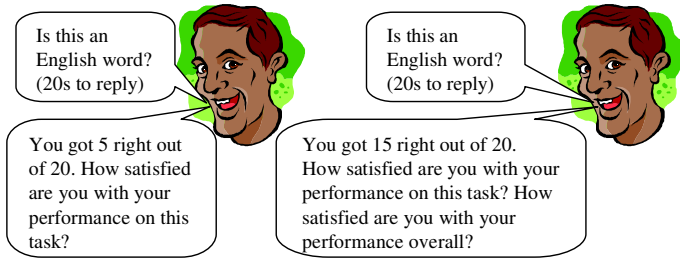


Fig. 21.4: Experiment 4: Measuring overall satisfaction during a series of tasks

21.5.1 Modelling an Individual’s Satisfaction on its Own

In [12], we investigated four satisfaction functions to model an individual’s satisfaction. We compared the predictions of these satisfaction functions with the predictions of real users. We also performed an experiment (see Figure 21.4) to compare the predictions with the real feelings of users.

The satisfaction function that performed best defines the satisfaction of a user with a new item i after having seen a sequence $items$ of items as:

$$Sat(items + \langle i \rangle) = \frac{\delta \times Sat(items) + Impact(i, \delta \times Sat(items))}{1 + \delta} \tag{21.1}$$

with the impact on satisfaction of new item i given existing satisfaction s defined as

$$Impact(i, s) = Impact(i) + (s - Impact(i)) \times \epsilon, \text{ for } 0 \leq \epsilon \leq 1 \text{ and } 0 \leq \delta \leq 1 \tag{21.2}$$

Parameter δ represents satisfaction decaying over time (with $\delta=0$ past items have no influence, with $\delta=1$ there is no decay).

Parameter ϵ represents the influence of the user’s satisfaction after experiencing previous items on the impact of a new item. This parameter is inspired by the psychology and economics literature, which shows that mood impacts evaluative judgement [12]. For instance, half the participants answering a questionnaire about their TVs received a small present first to put them in a good mood. These participants were found to have televisions that performed better. So, if a user is in a good mood due to liking previous items, the impact of an item they normally dislike may be smaller (with how much smaller depending on ϵ).

Parameters δ and ϵ are user dependent (as confirmed in the experiment in [12]). We will not define $Impact(i)$ in this chapter, see [12] for details, but it involves quadratic ratings and normalization as found in the experiment discussed above.

21.5.2 Effects of the Group on an Individual's Satisfaction

The satisfaction function given does not take the satisfaction of other users in the group into account, which may well influence a user's satisfaction. As argued in [12] based on social psychology, two main processes can take place.

Emotional Contagion. Firstly, the satisfaction of other users can lead to so-called emotional contagion: other users being satisfied may increase a user's satisfaction (e.g. if somebody smiles at you, you may automatically smile back and feel better as a result). The opposite may also happen: other users being dissatisfied may decrease a user's satisfaction. For instance, if you are watching a film with a group of friends than the fact that your friends are clearly not enjoying it may negatively impact your own satisfaction.

Emotional contagion may depend on your personality (some people are more easily contagated than others), and your relationship with the other person. Anthropologists and social psychologists have found substantial evidence for the existence of four basic types of relationships, see Figure 21.5. In Experiment 5 (see Figure 21.6), we confirmed that emotional contagion indeed depends on the relationship you have: you are more likely to be contagated by somebody you love (like your best friend) or respect (like your mother or boss) then by somebody you are on equal footing with or are in competition with.

Conformity. Secondly, the opinion of other users may influence your own expressed opinion, based on the so-called process of conformity.

Figure 21.7 shows the famous conformity experiment by Asch [3]. Participants were given a very easy task to do, like decide which of the four lines has the same orientation as the line in Card A. They thought they were surrounded by other participants, but in fact the others where part of the experiment team. The others all

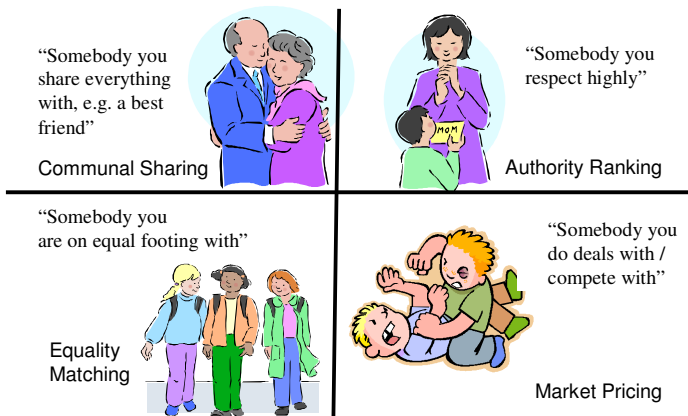


Fig. 21.5: Types of relationship

answered the question before them, picking the same wrong answer. It was shown that most participants then pick that same wrong answer as well.

Two types of conformity exist: (1) normative influence, in which you want to be part of the group and express an opinion like the rest of the group even though inside you still believe differently, and (2) informational influence, in which your own opinion changes because you believe the group must be right. Informational influence would change your own satisfaction, while normative influence can change the satisfaction of others through emotional contagion because of the (insincere) emotions you are portraying.

More complicated satisfaction functions are presented in [12] to model emotional contagion and both types of conformity.

21.6 Using Affective State inside Aggregation Strategies

Once you have an accurate model of the individual users' satisfaction, it would be nice to use this model to improve on the group aggregation strategies. For instance, the aggregation strategy could set out to please the least satisfied member of the group. This can be done in many different ways, and we have only started to explore this issue. For example:

- *Strongly Support Grumpiest strategy.* This strategy picks the item which is *most liked* by the least satisfied member. If multiple of these items exist, it uses one of the standard aggregation strategies, for instance the Multiplicative Strategy, to distinguish between them.
- *Weakly Support Grumpiest strategy.* This strategy selects the items that are *quite liked* by the least satisfied member, for instance items with a rating of 8 or above. It uses one of the standard aggregation strategies, like the Multiplicative Strategy, to choose between these items.
- *Weighted strategy.* This strategy assign weights to users depending on their satisfaction, and then use a weighted form of a standard aggregation strategy. For instance, Table 21.4 shows the effect of assigning double the weight to Jane



Fig. 21.6: Experiment 5: Impact of relationship type on emotional contagion

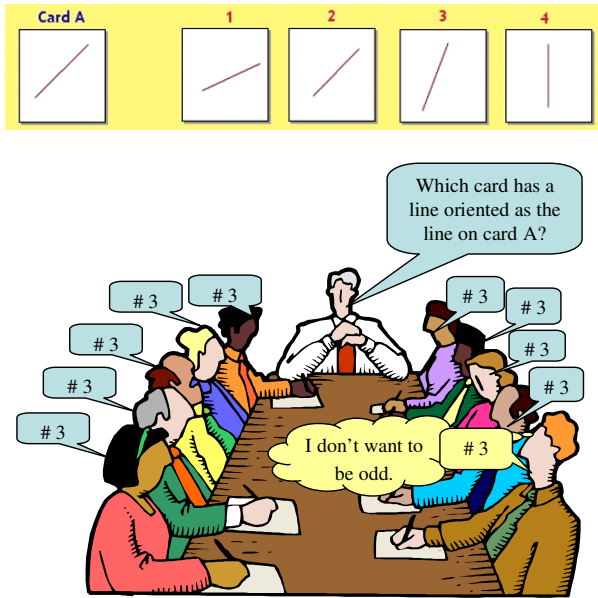


Fig. 21.7: Conformity experiment by Asch

when using the Average Strategy. Note that weights are impossible to apply to a strategy like the Least Misery Strategy.

Table 21.4: Results of Average strategy with equal weights and with twice the weight for Jane

	A	B	C	D	E	F	G	H	I	J
Peter	10	4	3	6	10	9	6	8	10	8
Jane	1	9	8	9	7	9	6	9	3	8
Mary	10	5	2	7	9	8	5	6	7	6
Average (equal weights)	7	6	4.3	7.3	8.7	8.7	5.7	7.7	6.7	7.3
Average (Jane twice)	5.5	6.8	5.3	8.3	8.3	8.8	5.8	8	5.8	7.5

In [14], we discuss this in more detail, propose an agent-based architecture for applying these ideas to the ambient intelligent scenario, and describe an implemented prototype. Clearly, empirical research is needed to investigate the best way of using affective state inside an aggregation strategy.

21.7 Applying Group Recommendation to Individual Users

So, what if you are developing an application that recommends to a single user? Group recommendation techniques can be useful in three ways: (1) to aggregate multiple criteria, (2) to solve the so-called cold-start problem, (3) to take into account opinions of others. Chapter 22 also discusses how aggregation may be needed when recommending to individuals, and covers several specific aggregation functions.

21.7.1 Multiple Criteria

Sometimes it is difficult to give recommendations because the problem is multi-dimensional: multiple criteria play a role. For instance, in a news recommender system, a user may have a preference for location (being more interested in stories close to home, or related to their favourite holiday place). The user may also prefer more recent news, and have topical preferences (e.g. preferring news about politics to news about sport). The recommender system may end up with a situation like in Table 21.5, where different news story rate differently on the criteria. Which news stories should it now recommend?

Table 21.5: Ratings on criteria for 10 news items

	A	B	C	D	E	F	G	H	I	J
Topic	10	4	3	6	10	9	6	8	10	8
Location	1	9	8	9	7	9	6	9	3	8
Recency	10	5	2	7	9	8	5	6	7	6

The issue of multiple criteria is discussed in details in Chapter 24. Here we show how to address this issue in group recommender systems.

Table 21.5 resembles the one we had for group recommendation above (Table 21.1), except that now instead of multiple users we have multiple criteria to satisfy. It is possible to apply our group recommendation techniques to this problem. However, there is an important difference between adapting to a group of people and adapting to a group of criteria. When adapting to a group of people, it seems sensible and morally correct to treat everybody equally. Of course, there may be some exceptions, for instance when the group contains adults as well as children, or when it is somebody's birthday. But in general, equality seems a good choice, and this was used in the group adaptation strategies discussed above. In contrast, when adapting to a group of criteria, there is no particular reason for assuming all criteria are as important. It is even quite likely that not all criteria are equally important to a par-

ticular person. Indeed, in an experiment we found that users treat criteria in different ways, giving more importance to some criteria (e.g. recency is seen as more important than location) [11]. So, how can we adapt the group recommendation strategies to deal with this? There are several ways in which this can be done:

- Apply the strategy to the most respected criteria only. The ratings of unimportant criteria are ignored completely. For instance, assume criterion Location is regarded unimportant, then its ratings are ignored. Table 21.6 shows the result of the Average Strategy when ignoring Location.
- Apply the strategy to all criteria but use weights. The ratings of unimportant criteria are given less weight. For instance, in the Average Strategy, the weight of a criterion is multiplied with its ratings to produce new ratings. For instance, suppose criteria Topic and Recency were three times as important as criterion Location. Table 21.7 shows the result of the Average Strategy using these weights. In case of the Multiplicative Strategy, multiplying the ratings with weights does not have any effect. In that strategy, it is better to use the weights as exponents, so replace the ratings by the ratings to the power of the weight. Note that in both strategies, a weight of 0 results in ignoring the ratings completely, as above.
- Adapt a strategy to behave differently to important versus unimportant criteria: Unequal Average Without Misery. Misery is avoided for important criteria but not for unimportant ones. Assume criterion Location is again regarded as unimportant. Table 21.8 shows the results of the Unequal Average Without Misery strategy with threshold 6.

Table 21.6: Average Strategy ignoring unimportant criterion Location

	A	B	C	D	E	F	G	H	I	J
Topic	10	4	3	6	10	9	6	8	10	8
Recency	10	5	2	7	9	8	5	6	7	6
Group	20	9	5	13	19	17	11	14	17	14

Table 21.7: Average Strategy with weights 3 for Topic and Recency and 1 for Location

	A	B	C	D	E	F	G	H	I	J
Topic	30	12	9	18	30	27	18	24	30	24
Location	1	9	8	9	7	9	6	9	3	8
Recency	30	15	6	21	27	24	15	18	21	18
Group	61	36	23	48	64	60	39	51	54	50

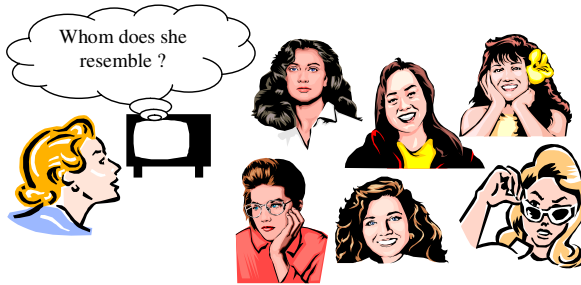


Fig. 21.8: Cold-start problem in case of social-filtering

Table 21.8: Unequal Average Without Misery Strategy with Location unimportant and threshold 6

	A	B	C	D	E	F	G	H	I	J
Topic	10	4	3	6	10	9	6	8	10	8
Location	1	9	8	9	7	9	6	9	3	8
Recency	10	5	2	7	9	8	5	6	7	6
Group	21			22	26	26		23	20	22

We have some evidence that people’s behaviour reflects the outcomes of these strategies [11], however, more research is clearly needed in this area to see which strategy is best. Also, more research is needed to establish when to regard a criterion as ”unimportant”.

The issue of multiple criteria is also the topic of another chapter in this handbook (see Chapter 24).

21.7.2 Cold-Start Problem

A big problem for recommender systems is the so-called cold-start problem: to adapt to a user, the system needs to know what the user liked in the past. This is needed in content-based filtering to decide on items similar to the ones the user liked. It is needed in social filtering to decide on the users who resemble this user in the sense that they (dis)liked the same items in the past (see Figure 21.8). So, what if you do not know anything about the user yet, because they only just started using the system? Recommender system designers tend to solve this problem by either getting users to rate items at the start, or by getting them to answer some demographic questions (and then using stereotypes as a starting point, e.g. elderly people like classical music).

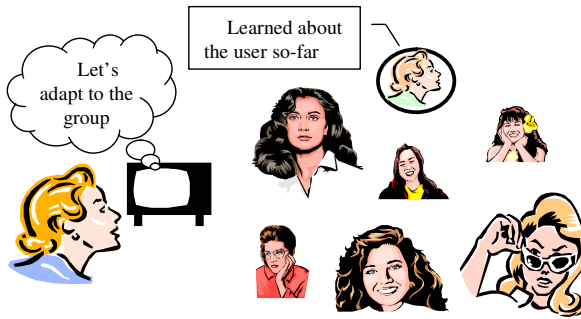


Fig. 21.9: Gradually learning about the user, and whom she resembles most

Both methods require user effort. It is also not easy to decide which items to get a user to rate, and stereotypes can be quite wrong and offensive (some elderly people prefer pop music and people might not like being classified as elderly).

The group recommendation work presented in this chapter provides an alternative solution. When a user is new to the system, we simply provide recommendations to that new user that would keep the whole group of existing users happy. We assume that our user will resemble one of our existing users, though we do not know which one, and that by recommending something that would keep all of them happy, the new user will be happy as well.

Gradually, we will learn about the new user's tastes, for instance, by them rating our recommended items or, more implicitly, by them spending time on the items or not. We provide recommendations to the new user that would keep the group of existing users happy including the new user (or more precisely, the person we now assume the new user to be). The weight attached to the new user will be low initially, as we do not know much about them yet, and will gradually increase. We also start to attach less weight to existing users whose taste now evidently differs from our new user.

Figure 21.9 shows an example of the adaptation: the system is including the observed tastes of the new user to some extent, and has started to reduce the weights of some of the other users. After prolonged use of the system, the user's inferred wishes will completely dominate the selection.

We have done a small-scale study using the MovieLens dataset to explore the effectiveness of this approach. We randomly selected five movies, and twelve users who had rated them: ten users as already known to the recommender, and two as new users. Using the Multiplicative Strategy on the group of known users, movies were ranked for the new users. Results were encouraging: the movie ranked highest was in fact the most preferred movie for the new users, and also the rest of the ranking was fine given the new users' profiles. Applying weights led to a further improvement of the ranking, and weights started to reflect the similarity of the new users with known users. More detail on the study and on applying group adaptation to solve the cold-start problem is given in [9].

21.7.3 Virtual Group Members

Finally, group adaptation can also be used when adapting to an individual by adding virtual members to the group. For instance, a parent may be fine with the television entertaining their child, but may also want the child occasionally to learn something. When the child is alone, the profile of the parent can be added to the group as a virtual group member, and the TV could try to satisfy both.

21.8 Conclusions and Challenges

Group recommendation is a relatively new research area. This chapter is intended as an introduction in the area, in particular on aggregating individual user profiles. For more detail please see [10, 12, 14, 11, 9, 7, 8].

21.8.1 Main Issues Raised

The main issues raised in this chapter are:

- Adapting to groups is needed in many scenarios such as interactive TV, ambient intelligence, recommending to tourist groups, etc. Inspired by the differences between scenarios, group recommenders can be classified using multiple dimensions.
- Many strategies exist for aggregating individual preferences (see Table 21.3), and some perform better than others. Users seem to care about avoiding misery and fairness.
- Existing group recommenders differ on the classification dimensions and in the aggregation strategies used. See Table 21.9 for an overview.
- When recommending a sequence of items, aggregation of individual profiles has to occur at each step in the sequence, as earlier items may impact the ratings of later items.
- It is possible to construct satisfaction functions to predict how satisfied an individual will be at any time during a sequence. However, group interaction effects (such as emotional contagion and conformity) can make this complicated.
- It is possible to evaluate in experiments how good aggregation strategies and satisfaction functions are, though this is not an easy problem.
- Group aggregation strategies are not only important when recommending to groups of people, but can also be applied when recommending to individuals, e.g. to prevent the cold-start problem and deal with multiple criteria.

21.8.2 *Caveat: Group Modelling*

The term "group modelling" is also used for work that is quite different from that presented in this chapter. A lot of work has been on modelling common knowledge between group members (e.g. [6, 19]), modelling how a group interacts (e.g. [18, 5]) and group formation based on individual models (e.g. [18, 1]).

21.8.3 *Challenges*

Compared to work on individual recommendations, group recommendation is still quite a novel area. The work presented in this chapter is only a starting point. There are many challenging directions for further research, including:

- *Recommending item sequences to a group.* Our own work seems to be the only work to date on recommending balanced *sequences* that address the issue of fairness. Even though sequences are important for the usage scenario of INTRIGUE, their work has not investigated making sequences balanced nor has it looked at sequence order. Clearly, a lot more research is needed on recommending and ordering sequences, in particular on how already shown items should influence the ratings of other items. Some of this research will have to be recommender domain specific.
- *Modelling of affective state.* There is a lot more work needed to produce validated satisfaction functions. The work presented in this chapter and [12] is only the starting point. In particular, large scale evaluations are required, as are investigations on the affect of group size.
- *Incorporating affective state within an aggregation strategy* As noted in Section 21.6, there are many ways in which affective state can be used inside an aggregation strategy. We presented some initial ideas in this area, but extensive empirical research is required to investigate this further.
- *Explaining group recommendations: Transparency and Privacy* One might think that accurate predictions of individual satisfaction can also be used to improve the recommender's transparency: showing how satisfied other group members are could improve users' understanding of the recommendation process and perhaps make it easier to accept items they do not like. However, users' need for privacy is likely to conflict with their need for transparency. An important task of a group recommender system is to avoid embarrassment. Users often like to conform to the group to avoid being disliked (we discussed normative conformity as part of Section 21.5.2 on how others in the group can influence an individual's affective state). In [12], we have investigated how different group aggregation strategies may affect privacy. More work is needed on explanations of group recommendations, in particular on how to balance privacy with transparency and scrutability. Chapter 15 provides more detail on the different roles of explanations in recommender systems.

- *User interface design.* An individual's satisfaction with a group recommendation may be increased by good user interface design. For example, when showing an item, users could be shown what the next item will be (e.g. in a TV programme through a subtitle). This may inform users who do not like the current item that they will like the next one better.
- *Group aggregation strategies for cold-start problems.* In Section 21.7.2, we have sketched how group aggregation can be used to help solve the cold-start problem. However, our study in this area was very small, and a lot more work is required to validate and optimise this approach.
- *Dealing with uncertainty.* In this chapter, we have assumed that we have accurate profiles of individuals' preferences. For example, in Table 21.1, the recommender knows that Peter's rating of item B is 4. However, in reality we will often have probabilistic data. For example, we may know with 80% certainty that Peter's rating is 4. Adaptations of the aggregation strategies may be needed to deal with this. DeCampos et al try to deal with uncertainty by using Bayesian networks [4]. However, they have so far focussed on the Average and Plurality Voting strategies, not yet tackling the avoidance of misery and fairness issues.
- *Empirical Studies.* More empirical evaluations are vital to bring this field forwards. It is a challenge to design well-controlled, large scale empirical studies in a real-world setting, particularly when dealing with group recommendations and affective state. All research so far (including my own) has either been on a small scale, in a contrived setting or lacks control.

Table 21.9: Group recommender systems

System	Usage scenario	Classification			Strategy used
		Preferences known	Direct Experience	Group Active	
MUSICFX [15]	Chooses radio station in fitness centre based on people working out	Yes	Yes	No	Average Without Misery
POLYLENS [17]	Proposes movies for a group to view	Yes	No	No	Least Misery
INTRIGUE [2]	Proposes tourist attractions to visit for a group based on characteristics of sub-groups (such as children and the disabled)	Yes	No	No	Average
TRAVEL DECISION FORUM [7]	Proposes a group model of desired attributes of a planned joint vacation and helps a group of users to agree on these	Yes	No	Yes	Median
Yu's TV RECOMMENDER [20]	Proposes a TV program for a group to watch based on individuals' ratings for multiple features	Yes	No	No	Average
CATS [16]	Helps users to choose a joint holiday, based on individuals' critiques of proposed holidays	No	No	Yes	Counts requirements met Uses Without Misery
MASTHOFF'S GROUP RECOMMENDER [10, 12]	Chooses a sequence of music video clips for a group to watch based on individuals' ratings	Yes	Yes	No	Multiplicative etc

Acknowledgments

Judith Masthoff's research has been partly supported by Nuffield Foundation Grant No. NAL/00258/G.

References

1. Alfonseca, E., Carro, R.M., Martn, E., Ortigosa, A., Paredes, P.: The Impact of Learning Styles on Student Grouping for Collaborative Learning: A Case Study. *UMUAI 16* (2006) 377-401
2. Ardissono, L., Goy,A., Petrone, G., Segnan, M., Torasso, P.: Tailoring the Recommendation of Tourist Information to Heterogeneous User Groups. In S. Reich, M. Tzagarakis, P. De Bra (eds.), *Hypermedia: Openness, Structural Awareness, and Adaptivity*, International Workshops OHS-7, SC-3, and AH-3. *Lecture Notes in Computer Science 2266*, Springer Verlag, Berlin (2002) 280-295
3. Asch,S.E.:Studies of independence and conformity: a minority of one against a unanimous majority. *Psychol. Monogr.* 70 (1956) 1-70
4. de Campos, L.M., Fernandez-Luna, J.M., Huete, J.F., Rueda-Morales, M.A.: Managing uncertainty in group recommending processes. *UMUAI 19* (2009) 207-242
5. Harrer, A., McLaren, B.M., Walker, E., Bollen L., Sewall, J.: Creating Cognitive Tutors for Collaborative Learning: Steps Toward Realization. *UMUAI 16* (2006) 175-209
6. Introne, J., Alterman,R.: Using Shared Representations to Improve Coordination and Intent Inference. *UMUAI 16* (2006) 249-280
7. Jameson, A.: More than the Sum of its Members: Challenges for Group Recommender Systems. *International Working Conference on Advanced Visual Interfaces*, Gallipoli, Italy (2004)
8. Jameson, A., Smyth, B.: Recommendation to groups. In: Brusilovsky, P., Kobsa, A., Njedi, W. (Eds). *The Adaptive Web Methods and Strategies of Web Personalization*. Springer (2007) 596-627
9. Masthoff, J.: Modeling the multiple people that are me. In: P. Brusilovsky, A. Corbett, and F. de Rosis (eds.) *Proceedings of the 2003 User Modeling Conference*, Johnstown, PA. Springer Verlag, Berlin (2003) 258-262
10. Masthoff, J.: Group Modeling: Selecting a Sequence of Television Items to Suit a Group of Viewers. *UMUAI 14* (2004) 37-85
11. Masthoff, J.: Selecting News to Suit a Group of Criteria: An Exploration. *4th Workshop on Personalization in Future TV - Methods, Technologies, Applications for Personalized TV*, Eindhoven, the Netherlands (2004)
12. Masthoff, J., Gatt, A.: In Pursuit of Satisfaction and the Prevention of Embarrassment: Affective state in Group Recommender Systems. *UMUAI 16* (2006) 281-319
13. Masthoff, J.: The user as wizard: A method for early involvement in the design and evaluation of adaptive systems. *Fifth Workshop on User-Centred Design and Evaluation of Adaptive Systems* (2006).
14. Masthoff, J., Vasconcelos, W.W., Aitken, C., Correa da Silva, F.S.: Agent-Based Group Modelling for Ambient Intelligence. *AISB symposium on Affective Smart Environments*, Newcastle, UK (2007)
15. McCarthy, J., Anagnost, T.: MusicFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts. *CSCW*, Seattle, WA. (1998) 363-372
16. McCarthy, K., McGinty, L., Smyth, B., Salamo, M.: The needs of the many: A case-based group recommender system. *European Conference on Case-Based Reasoning*, Springer (2006) 196-210

17. O' Conner, M., Cosley, D., Konstan, J.A., Riedl, J.: PolyLens: A Recommender System for Groups of Users. ECSCW, Bonn, Germany (2001) 199-218. As accessed on <http://www.cs.umn.edu/Research/GroupLens/poly-camera-final.pdf>
18. Read, T., Barros, B., Brcena, E., Pancorbo, J.: Coalescing Individual and Collaborative Learning to Model User Linguistic Competences. UMUAI 16 (2006) 349-376
19. Suebnukarn, S., Haddawy, P.: Modeling Individual and Collaborative Problem-Solving in Medical Problem-Based Learning. UMUAI 16 (2006) 211-248
20. Yu, Z., Zhou, X., Hao, Y. Gu, J.: TV Program Recommendation for Multiple Viewers Based on User Profile Merging. UMUAI 16 (2006) 63-82

Part V
Advanced Algorithms

Chapter 22

Aggregation of Preferences in Recommender Systems

Gleb Beliakov, Tomasa Calvo and Simon James

Abstract This chapter gives an overview of aggregation functions toward their use in recommender systems. Simple aggregation functions such as the arithmetic mean are often employed to aggregate user features, item ratings, measures of similarity, etc., however many other aggregation functions exist which could deliver increased accuracy and flexibility to many systems. We provide definitions of some important families and properties, sophisticated methods of construction, and various examples of aggregation functions in the domain of recommender systems.

22.1 Introduction

Aggregation of preferences, criteria or similarities happens at various stages in recommender systems. Typically such aggregation is done by using either the arithmetic mean or maximum/minimum functions. Many other aggregation functions which would deliver flexibility and adaptability towards more relevant recommendations are often overlooked. In this chapter we will review the basics of aggregation functions and their properties, and present the most important families, including generalized means, Choquet and Sugeno integrals, ordered weighted averaging, triangular norms and conorms, as well as bipolar aggregation functions. Such functions can model various interactions between the inputs, conjunctive, disjunctive and mixed behavior. Following, we present different methods of construction of aggregation functions, based either on analytical formulas, algorithms, or empirical data. We discuss how parameters of aggregation functions can be fitted to observed

Gleb Beliakov and Simon James
School of Information Technology, Deakin University
221 Burwood Hwy, Burwood 3125, Australia, e-mail: gleb@deakin.edu.au, sjames@deakin.edu.au

Tomasa Calvo
Departamento de Ciencias de la Computación, Universidad de Alcalá
28871-Alcalá de Henares (Madrid), Spain. e-mail: tomasa.calvo@uah.es

data, while preserving these essential properties. By replacing the arithmetic mean with more sophisticated, adaptable functions, by canceling out redundancies in the inputs, one can improve the quality of automatic recommendations, and tailor recommender systems to specific domains.

22.2 Types of Aggregation in Recommender Systems

In general, recommender systems (RS) guide users to *items of interest* selected from vast databases of electronic objects and information. The orientation toward the presentation of personalized item-subsets distinguishes RS conceptually from similar processes such as internet filtering, with the RS drawing on a number of user-specific justifications in order to generate individualized recommendations. Since their inception, the use of RS has expanded rapidly with existing applications that recommend movies [34], web-pages [5], news articles [36], medical treatments [14, 31], music and other products [32, 40].

Clearly, the justifications used to recommend an item will depend on the specific application and the way data is collected and used by the system. Recommendations based on justifications concerning item features can be broadly classified as *content-based* (CB), whereas recommendations that utilize user similarity are referred to as *collaborative* (CF) [1, 2]. It is useful to further identify demographic (DF), utility- (UB) and knowledge-based (KB) methods [16] as distinct from the usual perception of CB recommendation as anything that uses item-item similarity. The more recent literature has been characterized by a focus on hybrid systems (HS), which combine two or more of these approaches.

Collaborative methods use the item preferences or ratings of similar users as justification for recommendation. This type of RS has been successful for e-commerce sites like *Amazon.com* [32] where interest is better inferred through similar taste than vague or subjective item descriptions. Consider a periphery genre like *Indie* music, which is defined loosely by its separation from the mainstream. As the genre encompasses a broad range of styles, Indie artists may have little in common besides their fans.¹ Aggregation functions (usually the simple or weighted average) are often employed to aggregate the ratings or preferences of similar users, however they can also be used to determine user similarity and help define *neighborhoods* (see also Chapter 4 of this book).

Content-based filtering methods form justifications by matching item-features to user profiles. For instance, a news recommender may build a profile for each user that consists of *keywords* and the interest in an unseen news item can be predicted by the number of keywords in the story that correspond to those in the user's profile. The way aggregation functions are used (and whether they are used) for content-based methods depends on the nature of the profile that is given to each user and

¹ It is interesting that Indie music fans, who thrive on a lack of conformity to pop culture and consumerism, have become an easy target-market for e-commerce sites that utilize collaborative RS. This is discussed in a recent literary article [26].

the description of items. We consider their use in item score computation, similarity computation and the construction of profiles.

Demographic filtering techniques assign each user to a demographic class based on their user profiles. Each demographic class has an associated user archetype or user stereotype that is then used to form justifications for recommendation. Rather than item history, user similarity here is more likely to be calculated from personal information and hence may be of lower dimension than most collaborative techniques. This makes nearest-neighbor or other classification and clustering tools particularly useful.

Rather than build long-term models, *utility-based* recommenders match items to the current needs of the users, taking into account their general tendencies and preferences. For instance, a user may be looking for a particular book, and it is known from past behavior that old hardback editions are preferred even if it takes longer to ship them. As is the case with content-based filtering, items can be described in the system by their features and, more specifically, the utility associated with each of those features. Aggregation can then be performed as it is with content-based filtering, although the user profiles and system information may differ.

Knowledge-based recommenders use background knowledge about associated and similar items to infer the needs of the user and how they can best be met. Knowledge-based methods will then draw not only on typical measures of similarity like correlation, but also on feature similarities that will interest the user. For instance, when a user indicates that he liked *A Good Year*, a KB recommender system might know that this film could be associated with either *A Beautiful Mind* (which also stars Russell Crowe) or *Jeux d'Enfants* (which also stars Marion Cotillard). Since the user has shown a preference for French films in the past, the system will assume that the user liked *A Good Year* because it featured Marion Cotillard, and recommend accordingly. It is pointed out in [16] that KB recommenders often draw on case-based reasoning approaches.

Hybrid recommender systems are employed to overcome the inherent drawbacks of each recommendation method. Burke [16] distinguishes weighted, mixed, switching, feature combination, cascade, feature augmentation and meta-level HS. Aggregation functions may be involved in the hybridization process - e.g. to combine different recommender scores in weighted HS or the features in feature combination HS. On the other hand, some of these hybrid methods are particularly useful in improving the performance of aggregation functions used at different stages. For instance, cascade methods use one filtering technique to reduce the size of the dataset, while feature augmentation HS might use one method to reduce its dimension. Similarity measures used for CF could be based on the similarity between user-specific aggregation functions (e.g. the similarity between weights and parameters) constructed in UB and CB frameworks. Similar meta-level HS are described in [16]. The switching criteria in switching HS could be based to some degree on aggregation functions, however here, as with mixed HS, their use is less likely.

Aggregation functions take multiple inputs and merge them into a single representative output. Simple examples of aggregation functions include the arithmetic mean, median, maximum and minimum. The use of more complicated and expres-

sive functions in RS would usually be motivated by the desire for more accurate recommendations, however in some circumstances aggregation functions might provide a practical alternative to other data processing methods. In the following subsections we will investigate the role of aggregation functions within different types of recommender system, indicating where they can be and have been applied.

22.2.1 Aggregation of Preferences in CF

Given a user u and a neighborhood of similar users $U_k = \{u_1, \dots, u_k\}$, the preference of u for an unseen item d_i can be predicted by aggregating the scores given by U_k . We will denote the predicted degree of interest, rating or preference by $R(u, d_i)$.

$$R(u, d_i) = \sum_{j=1}^k \text{sim}(u, u_j) R(u_j, d_i) \quad (22.1)$$

The function can be interpreted as a weighted arithmetic mean (WAM) where similarities between the user and similar users $\text{sim}(u, u_j) = w_j$ are the weights and $R(u_j, d_i) = x_j$ are the inputs to be aggregated. Provided $w_j, x_j \geq 0$, the function $R(u, d_i)$ is an aggregation function. Whilst the WAM is simply interpreted, satisfies many useful properties and is computationally inexpensive, other aggregation functions including power means (which can be non-linear) or the Choquet integral (which accounts for correlated inputs) may give a more accurate prediction of the users' ratings.

22.2.2 Aggregation of Features in CB and UB Recommendation

Where the profile is representable as a vector of feature preferences, $P_u = (p_1, \dots, p_n)$, items can then be described in terms of the degree to which they satisfy these features, i.e. $d_i = (x_1, \dots, x_n)$. Here, a value of $x_j = 1$ indicates that the preference p_j is completely satisfied by the item. P_u could also be a vector of keywords, in which case $x_j = 1$ might simply mean that the keyword p_j is mentioned once. The overall rating $R(u, d_i)$ of an item is then determined by aggregating the x_j ,

$$R(u, d_i) = f(x_1, \dots, x_n). \quad (22.2)$$

Eq. (22.2) is an aggregation function provided the function satisfies certain boundary conditions and is monotone with respect to increases in x_j . The $R(u, d_i)$ scores can be used to provide a ranking of unseen items, which can then be recommended. If the RS allows only one item to be shown, the how and why of this score evaluation becomes paramount. If the user is only likely to buy/view items when

all of their preferences are satisfied, a *conjunctive* function like the *minimum* should be used. On the other hand, if some of the preferences are unlikely to be satisfied simultaneously, e.g. the user is interested in drama and horror films, an *averaging* or *disjunctive* function might be more reliable. We present many examples of these broad classes of aggregation functions in Section 22.3.

In situations where it is practical to calculate item-item similarity, content-based filtering could also be facilitated using methods that mirror those in collaborative filtering [2]. In this case, a user profile might consist of all or a subset of previously rated/purchased items, $D = \{d_1, \dots, d_q\}$, and a measure of similarity is calculated between the unseen item d_i and those in D ,

$$R(u, d_i) = \sum_{j=1, (j \neq i)}^q \text{sim}(d_i, d_j) R(u, d_j). \quad (22.3)$$

In this case, content-based methods can benefit from the use of aggregation functions in determining item similarity and item neighborhoods as in Section 22.2.4.

22.2.3 Profile Construction for CB, UB

More sophisticated systems will assign a weight w_j to each of the preferences in P_u . To enhance the online-experience, many recommenders opt to learn the preferences (and weights) from online behavior, rather than ask the user to state them explicitly. The features of previously rated or purchased items can be aggregated to give an overall score for each preference. Given a preference p_j , let x_{ij} be the degree to which item d_i satisfies p_j , then the score $w(p_j)$ will be

$$w(p_j) = f(x_{1j}, \dots, x_{nj}). \quad (22.4)$$

Once all the preferences are determined, these $w(p_j)$ can be used to determine w_j for use in calculations such as Eq. (22.2).

22.2.4 Item and User Similarity and Neighborhood Formation

The behavior and accuracy of recommendation when using (22.1) will be largely dependent on how similarity (the weighting vector) is determined. The similarity between one user and another can be measured in terms of items previously rated or bought, or may be calculated based on known features associated with each user - e.g. the age, location and interests of a user may be known. The most commonly used measures of similarity, i.e. the weights in Eq. (22.1), are based on the cosine calculation [39] and Pearson's correlation coefficient [36]. Recently, other similarity measures have emerged such as fuzzy distance [4] and other recommender-specific

metrics [18, 3], based on the distribution of user ratings (see also Chapter 4 of this book).

Eq. (22.1) can also be considered within the framework of a *k-nearest-neighbors* (kNN) approach. Aggregation functions have been used to enhance the accuracy and efficiency of nearest-neighbor rules, with the OWA and Choquet integral providing the framework to model decaying weights and neighbor interaction [45, 12]. In the nearest-neighbor setting, similarity is tantamount to multi-dimensional proximity or distance. Euclidean distance was considered for measuring similarity for recommenders that use both ratings and personal information as inputs in [42]. Euclidean distance is just one type of metric, and may not capture the concept of distance well - for instance, where the data dimensions are correlated to some degree or even incommensurable. Metrics defined with the help of certain aggregation functions, including the OWA operator and Choquet integral, have been investigated in [41, 13] and could potentially prove useful for measuring similarity in some RS.

If we regard each value $sim(u, u_j)$ in (22.1) as a weight rather than a similarity, we can keep in mind that the problem of weight identification for various aggregation functions has been studied extensively. One method is to learn the weights from a data subset by using least-squares fitting techniques. For instance, given a set of mutually rated items $D = \{d_1, \dots, d_q\}$, the weights of a WAM can be fitted using the following program:

$$\begin{aligned} & \text{minimize } \sum_{i=1}^q \left(R(u, d_i) - \sum_{j=1}^k w_j R(u_j, d_i) \right)^2 \\ & \text{s.t.} \quad w_j \geq 0, \quad \forall j \\ & \quad \quad \sum_{j=1}^k w_j = 1. \end{aligned}$$

What is actually being determined is the vector of weights $\mathbf{w} = (w_1, \dots, w_k)$ that minimizes the residual errors. Each weight is then the importance of a given user u_j in accurately predicting $R(u, d_i)$. Non-linear functions such as the weighted geometric mean can also be fitted in this way. Such algorithms are relatively efficient in terms of computation time, and could be calculated either offline or in real-time depending on the RS and size of the database.

Alternatively, aggregation functions can be used to combine differing measures of similarity. Given a number of similarity measures $sim_1(u, u_1)$, $sim_2(u, u_1)$ etc., an overall measure of similarity can be obtained². This type of aggregated similarity was used in [20] for the recommendation of movies. In this example, cosine and correlation scores were combined using the product, which is a non-linear and conjunctive aggregation function.

² For similarity based on multiple criteria, see Chapter 24 of this book.

22.2.5 Connectives in Case-Based Reasoning for RS

The approach of many researchers in the fuzzy sets community has been to frame the recommendation problem in terms of case-based reasoning [23] where aggregation functions can be used as connectives. This results in rules of the form,

$$\text{If } d_{i1} \text{ is } A_1 \text{ AND } d_{i2} \text{ is } A_2 \text{ OR } \dots d_{in} \text{ is } A_n \text{ THEN } \dots \quad (22.5)$$

x_1, x_2, \dots, x_n denote the degrees of satisfaction of the rule predicates d_{i1} is A_1 , etc., and aggregation functions are used to replace the AND and OR operations. For instance, a user whose profile indicates a preference for comedies and action films might have a recommendation rule “IF the film is a comedy OR an action THEN recommend it.”³ Each genre can be represented as a fuzzy set with fuzzy connectives used to aggregate the degrees of satisfaction. The OR- and AND-type behavior are usually modeled by disjunctive and conjunctive aggregation functions respectively. In recommender systems, it has been shown that the property of noble reinforcement is desirable [44, 9]. This property allows many *strong* justifications to result in a *very strong* recommendation, or a number of *weak* justifications to reduce the recommendation if desired.

Functions that model (22.5) can be used to match items to profiles or queries in CB, UB and KB. In some demographic RS, items will be generically recommended to everyone in a given class, making the classification process the primary task of the RS. It may be desirable to classify users by the degree to which they satisfy a number of stereotypes, and in turn describe items in terms of their interest to each of these. For instance, a personal loan with an interest-free period could be very attractive to graduating students and somewhat attractive to new mothers, but of no interest to someone recently married. A user could partially satisfy each of these archetypes, requiring the system to aggregate the interest values in each demographic. This leads to rules similar to (22.5). “IF the item is interesting to students OR interesting to mothers THEN it will be interesting to user u ” or “IF user u is unmarried AND either a student OR mother, THEN recommend the item”.

22.2.6 Weighted Hybrid Systems

Given a number of recommendation scores obtained by using different methods, e.g. $R_{CF}(u, d_i)$, $R_{CB}(u, d_i)$, etc., an overall score can be obtained using

$$R(u, d_i) = f(R_{CF}(u, d_i), R_{CB}(u, d_i), \dots) \quad (22.6)$$

with f an aggregation function. The P-Tango system [19] uses a linear combination of collaborative and content-based scores to make its recommendations, and adjusts

³ We note here also that such rules could be used in any RS to decide *when* to recommend items, e.g. “IF user is inactive THEN recommend *something*”.

the weight according to the inferred user preferences. Aggregation of two or more methods can be performed using a number of functions with different properties and behavior. The use of non-linear or more complicated functions would enable some recommenders to fine-tune the ranking process, creating less irrelevant and more accurate predictions.

22.3 Review of Aggregation Functions

The purpose of aggregation functions is to combine inputs that are typically interpreted as degrees of membership in fuzzy sets, degrees of preference, strength of evidence, or support of a hypothesis, and so on. In this section, we provide preliminary definitions and properties before giving an introduction to some well known families.

22.3.1 Definitions and Properties

We will consider aggregation functions defined on the unit interval $f : [0, 1]^n \rightarrow [0, 1]$, however other choices are possible. The input value 0 is interpreted as no membership, no preference, no evidence, no satisfaction, etc., and naturally, an aggregation of n 0s should yield 0. Similarly, the value 1 is interpreted as full membership (strongest preference, evidence), and an aggregation of 1s should naturally yield 1.

Aggregation functions also require monotonicity in each argument, where an increase to any input cannot result in a decrease in the overall score.

Definition 22.1 (Aggregation function). An aggregation function is a function of $n > 1$ arguments that maps the (n -dimensional) unit cube onto the unit interval $f : [0, 1]^n \rightarrow [0, 1]$, with the properties

- (i) $f(\underbrace{0, 0, \dots, 0}_{n\text{-times}}) = 0$ and $f(\underbrace{1, 1, \dots, 1}_{n\text{-times}}) = 1$.
- (ii) $\mathbf{x} \leq \mathbf{y}$ implies $f(\mathbf{x}) \leq f(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in [0, 1]^n$.

For some applications, the inputs may have a varying number of components (for instance, some values can be missing). Particularly in the case of automated systems, it may be desirable to utilize functions defined for $n = 2, 3, \dots$ arguments with the same underlying property in order to give consistent aggregation results. Functions satisfying the following definition [33] may then be worth considering.

Definition 22.2 (Extended aggregation function). An extended aggregation function is a mapping

$$F : \bigcup_{n \in \{1, 2, \dots\}} [0, 1]^n \rightarrow [0, 1],$$

such that the restriction of this mapping to the domain $[0, 1]^n$ for a fixed n is an n -ary aggregation function f , with the convention $F(x) = x$ for $n = 1$.

Aggregation functions are classed depending on their overall behavior in relation to the inputs [17, 21, 22]. In some cases we require high inputs to compensate for low inputs, or that inputs may average each other. In other situations, it may make more sense that high scores reinforce each other and low inputs are essentially discarded.

Definition 22.3 (Classes). An aggregation function $f : [0, 1]^n \rightarrow [0, 1]$ is:

- Averaging* if it is bounded by $\min(\mathbf{x}) \leq f(\mathbf{x}) \leq \max(\mathbf{x})$;
- Conjunctive* if it is bounded by $f(\mathbf{x}) \leq \min(\mathbf{x})$;
- Disjunctive* if it is bounded by $f(\mathbf{x}) \geq \max(\mathbf{x})$;
- Mixed* otherwise.

The class of aggregation function to be used depends on how the inputs of the recommender system are interpreted and how sensitive or broad an output is desired. When aggregating recommendation scores in CF, the use of averaging functions ensures that the predicted interest in an item is representative of the central tendency of the scores. On the other hand, the semantics of some mixed aggregation functions makes their use appealing. For instance, MYCIN [14] is a classical expert system used to diagnose and treat rare blood diseases and utilizes a mixed aggregation function so that inputs of only high scores reinforce each other, while scores below a given threshold are penalized.

There are several studied properties that can be satisfied by aggregation functions, making them useful in certain situations. We provide definitions for those that are frequently referred to in the literature.

Definition 22.4. [Properties] An aggregation function $f : [0, 1]^n \rightarrow [0, 1]$ is:

- Idempotent* if for every $t \in [0, 1]$ the output is $f(t, t \dots, t) = t$;
- Symmetric* if its value does not depend on the permutation of the arguments, i.e., $f(x_1, x_2, \dots, x_n) = f(x_{P(1)}, x_{P(2)}, \dots, x_{P(n)})$ for every \mathbf{x} and every permutation $P = (P(1), P(2), \dots, P(n))$ of $(1, 2 \dots, n)$;
- Associative* if, for $f : [0, 1]^2 \rightarrow [0, 1]$, $f(f(x_1, x_2), x_3) = f(x_1, f(x_2, x_3))$ holds for all x_1, x_2, x_3 ;
- Shift-invariant* if for all $\lambda \in [-1, 1]$ and for all $\mathbf{x} = (x_1, \dots, x_n)$, $f(x_1 + \lambda, \dots, x_n + \lambda) = f(\mathbf{x}) + \lambda$ whenever $(x_1 + \lambda, \dots, x_n + \lambda) \in [0, 1]^n$ and $f(\mathbf{x}) + \lambda \in [0, 1]$;
- Homogeneous* if for all $\lambda \in [0, 1]$ and for all $\mathbf{x} = (x_1, \dots, x_n)$, $f(\lambda x_1, \dots, \lambda x_n) = \lambda f(\mathbf{x})$;
- Strictly monotone* if $\mathbf{x} \leq \mathbf{y}$ but $\mathbf{x} \neq \mathbf{y}$ implies $f(\mathbf{x}) < f(\mathbf{y})$;
- Lipschitz continuous* if there is a positive number M , such that for any two inputs $\mathbf{x}, \mathbf{y} \in [0, 1]^n$, $|f(\mathbf{x}) - f(\mathbf{y})| \leq Md(\mathbf{x}, \mathbf{y})$, where $d(\mathbf{x}, \mathbf{y})$ is a distance between \mathbf{x} and \mathbf{y} . The smallest such number M is called the Lipschitz constant of f .
- Has neutral elements* if there is a value $e \in [0, 1]$ such that $f(e, \dots, e, t, e, \dots, e) = t$ for every $t \in [0, 1]$ in any position.
- Has absorbing elements* if there is a value $a \in [0, 1]$ such that $f(x_1, \dots, x_{j-1}, a, x_{j+1}, \dots, x_n) = a$ for any \mathbf{x} with $x_j = a$.

22.3.1.1 Practical Considerations in RS

We will discuss some of the implications of each of these properties with some examples before providing the formal definitions of many important and extensively studied aggregation functions.

Idempotency All averaging aggregation functions, including the means, OWA and Choquet integral defined in Section 22.3.2, are idempotent⁴. The usual interpretation of this property is toward a representation of consensus amongst the inputs. However in some RS applications, e.g. when aggregating ratings in CF, the relative ranking of items is of more concern than the commensurability of input/output interpretations.

Example 22.1. The geometric mean $G(x, y) = \sqrt{xy}$ is idempotent, whereas the product $T_P(x, y) = xy$ is not. For any two objects, $d_1 = (x_1, y_1)$ and $d_2 = (x_2, y_2)$, however it follows that $G(d_1) > G(d_2)$ implies $T_P(d_1) > T_P(d_2)$.

Example 22.2. Let $d_1 = (0.5, 0.5)$, $d_2 = (0.2, 0.8)$. Using the geometric mean and the product to aggregate gives $G(d_1) = 0.5$, $G(d_2) = 0.4$, $T_P(d_1) = 0.25$, $T_P(d_2) = 0.16$. If d_i are item scores in CF, it might be better to interpret the outputs as the predicted ratings for user u , in which case we use G . If d_i are items described by the degree to which they satisfy two of the user preferences in UB filtering, the overall utility might be better indicated by T_P since we want most of the preferences satisfied.

Symmetry Symmetry is often used to denote equal importance with regard to the inputs. Weighted and non-weighted quasi-arithmetic means can be used depending on the situation. Although the ordered weighted averaging function (OWA) is defined with respect to a weighting vector, the inputs are pre-sorted into non-increasing order, hence it is symmetric regardless of \mathbf{w} .

Example 22.3. A collaborative RS considers an item rated by three similar users $d_i = (0.2, 0.7, 0.5)$. We consider using the weighting vector $\mathbf{w} = (0.6, 0.3, 0.1)$ with either an OWA function or a weighted arithmetic mean (WAM). In the case of the WAM, the weights suggest that user u_1 is very

⁴ Idempotency and averaging behavior are equivalent for aggregation functions due to the monotonicity requirement. This property is sometimes referred to as unanimity since the output agrees with each input when the inputs are unanimous.

similar to user u , and further that $\text{sim}(u, u_1) > \text{sim}(u, u_2) > \text{sim}(u, u_3)$. The aggregated score in this case would be $R(u, d_i) = \text{WAM}(d_i) = 0.6(0.2) + 0.3(0.7) + 0.1(0.5) = 0.38$ since u_1 didn't particularly like the item. If using the OWA, one interpretation of the weights suggests that user u will like the item if one or two similar users liked it, no matter which of the similar users it is. This gives $R(u, d_i) = \text{OWA}(d_i) = 0.6(0.7) + 0.3(0.5) + 0.1(0.2) = 0.59$.

Associativity Associativity is a useful property for automatic computation as it allows functions to be defined recursively for any dimension. This is potentially useful for collaborative RS where data sparsity is a problem. The same function could be used to evaluate one item rated by 10 similar users, and another rated by 1000 similar users. T-norms and t-conorms, uninorms and nullnorms are associative, however the quasi-arithmetic means are not.

Example 22.4. A collaborative RS uses personal information to determine similarity between users (i.e. the values do not need to be reassessed every time a new item is rated). Rather than store an $\text{items} \times \text{users}$ matrix for each user, the system uses a uninorm $U(x, y)$ to aggregate the similar user ratings and stores a single vector of aggregated item scores $\mathbf{d} = (U(d_i), \dots, U(d_n))$. When a new item score x_{ij} is added, the system aggregates $U(U(d_i), x_{ij})$ and stores this instead of $U(d_i)$. The advantage here is that neither the previous scores nor the number of times the item is rated is required in order to update the predicted rating.

Shift-invariance and Homogeneity The main advantage of shift-invariant and homogeneous functions is that translating or dilating the domain of consideration will not affect relative orderings of aggregated inputs. The weighted arithmetic mean, OWA and Choquet integral are all shift invariant, so it makes no difference whether inputs are considered on $[0, 100]$ or $[1, 7]$, as long as the inputs are commensurable.

Strict monotonicity Strict monotonicity is desired in applications where the number of items to be shown to the user is limited. Weighted arithmetic means and OWA functions are strictly monotone when $w_j > 0, \forall j$, while geometric and harmonic means are strict for $\mathbf{x} \in]0, 1]^n$. Aggregation functions which are not strict, the *maximum* function for instance, could not distinguish between an item $d_1 = (0.3, 0.8)$ and another $d_2 = (0.8, 0.8)$.

Example 22.5. A holiday recommendation site uses a utility-based RS where the Łukasiewicz t-conorm $S_L(x, y) = \min(x + y, 1)$ is used to aggregate item features. It is able to show the user every item $S_L(d_i) = 1$ by notifications through e-mail. It doesn't matter that $d_1 = (0.3, 0.8)$ and $d_2 = (0.8, 0.8)$, since both of them are predicted to completely satisfy the user's needs.

Lipschitz continuity Continuity, in general, ensures that small input inaccuracies cannot result in drastic changes in output. Such a property is especially important in RS where the inputs, whether item descriptions or user ratings, are likely to be inexact. Some functions only violate this property on a small portion of the domain (See Example 22.6). As long as this is taken into account when the RS considers the recommendation scores, the function might still be suitable.

Example 22.6. The geometric mean $G(x, y) = \sqrt{xy}$ fails the Lipschitz property since the rate-of-change is unbounded when one of the inputs is close to zero. On the other hand, the harmonic mean, given by $H(x, y) = \frac{2xy}{x+y}$ (in the two-variate case) is Lipschitz continuous with Lipschitz constant $M = 2$.

Neutral and absorbent elements Absorbent elements could be useful in RS to ensure that certain items always or never get recommended. For example, a UB recommender could remove every item from consideration which has any features that score zero, or definitely recommend items which completely satisfy one of the user preferences. T-norms and t-conorms each have absorbent elements. Incorporating functions with neutral elements into a recommender system that aggregates user ratings (in either a CF or CB framework) allows values to be specified which will not affect recommendation scores. A movie that is liked by many people, for instance, would usually have its overall approval rating reduced by someone who was indifferent toward it but still required to rate it. If a neutral value exists it will not influence the aggregated score.

22.3.2 Aggregation Families

22.3.2.1 Quasi-Arithmetic Means

The family of weighted quasi-arithmetic means generalizes the power mean, which in turn includes other classical means such as the arithmetic and geometric mean as special cases (see [15] for an overview of means).

Definition 22.5 (Weighted quasi-arithmetic means). For a given strictly monotone and continuous function $g : [0, 1] \rightarrow [-\infty, +\infty]$, called a generating function or generator, and a weighting vector $\mathbf{w} = (w_1, \dots, w_n)$, the weighted quasi-arithmetic mean is the function

$$M_{\mathbf{w},g}(\mathbf{x}) = g^{-1} \left(\sum_{i=1}^n w_j g(x_j) \right), \tag{22.7}$$

where $\sum w_j = 1$ and $w_j \geq 0 \forall j$.

Special cases include:

Arithmetic means $WAM_{\mathbf{w}} = \sum_{j=1}^n w_j x_j, \quad g(t) = t;$

Geometric means $G_{\mathbf{w}} = \prod_{j=1}^n x_j^{w_j}, \quad g(t) = \log(t);$

Harmonic means $H_{\mathbf{w}} = \left(\sum_{j=1}^n \frac{w_j}{x_j} \right)^{-1}, \quad g(t) = \frac{1}{t};$

Power means $M_{\mathbf{w},[r]} = \left(\sum_{j=1}^n w_j x_j^r \right)^{\frac{1}{r}}, \quad g(t) = t^r.$

The term *mean* is usually used to imply averaging behavior. Quasi-arithmetic means defined with respect to a weighting vector with all $w_j = \frac{1}{n}$ are symmetric, and asymmetric otherwise. Usually the weight allocated to a particular input is indicative of the importance of that particular input. All power means (including $WAM_{\mathbf{w}}, G_{\mathbf{w}}$ and $H_{\mathbf{w}}$) are idempotent, homogeneous and strictly monotone on the open interval $]0, 1[^n$, however only the weighted arithmetic mean is shift-invariant. The geometric mean is not Lipschitz continuous⁵.

22.3.2.2 OWA Functions

Ordered weighted averaging functions (OWA) are also averaging aggregation functions, which associate a weight not with a particular input, but rather with its relative value or order compared to others. They have been introduced by Yager [43] and have become very popular in the fuzzy sets community.

Definition 22.6 (OWA). Given a weighting vector \mathbf{w} , the OWA function is

$$OWA_{\mathbf{w}}(\mathbf{x}) = \sum_{j=1}^n w_j x_{(j)},$$

where the $(.)$ notation denotes the components of \mathbf{x} being arranged in non-increasing order $x_{(1)} \geq x_{(2)} \geq \dots \geq x_{(n)}$.

Special cases of the OWA operator, depending on the weighting vector \mathbf{w} include:

⁵ The Lipschitz property for quasi-arithmetic means and other generated aggregation functions is explored in [11].

Arithmetic mean where all the weights are equal, i.e. all $w_j = \frac{1}{n}$;

Maximum function for $\mathbf{w} = (1, 0, \dots, 0)$;

Minimum function for $\mathbf{w} = (0, \dots, 0, 1)$;

Median function for $w_j = 0$ for all $j \neq m$, $w_m = 1$ if $n = 2m + 1$ is odd, and $w_j = 0$ for all $j \neq m, m + 1$, $w_m = w_{m+1} = 0.5$ if $n = 2m$ is even.

The OWA function is a piecewise linear idempotent aggregation function. It is symmetric, homogeneous, shift-invariant, Lipschitz continuous and strictly monotone if $w_j > 0, \forall j$.

22.3.2.3 Choquet and Sugeno integrals

Referred to as fuzzy integrals, the Choquet integral and the Sugeno integral are averaging aggregation functions defined with respect to a fuzzy measure. They are useful for modeling interactions between the input variables x_j .

Definition 22.7 (Fuzzy measure). Let $\mathcal{N} = \{1, 2, \dots, n\}$. A discrete fuzzy measure is a set function⁶ $v : 2^{\mathcal{N}} \rightarrow [0, 1]$ which is monotonic (i.e. $v(A) \leq v(B)$ whenever $A \subseteq B$) and satisfies $v(\emptyset) = 0, v(\mathcal{N}) = 1$. Given any two sets $A, B \subseteq \mathcal{N}$, fuzzy measures are said to be:

Additive where $v(A \cup B) = v(A) + v(B)$, for $v(A \cap B) = \emptyset$;

Symmetric where $|A| = |B| \rightarrow v(A) = v(B)$;

Submodular if $v(A \cup B) - v(A \cap B) \leq v(A) + v(B)$;

Supermodular if $v(A \cup B) - v(A \cap B) \geq v(A) + v(B)$;

Subadditive if $v(A \cup B) \leq v(A) + v(B)$ whenever $A \cap B = \emptyset$;

Superadditive if $v(A \cup B) \geq v(A) + v(B)$ whenever $A \cap B = \emptyset$;

Decomposable if $v(A \cup B) = f(v(A), v(B))$ whenever $A \cap B = \emptyset$, for a given function $f : [0, 1]^2 \rightarrow [0, 1]$;

Sugeno (λ -fuzzy measure) if v is decomposable with $f = v(A) + v(B) + \lambda v(A)v(B)$, $\lambda \in]-1, \infty[$.

The behavior of the Sugeno and Choquet integral depends on the values and properties of the associated fuzzy measure. The fuzzy measure used to define the Choquet integral can be interpreted as a weight allocation, not merely to individual inputs but rather to each subset of inputs. It may be that there are redundancies among the inputs, or that certain inputs complement each other.

Definition 22.8 (Choquet integral). The discrete Choquet integral with respect to a fuzzy measure v is given by

$$C_v(\mathbf{x}) = \sum_{j=1}^n x_{(j)} [v(\{k | x_k \geq x_{(j)}\}) - v(\{k | x_k \geq x_{(j+1)}\})], \tag{22.8}$$

⁶ A set function is a function whose domain consists of all possible subsets of \mathcal{N} . For example, for $n = 3$, a set function is specified by $2^3 = 8$ values at $v(\emptyset)$, $v(\{1\})$, $v(\{2\})$, $v(\{3\})$, $v(\{1, 2\})$, $v(\{1, 3\})$, $v(\{2, 3\})$, $v(\{1, 2, 3\})$.

where (\cdot) in this case denotes the components of \mathbf{x} being arranged in non-decreasing order such that $(x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(n)})$ (note that this is opposite to OWA).

Special cases of the Choquet integral include weighted arithmetic means and the OWA function where the fuzzy measure is additive or symmetric respectively. Sub-modular fuzzy measures result in Choquet integrals which are concave, the upshot of which is that increases to lower inputs affect the function more than increases to higher inputs. Conversely, supermodular fuzzy measures result in convex functions. Choquet integrals are idempotent, homogeneous, shift-invariant and strictly monotone where $A \subsetneq B \rightarrow v(A) < v(B)$. Where the fuzzy measure is symmetric, the function will obviously satisfy the symmetry property.

The Choquet integral has been predominantly used for numerical inputs, the Sugeno integral defined below is useful where the inputs are ordinal. It also uses fuzzy measures for its definition.

Definition 22.9 (Sugeno integral). The Sugeno integral with respect to a fuzzy measure v is given by

$$S_v(\mathbf{x}) = \max_{j=1, \dots, n} \min\{x_{(j)}, v(H_j)\}, \tag{22.9}$$

where (\cdot) denotes a non-decreasing permutation of the inputs such that $(x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(n)})$ (the same as with the Choquet integral), and $H_j = \{(j), \dots, (n)\}$.

Certain indices have been introduced in order to better understand the behavior of the Choquet and Sugeno integrals. In particular, the Shapley value gives an indication of the overall importance of a given input, while the interaction index between two inputs shows to what extent they are redundant or complimentary.

Definition 22.10 (Shapley value).

Let v be a fuzzy measure. The Shapley index for every $i \in \mathcal{N}$ is

$$\phi(i) = \sum_{A \subseteq \mathcal{N} \setminus \{i\}} \frac{(n - |A| - 1)! |A|!}{n!} [v(A \cup \{i\}) - v(A)].$$

The Shapley value is the vector $\phi(v) = (\phi(1), \dots, \phi(n))$.

Definition 22.11 (Interaction index). Let v be a fuzzy measure. The interaction index for every pair $i, j \in \mathcal{N}$ is

$$I_{ij} = \sum_{A \subseteq \mathcal{N} \setminus \{i, j\}} \frac{(n - |A| - 2)! |A|!}{(n - 1)!} [v(A \cup \{i, j\}) - v(A \cup \{i\}) - v(A \cup \{j\}) + v(A)].$$

Where the interaction index is negative, there is some redundancy between the two inputs. Where it is positive, the inputs complement each other to some degree and their weight together is worth more than their combined individual weights.

22.3.2.4 T-Norms and T-Conorms

The prototypical examples of conjunctive and disjunctive aggregation functions are so-called triangular norms and conorms respectively (t-norms and t-conorms) [28]. Given any t-norm $T : [0, 1]^2 \rightarrow [0, 1]$, there is a dual function which is a t-conorm S , with

$$S(x, y) = 1 - T(1 - x, 1 - y)$$

and vice-versa. T-norms and t-conorms are hence often studied in parallel, as many properties concerning S can be determined from T . Triangular norms are associative, symmetric with the neutral element $e = 1$, whereas triangular conorms are associative, symmetric and have the neutral element $e = 0$. The definitions of the four basic t-norms and t-conorms are provided below.

Definition 22.12 (The four basic t-norms). The two-variate cases for the four basic t-norms are given by

<i>Minimum</i>	$T_{min}(x, y) = \min(x, y);$
<i>Product</i>	$T_P(x, y) = xy;$
<i>Lukasiewicz t-norm</i>	$T_L(x, y) = \max(x + y - 1, 0);$
<i>Drastic Product</i>	$T_D(x, y) = \begin{cases} 0, & \text{if } (x, y) \in [0, 1]^2, \\ \min(x, y) & \text{otherwise.} \end{cases}$

Definition 22.13 (The four basic t-conorms). The two-variate cases for the four basic t-conorms are given by

<i>Maximum</i>	$S_{max}(x, y) = \max(x, y);$
<i>Probabilistic Sum</i>	$S_P(x, y) = x + y - xy;$
<i>Lukasiewicz t-conorm</i>	$S_L(x, y) = \min(x + y, 1);$
<i>Drastic Product</i>	$S_D(x, y) = \begin{cases} 1, & \text{if } (x, y) \in [0, 1]^2, \\ \max(x, y) & \text{otherwise.} \end{cases}$

There are families of parameterized t-norms and t-conorms that include the above as special or limiting cases. These families are defined with respect to generating functions and are known as Archimedean t-norms.

Definition 22.14 (Archimedean t-norm). A t-norm is called Archimedean if for each $(a, b) \in]0, 1[^2$ there is an $n = \{1, 2, \dots\}$ with $T(\overbrace{a, \dots, a}^{n\text{-times}}) < b$.

For t-conorms, the inequality is reversed, i.e. the t-conorm $S > b$. Continuous Archimedean t-norms can be expressed by use of their generators as

$$T(x_1, \dots, x_n) = g^{(-1)}(g(x_1) + \dots + g(x_n)),$$

where $g : [0, 1] \rightarrow [0, \infty]$ with $g(1) = 0$ is a continuous, strictly decreasing function and $g^{(-1)}$ is the pseudo inverse of g , i.e.,

$$g^{(-1)}(x) = g^{-1}(\min(g(1), \max(g(0), x))).$$

Archimedean families include Schweizer-Sklar, Hamacher, Frank, Yager, Dombi, Aczel-Alsina, Mayor-Torrens and Weber-Sugeno t-norms and t-conorms.

22.3.2.5 Nullnorms and Uninorms

In some situations, it may be required that high input values reinforce each other whereas low values pull the overall output down. In other words, the aggregation function has to be disjunctive for high values, conjunctive for low values, and perhaps averaging if some values are high and some are low. This is typically the case when high values are interpreted as “positive” information, and low values as “negative” information.

In other situations, it may be that aggregation of both high and low values moves the output towards some intermediate value. Thus certain aggregation functions need to be conjunctive, disjunctive or averaging in different parts of their domain.

Uninorms and nullnorms are typical examples of such aggregation functions, but there are many others. We provide the definitions below.

Definition 22.15 (Nullnorm). A nullnorm is a bivariate aggregation function $V : [0, 1]^2 \rightarrow [0, 1]$ which is associative, symmetric, such that there exists an element a belonging to the open interval $]0, 1[$ verifying

$$\begin{aligned} \forall t \in [0, a], \quad V(t, 0) &= t, \\ \forall t \in [a, 1], \quad V(t, 1) &= t. \end{aligned}$$

Definition 22.16 (Uninorm). A uninorm is a bivariate aggregation function $U : [0, 1]^2 \rightarrow [0, 1]$ which is associative, symmetric and has a neutral element e belonging to the open interval $]0, 1[$.

Some uninorms can be built from generating functions in a similar way to quasi-arithmetic means and Archimedean t-norms. These are called representable uninorms.

Definition 22.17 (Representable uninorm). Let $u : [0, 1] \rightarrow [-\infty, +\infty]$ be a strictly increasing bijection verifying $g(0) = -\infty, g(1) = +\infty$ such that $g(e) = 0$ for some $e \in]0, 1[$.

- The function given by

$$U(x, y) = \begin{cases} g^{-1}(g(x) + g(y)), & \text{if } (x, y) \in [0, 1]^2 \setminus \{(0, 1), (1, 0)\}, \\ 0, & \text{otherwise,} \end{cases}$$

is a conjunctive uninorm with the neutral element e , known as a *conjunctive representable uninorm*.

- The function given by

$$U(x,y) = \begin{cases} g^{-1}(g(x) + g(y)), & \text{if } (x,y) \in [0, 1]^2 \setminus \{(0, 1), (1, 0)\}, \\ 1, & \text{otherwise,} \end{cases}$$

is a disjunctive uninorm with the neutral element e , known as a *disjunctive representable uninorm*.

The $3 - \Pi$ function is an example of a representable uninorm [46]. It uses a generating function $g(x) = \ln(\frac{x}{1-x})$ and is used by the expert system PROSPECTOR [24] for combining uncertainty factors.

$$f(\mathbf{x}) = \frac{\prod_{i=1}^n x_i}{\prod_{i=1}^n x_i + \prod_{i=1}^n (1 - x_i)},$$

with the convention $\frac{0}{0} = 0$. It is conjunctive on $[0, \frac{1}{2}]^n$, disjunctive on $[\frac{1}{2}, 1]^n$ and averaging elsewhere. It is associative, with the neutral element $e = \frac{1}{2}$, and discontinuous on the boundaries of $[0, 1]^n$.

22.4 Construction of Aggregation Functions

There are infinitely many aggregation functions. The question is how to choose the most suitable aggregation function for a specific application. Sometimes one function may suffice for all components of the application, at other times a different type of aggregation may be employed at various stages. The following considerations should be helpful.

22.4.1 Data Collection and Preprocessing

The type of data, and how it is collected affects the way it can be aggregated to form justifications. If users could thoughtfully provide accurate scores on a consistent scale for each item, or numerical descriptions of themselves with their preferences expressed to a degree of certainty, an RS could quite comfortably make some relevant recommendations. Of course, the aesthetic preference is usually to limit the explicit information required from the user and hence enhance the interactive experience. We will briefly consider the different types of data that systems are able to obtain and how this might affect the suitability of certain aggregation functions.

Ordinal Data CF recommenders that ask for explicit ratings information will usually do so on a finite ordinal scale - e.g. $\{1 = \text{didn't like it!}, \dots, 5 = \text{loved it!}\}$. On the other hand, it may be possible to convert user actions into ordinal values as part of their profile - e.g. $\{\text{regularly views, sometimes views, etc.}\}$. Where there is an ordinal scale, these values can be turned into numbers and aggregated.

For non-homogeneous functions and those which lack the shift-invariance property, it will be necessary to express these ordinal values on the unit interval. The coarseness of the aggregated values may make the difference between, say, the weighted arithmetic mean and the geometric mean negligible. Examples of aggregation functions particularly suitable for the aggregation of ordinal data are the Sugeno integral and the induced OWA.

- The Sugeno integral S_γ (Def. 22.9), is a function which is able to process ordinal data and take into account interactions. It is necessary for the fuzzy measure values to be on the same ordinal scale as the input values. The Sugeno integral is capable of modeling median-type functions as well as minimum and maximum functions, and has the advantage of expressing outputs as ordinal values.
- The induced OWA function [45] is capable of modeling nearest-neighbor approaches even if the similarity is expressed as ordinal values, although it does require the ratings to be expressed numerically.

Numerical Data Where a system is capable of representing user inputs or actions as numerical data, it is useful to take into account whether these values are accurate, whether they are commensurate, and whether they are independent. Functions such as the geometric mean have a higher rate of change when input values are high than the arithmetic mean. This can help provide granularity to the outputs, however it also means that errors on this portion of the domain will influence the recommendation accuracy. In CF, two users might have similar preferences however one may consistently overrate items. In these cases, it might make sense to standardize the ratings before aggregating so the values between users are comparable. The use of the WAM implies independence between inputs, however other averaging functions, especially the Choquet integral, can express interaction and correlation either among certain inputs or relative scores.

Categorical Data In some cases, the use of categorical data may make it impractical to use aggregation functions. If there is no order between categories, it is meaningless to take the *average* or *maximum*, and other techniques may be useful for establishing similarity between users etc. It may be possible to transform the categorical data, for example, by the degree to which it contributes towards a certain archetype in DF.

There could however, be variations: some components of the vectors associated with d_i could be missing - e.g. ratings in CF, or the inputs $d_i = (x_1, \dots, x_n)$ may have varying dimension by construction. In other cases, the uncertainty associated with some of the inputs or outputs could prescribe a range of values - e.g., the interval $[6, 8]$ as a rating for a film. Associative or generating functions are capable of aggregating inputs of varying dimension with some consistency in terms of the properties, while either transformations of the data or interval-valued functions can be employed in the latter case.

22.4.2 *Desired Properties, Semantics and Interpretation*

The first step in choosing an aggregation function once the data structure is known is usually to decide which class of either averaging, conjunctive, disjunctive or mixed is desired. As discussed in Section 22.3.1.1, sometimes it will be more important to have a function which sorts items into order of preference than one which gives easily interpreted outputs. We consider four functions whose semantics can be used to decide which class of function is required:

Minimum (conjunctive) The minimum uses the minimum input as its output. This means the function can only return a high output if all the inputs are high. Such aggregation is useful for certain KB or UB systems using Eq. (22.5) or even CB where it is desired that all the inputs be satisfied. Functions such as the product (T_P) have an accumulative effect for any output which is not perfect, so might be less useful than the min when the dimension is high.

Maximum (disjunctive) Whereas the minimum models AND-like aggregation, disjunctive functions model OR. This type of aggregation results in outputs which are equal to or greater than the highest input. This is useful in KB, UB or CB as well if there are multiple preferences or criteria and one good score is enough justification for recommendation. Consider Example 22.7.

Example 22.7. A user of a CB news recommender has the keywords {Haruki Murakami, X-Men, bushfires, mathematics, Jupiter orbit} associated with her profile. It is unlikely that any one news story will be highly relevant to all or even any few of these keywords, so the RS uses disjunctive aggregation as a basis for recommendation.

Arithmetic Mean (averaging) When aggregating user ratings in CF or item features in CB it is reasonable to assume that although scores will vary, if enough inputs are used, the output will be reliable. We do not want the recommendations to be severely affected by an isolated user that is unsatisfied with every item he purchases, or a single feature among twenty or so that is completely satisfied.

Uninorm (mixed) In cases where different behavior is required on different parts of the domain, a mixed aggregation function may be required. This can be as straightforward as deciding that only values with *all* high inputs should be high, or it could be that the bounded behavior affects the accuracy of the function. The use of a uninorm, for instance, allows high values to push the score up and low values push the score down. An item with consistently high scores would be preferred to one with mostly high scores but one or two low ones.

Certain properties of aggregation functions might also make them appealing. Table 22.1 lists the main aggregation functions we have presented and whether they always, or under certain circumstances, satisfy the properties detailed in Section 22.4.

Table 22.1: Aggregation Functions and Properties

Property	WAM_w	G_w	H_w	$M_{w,[r]}$	C_v	S_v	OWA_w	max	min	T_P	T_L	U	V
idempotent	◆	◆	◆	◆	◆	◆	◆	◆	◆				
symmetric	◇	◇	◇	◇	◇	◇	◆	◆	◆	◆	◆	◆	◆
asymmetric	◇	◇	◇	◇	◇	◇							
associative								◆	◆	◆	◆	◆	◆
strictly monotone	◇			◇	◇	◇	◇						
shift-invariance	◆			◇	◆	◆	◆	◆	◆				
homogeneous	◆	◆	◆	◆	◆	◆	◆	◆	◆				
Lipschitz continuous	◆		◆	◇	◆	◆	◆	◆	◆	◆	◆	◆	△
neutral elements							◇	◆	◆	◆	◆	◆	◆
absorbent elements		◆	◆				◇	◆	◆	◆	◆	◆	◆

◆ = always ◇ = depending on weights △ = depends on T,S used

22.4.3 Complexity and the Understanding of Function Behavior

In some cases, simple functions such as the WAM will be adequate to meet the goals of recommendation, with potential improvements to the RS lying in other directions. Due to its properties, the WAM is quite a robust and versatile function. It is not biased towards high or low scores, it does not accumulate the effects of errors, it is computationally inexpensive and its common use makes it well understood and easily interpreted. We present the power mean and Choquet integral as two example alternatives whose properties might make them more appropriate in certain situations.

The power mean The power mean is a parameterized function, capable of expressing functions that graduate from the minimum to the maximum including the WAM. This makes it immediately useful when fitting techniques are at our disposal, since we can use the one process to identify any number of functions as the best candidate. Consider the harmonic mean $M_{w,[-1]}$ and the quadratic mean $M_{w,[2]}$. The harmonic mean cannot give an output greater than zero if even one of the inputs is zero. This has the nice interpretation of only allowing items to be considered that at least partially satisfy every criteria, however it is not conjunctive, so still gives a score somewhere between the highest and lowest inputs. The harmonic mean is also concave and its output is equal to or less than the WAM for any choice of d_i . This allows less compensation for low inputs, so items must satisfy more of the criteria overall to rate highly. On the other hand, the quadratic power mean tends more towards high scores, favoring items that have a few very high scores which compensate more for low-scoring features or ratings.

The Choquet integral As with the power mean, the Choquet integral is capable of expressing functions ranging between the minimum and maximum. The use

of the Choquet integral is most interesting in asymmetric situations where there tends to be some correlation. For example, in a KB recommender, sometimes preferences will be contradictory while at other times one implies the other. In the case of Entree [16], it is noted that users might demonstrate a preference for *inexpensive* and *nice* restaurants. Since usually some trade-off is involved, a restaurant that does satisfy these criteria should be especially rewarded when it comes to recommendation. In the case of CB movie recommendation, it could be that a user likes *Johnny Depp* and *Tim Burton*. As there is a high frequency of films which are directed by Tim Burton that also star Johnny Depp, it might not make sense to *double-count* these features. The Choquet integral can account for a combination of these situations, since a weight is allocated to each subset of criteria. The subset of “stars Depp AND is directed by Burton” would be allocated less weight than the sum of its parts, while inexpensive and nice restaurants in the KB example would be allocated more.

Of course, sometimes the structure of the data might be difficult to understand and interpret towards the use of a particular function. In these cases, it might be worthwhile to check the accuracy of a number of functions on a subset of the data. A comparison of the minimum, maximum, arithmetic mean and harmonic mean could suggest much about which functions will be useful.

22.4.4 Weight and Parameter Determination

The determination of weights for use in ratings aggregation for CF is often understood in terms of the similarity between users and neighborhood formation. Weights in CB and UB are a measure of the importance of each feature to the user, while the weights in weighted HS are indicative of the reliability of each component in recommendation. Weights can be selected using predetermined measures like cosine, or might be decided in advance by the RS designers - e.g. we decide to weight the similar users with a decreasing weighting vector $\mathbf{w} = (0.4, 0.3, 0.2, 0.1)$. Some systems adjust weights incrementally according to implicit or explicit feedback concerning the quality of recommendation, for instance in the hybrid RS, P-Tango [19]. In Section 22.5, programming methods are discussed for determining weights from available data-sets.

22.5 Sophisticated Aggregation Procedures in Recommender Systems: Tailoring for Specific Applications

We consider the fitting problem in terms of a CF recommender, however it is also possible to fit weights in CB and UB recommender systems provided the system has access to input and output values so that the strength of fit can affirm the suitability

of the weights or parameters. Fitting can be accomplished by means of interpolation or approximation. In the case of interpolation, the aim is to fit the specified output values exactly (in the case of aggregation functions, the pairs $((0, 0, \dots, 0), 0)$ and $((1, 1, \dots, 1), 1)$ should always be interpolated). In the case of RS, the data will normally contain some errors or degree of approximation, and therefore it may not be appropriate to interpolate the inaccurate values. In this case our aim is to stay close to the desired outputs without actually matching them. This is the approximation problem.

The selection of an aggregation function can be stated formally as follows:

Let us have a number of mathematical properties P_1, P_2, \dots and the data $D = \{(\mathbf{x}_k, y_k)\}_{k=1}^K$. Choose an aggregation function f consistent with P_1, P_2, \dots , and satisfying $f(\mathbf{x}_k) \approx y_k, k = 1, \dots, K$.

We can also vary the problem to accommodate a fitting to intervals, i.e. we require $f(\mathbf{x}_k) \in [\underline{y}_k, \bar{y}_k]$. How these values are specified will depend on the application. In some cases it may be possible to fit the function exactly without violating any of the desired properties, however most of the time we merely want to minimize the error of approximation.

Mathematically, the satisfaction of approximate equalities $f(\mathbf{x}_k) \approx y_k$ can be translated into the following minimization problem.

$$\begin{aligned} & \text{minimize } \|\mathbf{r}\| & (22.10) \\ & \text{subject to } f \text{ satisfies } \mathcal{P}_1, \mathcal{P}_2, \dots, \end{aligned}$$

where $\|\mathbf{r}\|$ is the norm of the residuals, i.e., $\mathbf{r} \in R^K$ is the vector of the differences between the predicted and observed values $r_k = f(\mathbf{x}_k) - y_k$. There are many ways to choose the norm, and the most popular are the least squares norm

$$\|\mathbf{r}\|_2 = \left(\sum_{k=1}^K r_k^2 \right)^{1/2},$$

and the least absolute deviation norm

$$\|\mathbf{r}\|_1 = \sum_{k=1}^K |r_k|,$$

or their weighted analogues if some of the y_k are considered less reliable than others.

Consider Example 22.8.⁷

⁷ All examples in this section utilize the software packages *aotool* and *fntools* [8]

Example 22.8. In a CF recommending application we want to use five similar users to predict the ratings of new objects for a given user. At hand we have a data set of many items previously rated by the user and the five similar users or neighbors $\{(d_i, R(u, d_i))\}_{i=1}^{10}$ where $d_i = (R(u_1, d_i), \dots, R(u_5, d_i))$ denotes the ratings given by each of the neighbors u_1, \dots, u_5 to a past item d_i , and the $R(u, d_i)$ are the user's actual ratings. I.e. $d_i = \mathbf{x}_k, R(u, d_i) = y_k$ from above. Table 22.2 shows an example data set with two items rated by the neighbors which the user is yet to rate and could be recommended. We want to define a weighted arithmetic mean using the least squares approach that assigns a weight w_i to each user. So we have

$$\begin{aligned} &\text{minimize } \sum_{i=1}^1 0 \left(\sum_{j=1}^5 w_j R(u_j, d_i) - R(u, d_i) \right)^2 \\ &\text{subject to } \sum_{j=1}^5 w_j = 1, \\ &w_1, \dots, w_5 \geq 0. \end{aligned}$$

This is a quadratic programming problem, which is solved by a number of standard methods. In the current example one resulting model allocates the weights $\mathbf{w} = \langle 0.27, 0.07, 0.06, 0.19, 0.41 \rangle$ with recommendation scores of 4.7 and 7.9 for the unrated items. The maximum difference between observed and predicted ratings is 2.45 with an average of 0.98. If we had instead used the cosine calculation to define the weights, we would have $\mathbf{w} = \langle 0.19, 0.24, 0.23, 0.18, 0.17 \rangle$ and recommendation scores of 5.6 and 7.1. The accuracy is similar for this method, with maximum error 2.48 and average error 1.6. Interestingly u_5 was least similar using this measure, but most important when accurately predicting the ratings for u .

Table 22.2: Example dataset for mutually rated items in CF

	Items $i = 1..10$ rated by user and neighbors										Unrated	
User ratings $R(u, d_i)$	6	4	6	8	10	5	7	7	5	5	?	?
Neighbor ratings												
$R(u_1, d_i)$	4	4	4	8	10	3	7	5	3	3	4	7
$R(u_2, d_i)$	6	0	6	4	6	1	3	3	1	5	8	7
$R(u_3, d_i)$	3	1	8	5	7	2	4	4	2	2	7	5
$R(u_4, d_i)$	6	5	6	8	8	6	5	5	3	5	3	8
$R(u_5, d_i)$	6	4	6	7	8	1	5	8	5	8	5	9

As mentioned, if the number of items to be recommended is limited, the ranking, rather than the accuracy of prediction becomes crucial (see also [27]). In situations where it makes sense, the ranking of the outputs can be preserved with $f(R(u_1, d_k), \dots, R(u_n, d_k)) \leq f(R(u_1, d_l), \dots, R(u_n, d_l))$ if $R(u, d_k) \leq R(u, d_l)$ for all pairs k, l added as an extra constraint. In CF, imposing this condition weights the similar users higher who have rankings that better reflect the user's. This is useful when we know that some users might tend to overrate or underrate items, but will be consistent in terms of the items they prefer.

The approximation problem thus far described may turn out to be a general non-linear optimization problem, or a problem from a special class. Some optimization problems utilize a convex objective function or variant of this, in which case the difficulty is not so much in this step, but rather in defining the constraints. Fitting the Choquet integral, for instance has an exponential number of constraints which need to be defined. Many problems, however can be specified as linear or quadratic programming problems, which have been extensively studied with many solution techniques available. Example 22.9 uses the same dataset (Table 22.2) with the Choquet integral as the desired function. In practice, it is preferable to have a much larger data set for the Choquet integral given that it is defined at 2^n points (so ideally, the number of data for fitting should be well above this). This ensures that the resulting function is not too specialized.

Example 22.9. (Continued from Example 22.8)... The system designers decide that they would prefer to use a Choquet integral to predict the unknown ratings. To make the fitting process less susceptible to outliers, they decide to use the least absolute deviation norm and express the optimization process as the following:

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^5 |C_v(d_i) - R(u, d_i)| \\ &\text{subject to} && v(A) - v(B) \geq 0, \text{ for all } B \subseteq A, \\ &&& v(A) \geq 0, \forall A \subset \mathcal{N}, v(\emptyset) = 0, v(\mathcal{N}) = 1. \end{aligned}$$

This results in a Choquet integral defined by a fuzzy measure with the following values

$$\begin{aligned} v(\{1\}) &= 1, v(\{2\}) = 0.33, v(\{3\}) = 0, v(\{4\}) = v(\{5\}) = 0.67 \\ v(\{2, 3\}) &= 0.33, v(\{2, 4\}) = v(\{3, 4\}) = v(\{3, 5\}) = v(\{2, 3, 4\}) = 0.67 \\ v(A) &= 1 \text{ for all other subsets.} \end{aligned}$$

The Shapley values provide a good indication of the influence of each of the neighbors, and are given as

$$\phi_1 = 0.39, \phi_2 = 0.11, \phi_3 = 0, \phi_4 = 0.22, \phi_5 = 0.28$$

. As with the weighted arithmetic mean, the values suggest that neighbors 1, 4 and 5 are perhaps more similar to the given user. We also note the interaction indices for pairs, given as

$$I_{12} = I_{24} = I_{45} = -0.17, I_{14} = -0.33, I_{15} = -0.5$$

$$I_{ij} = 0 \text{ for all other pairs.}$$

This shows the redundancy between some of the neighbors. In particular, neighbors 1 and 5 are very similar. The maximum error in this case is 1.6 and the average error is 0.6, with resulting recommendations 6.0 and 8.7. Because of the substitutive variables, the function behaves similar to a maximum function. We see the high score given for the latter item, mainly due to the high ratings given by neighbors 4 and 5.

The families of aggregation functions defined in Section 22.3.2 are convenient to use when trying to understand and interpret the results. The weights and parameters have a tangible meaning and fitting these functions essentially involves finding the best values for each parameter to maximize the reliability of the RS.

In other situations however, the interpretation side of things may not be as important: we just want to predict the unknown ratings reliably and automatically. There are many non-parametric methods for building aggregation functions, which do not have the advantage of system interpretation, however can be constructed automatically and fit the data closely. One “black-box” type method is to build a general aggregation operator piecewise from the data. We can ensure that monotonicity and boundary conditions are specified by smoothing the data and ensuring these properties hold for each individual segment. We consider here, the construction of spline based aggregation functions [10].

Monotone tensor product splines are defined as

$$f_B(x_1, \dots, x_n) = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \dots \sum_{j_n=1}^{J_n} c_{j_1 j_2 \dots j_n} B_{j_1}(x_1) B_{j_2}(x_2) \dots B_{j_n}(x_n).$$

If it is desired the built function belong to a particular class or hold certain properties, additional constraints can be added when fitting. In particular, we can ensure monotonicity holds by expressing linear conditions on the coefficients $c_{j_1 j_2 \dots j_n}$. The fitting of this function to data involves sparse matrices, their size increasing with the number of basis functions in respect to each variable and exponentially with n . We give an example of this fitting process in the Example 22.10.

Example 22.10. (Continued from Examples 22.8-22.9). It is not necessary in our application that the weighting of similar users be known. We simply want automatically built functions that can predict the ratings of unseen items. We decide that we still desire the properties of monotonicity and idempotency to ensure reliable outputs, and build a general aggregation operator represented by tensor product splines. The following quadratic programming problem is used:

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^5 (f_B(d_i) - R(u, d_i))^2 \\ & \text{subject to} && \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \dots \sum_{j_n=1}^{J_n} c_{j_1 j_2 \dots j_n} \geq 0, \\ & && f_B(0, \dots, 0) = 0, f_B(1, \dots, 1) = 0. \end{aligned}$$

Idempotency is also ensured by imposing a number of interpolation conditions such that $f_B(t_i, \dots, t_i) = t_i$. These conditions must be chosen in a certain way (see [6, 7]). The fitted non-parametric function gives resulting recommendation scores for the unrated items of 4.2 and 8.1 so it seems that the latter item should be suggested to the user.

Clearly it is the choice of system designers of whether to use non-parametric or parametric methods, and how complex an aggregation function should be used. Recommender systems usually require timely decisions and deal with large data sets, so a compromise between expressibility and simplicity is usually sought.

22.6 Conclusions

The purpose of this chapter has been to present the state of the art in aggregation functions and introduce established families of these functions that have properties useful for the purposes of recommendation. This has included means defined with various weights, Choquet integrals defined with respect to fuzzy measures, t-norms/t-conorms which can be built from generators, and representable uninorms. Many of the current methods used in recommender systems involve constructing weighted arithmetic means where weights are determined by varying measures of similarity, however in many cases the accuracy and flexibility of functions could be improved with only slight increases to complexity. We have provided a number of illustrative examples of the different ways in which aggregation functions can be applied to recommendation processes including ratings aggregation, feature

combination, similarity and neighborhood formation and component combination in weighted hybrid systems. We also referred to some current software tools which can be used to fit these functions to data (see also [29, 25]) when we are trying to find weights, similarity or the parameters used that best model the dataset.

The research in aggregation functions is extensive with a number of important results, some of which have been explored with the application to recommender systems in mind. As only an introduction has been provided here, we recommend the recent books listed under *Further Reading* which provide details of many aggregation methods.

22.7 Further Reading

- Alsina, C., Frank, M.J. and Schweizer, B.: *Associative Functions: Triangular Norms And Copulas*. World Scientific, Singapore (2006)
- Beliakov, G., Pradera, A. and Calvo, T.: *Aggregation Functions: A guide for practitioners*. Springer, Heidelberg, Berlin, New York (2007)
- Calvo, T. and Mayor, G. and Mesiar, R.: *Aggregation Operators: New Trends and Applications*. Physica-Verlag, Heidelberg, New York (2002)
- Grabisch, M., Marichal, J.-L. Mesiar, R. and Pap, E.: *Aggregation Functions*. Cambridge University Press, *Encyclopedia of Mathematics and its Applications*, No 127, Cambridge (2009)
- Klement, E.P., Mesiar, R. and Pap, E.: *Triangular Norms*. Kluwer, Dordrecht, (2000)
- Torra, V. and Narukawa, Y.: *Modeling Decisions. Information Fusion and Aggregation Operators*. Springer, Berlin, Heidelberg (2007)

Acknowledgements

T. Calvo wishes to acknowledge her support from the projects MTM2006-08322 and PR2007-0193 from Ministerio de Educación y Ciencia, Spain and the EU project 143423-2008-LLP-ES-KA3-KA3MP.

References

1. Adomavicius, G., Sankaranarayanan, R., Sen, S., and Tuzhilin, A.: Incorporating contextual information in recommender systems using a multi-dimensional approach, *ACM Transactions on information systems*, **23**(1), 103–145 (2005)
2. Adomavicius, G. and Kwon, Y.: New Recommendation Techniques for Multicriteria Rating Systems. *IEEE Intelligent Systems*. **22**(3), 48–55 (2007)
3. Ahn, H.J.: A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*. **178**, 37–51 (2008)
4. Al-Shamri, M.Y.H. and Bharadwaj, K.K.: Fuzzy-genetic approach to recommender systems based on a novel hybrid user model. *Expert Systems with Applications*, **35**, 1386–1399 (2008)
5. Balabanovic, M. and Shoham, Y.: Fab: Content-Based, Collaborative Recommendation. *Comm. ACM*. **40**(3), 66–72 (1997)
6. Beliakov, G.: Monotone approximation of aggregation operators using least squares splines. *Int. J. of Uncertainty, Fuzziness and Knowledge-Based Systems*. **10**, 659–676 (2002)
7. Beliakov, G.: How to build aggregation operators from data? *Int. J. Intelligent Systems*. **18**, 903–923 (2003)
8. Beliakov, G.: FMTools package, version 1.0, <http://www.deakin.edu.au/~gleb/aotool.html>, (2007)
9. Beliakov, G. and Calvo, T.: Construction of Aggregation Operators With Noble Reinforcement. *IEEE Transactions on Fuzzy Systems*. **15**(6), 1209–1218 (2007)
10. Beliakov, G., Pradera, A. and Calvo, T.: *Aggregation Functions: A guide for practitioners*. Springer, Heidelberg, Berlin, New York (2007)
11. Beliakov, G., Calvo, T. and James, S.: On Lipschitz properties of generated aggregation functions. *Fuzzy Sets and Systems*, 161, 1437–1447 (2010)
12. Beliakov, G. and James, S.: Using Choquet Integrals for kNN Approximation and Classification. In Gary G. Feng (ed.), 2008 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2008), 1311–1317 (2008)
13. Bolton, J., Gader, P. and Wilson, J.N.: Discrete Choquet Integral as a Distance Metric. *IEEE Trans. on Fuzzy Systems*, **16**(4), 1107–1110 (2008)
14. Buchanan, B. and Shortliffe, E.: *Rule-based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA (1984)
15. Bullen, P.S.: *Handbook of Means and Their Inequalities*. Kluwer, Dordrecht (2003)
16. Burke, R.: Hybrid Recommender Systems: Survey and Experiments, User Modeling and User-adapted interaction, **12**(4), 331–370 (2002)
17. Calvo, T., Kolesárová, A., Komorníková, M. and Mesiar, R.: Aggregation operators: properties, classes and construction methods. In : Calvo, T., Mayor, G. and Mesiar, R. (eds.) *Aggregation Operators. New Trends and Applications*, pp. 3–104. Physica-Verlag, Heidelberg, New York (2002)
18. Chen, Y.-L. and Cheng, L.-C.: A novel collaborative filtering approach for recommending ranked items. *Expert Systems with Applications*. **34**, 2396–2405 (2008)
19. Claypool, M., Gokhale, A., Miranda, T., Mumikov, P., Netes, D. and Sartin, M.: Combining Content-Based and Collaborative Filters in an Online Newspaper. In *Proceedings of SIGIR 99 Workshop on Recommender Systems: Algorithms and Evaluation*, Berkeley, CA (1999)
20. Campos, L.M.d., Fernández-Luna, J.M. and Huete, J.F.: A collaborative recommender system based on probabilistic inference from fuzzy observations. *Fuzzy Sets and Systems*. **159**, 1554–1576 (2008)
21. Dubois, D. and Prade, H.: *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York (1980)
22. Dubois, D. and Prade, H.: *Fundamentals of Fuzzy Sets*. Kluwer, Boston (2000)
23. Dubois, D., Hllermeier, E., and Prade, H.: Fuzzy methods for case-based recommendation and decision support. *J. Intell. Inform. Systems*. **27**(2), 95–115 (2006)

24. Duda, R. Hart, P. and Nilsson, N.: Subjective Bayesian methods for rule-based inference systems. In Proc. Nat. Comput. Conf. (AFIPS), volume 45, 1075–1082 (1976)
25. Grabisch, M., Kojadinovic, I., and Meyer, P.: A review of methods for capacity identification in Choquet integral based multi-attribute utility theory: Applications of the Kappalab R package. *European Journal of Operational Research*. **186**, 766–785 (2008)
26. Hibbert, R.: What is Indie Rock? *Popular Music and Society*. **28**(1), 55-77 (2005)
27. Kaymak, U. and van Nauta Lemke, H.R.: Selecting an aggregation operator for fuzzy decision making. In 3rd IEEE Intl. Conf. on Fuzzy Systems, volume 2, 1418–1422 (1994)
28. Klement, E.P., Mesiar, R. and Pap, E.: *Triangular Norms*. Kluwer, Dordrecht, (2000)
29. Kojadinovic, I. and Grabisch, M.: Non additive measure and integral manipulation functions, R package version 0.2, <http://www.polytech.univ-nantes.fr/kappalab>, (2005)
30. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., and Riedl, J.: GroupLens: applying collaborative filtering to Usenet news, *Communications of the ACM*, **40**(3), 77–87 (1997)
31. Krawczyk, H., Knopa, R., Lipczynska, K., and Lipczynski, M.: Web-based Endoscopy Recommender System - ERS. In International Conference on Parallel Computing in Electrical Engineering (PARELEC'00), Quebec, Canada, 257–261 (2000)
32. Linden, G., Smith, B., and York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, **7**(1), 76–80 (2003)
33. Mayor, G., and Calvo, T.: Extended aggregation functions. In IFSA'97, volume 1, Prague, 281–285 (1997)
34. Miller, B.N., Albert, I., Lam, S.K., Konstan, J.A., and Riedl, J.: MovieLens unplugged: experiences with an occasionally connected recommender system. In Proceedings of the 8th international ACM conference on Intelligent user interfaces, Miami, USA, 263–266 (2003)
35. Riedl, J. and Dourish, P.: Introduction to the special section on recommender systems, *ACM Transactions on Computer-Human Interaction*, **12**(9), 371–373 (2005)
36. Resnick, P., Iakovou, N., Sushak, M., Bergstrom, P., and Riedl, J.: GroupLens: An open architecture for collaborative filtering of Netnews. In Proceedings of ACM conference on computer supported cooperative work, Chapel Hill, NC, 175–186 (1994)
37. Rokach, L., Maimon, O., and Arbel, R.: Selective voting-getting more for less in sensor fusion, *International Journal of Pattern Recognition and Artificial Intelligence* **20** (3), pp. 329–350 (2006)
38. Rokach, L. and Maimon, O., Theory and applications of attribute decomposition, *IEEE International Conference on Data Mining*, IEEE Computer Society Press, pp. 473–480 (2001).
39. Salton, G. and McGill, M.: *Introduction to Modern Information retrieval*. McGraw Hill, New York (1983)
40. Schafer, J.B., Konstan, J.A., and Riedl, J.: E-commerce recommendation applications, *Data Mining and Knowledge Discover*, **5**, 115–153 (2001)
41. Santini, S. and Jain, R.: Similarity Measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **21**(9), 871–883 (1999)
42. Ujjin, S., and Bentley, P.: Using evolutionary to learn user preferences. In: Tan, K., Lim, M., Yao, X. and Wang, L. (eds.). *Recent advances in simulated evolution and learning*, pp. 20-40. World Scientific Publishing (2004)
43. Yager, R.: On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Trans. on Systems, Man and Cybernetics*. **18**, 183–190 (1988)
44. Yager, R.: Noble Reinforcement in Disjunctive Aggregation Operators. *IEEE Transactions on Fuzzy Systems*. **11**(6) 754–767 (2003)
45. Yager, R. R. and Filev, D. P.: Induced ordered weighted averaging operators. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, **20**(2), 141–150 (1999)
46. Yager, R. and Rybalov. A.: Uninorm aggregation operators. *Fuzzy Sets and Systems*. **80**, 111–120 (1996)

Chapter 23

Active Learning in Recommender Systems

Neil Rubens, Dain Kaplan, and Masashi Sugiyama

23.1 Introduction

Recommender Systems (RSs) are often assumed to present items to users for one reason – to *recommend* items a user will likely be interested in. Of course RSs *do* recommend, but this assumption is biased, with no help of the title, towards the “recommending” the system will do. There is another reason for presenting an item to the user: to learn more about his/her preferences, or his/her *likes* and *dislikes*. This is where Active Learning (AL) comes in. Augmenting RSs with AL helps the user become more self-aware of their own likes/dislikes while at the same time providing new information to the system that it can analyze for subsequent recommendations. In essence, applying AL to RSs allows for personalization of the recommending process, a concept that makes sense as recommending is inherently geared towards personalization. This is accomplished by letting the system actively influence which items the user is exposed to (e.g. the items displayed to the user during sign-up or during regular use), and letting the user explore his/her interests freely.

Unfortunately, there are very few opportunities for the system to acquire information, such as when a user rates/reviews an item, or through a user’s browsing history.¹ Since these opportunities are few, we want to be as sure as possible that the data we acquire tells us something *important* about the user’s preferences. After all, one of the most valuable assets of a company is user data.

Neil Rubens
University of Electro-Communications, Tokyo, Japan, e-mail: rubens@hrstc.org

Dain Kaplan
Tokyo Institute of Technology, Tokyo, Japan e-mail: dain@cl.cs.titech.ac.jp

Masashi Sugiyama
Tokyo Institute of Technology, Tokyo, Japan e-mail: sugi@cs.titech.ac.jp

¹ There is an increasing trend to utilize social networks for acquiring additional data (see Chapters 18 and 19).

For example, when a new user starts using a recommender system, very little is known about his/her preferences [44, 36, 2]. A common approach to learning the user's preferences is to ask him/her to rate a number of items (known as training points). A model that approximates the user's preferences is then constructed from this data. Since the number of items reviewed by the user cannot span the system's entire catalog (and indeed would make the task of AL as well as *recommending* moot points), the collection of items presented to the user for review must necessarily be very limited. The accuracy of the learned model thus greatly depends on the selection of good training points. A system might ask the user to rate *Star Wars I*, *II*, and *III*. By rating all three volumes of this trilogy, we will have a good idea of the user's preferences for *Star Wars*, and possibly by extension, an inclination for other movies within the Sci-Fi genre, but overall the collected knowledge will be limited. It is therefore unlikely that picking the three volumes of a trilogy will be informative.² Another issue with selecting a popular item such as *Star Wars* is that by definition the majority of people like them (or they would not be popular). It is not surprising then, that often little insight is gained by selecting popular items to learn about the user (unless the user's tastes are atypical).

There is sometimes a notion that AL is a bothersome, intrusive process, but it does not have to be this way [54, 38]. If the items presented to the user are interesting, it could be both a process of discovery and of exploration. Some Recommender Systems provide a "surprise me!" button to motivate the user into this explorative process, and indeed there are users who browse suggestions just to see what there is without any intention of buying. Exploration is crucial for users to become more self-aware of their own preferences (changing or not) and at the same time inform the system of what they are. Keep in mind that in a sense users can also be defined by the items they consume, *not* only by the ratings of their items, so by prompting users to rate different items it may be possible to further distinguish their preferences from one another and enable the system to provide better personalization and to better suit their needs.

This chapter is only a brief foray into Active Learning in Recommender Systems.³ We hope that this chapter can, however, provide the necessary foundations.

For further reading, [46] gives a good, general overview of AL in the context of Machine Learning (with a focus on Natural Language Processing and Bioinformatics). For a theoretical perspective related to AL (a major focus in the field of Experimental Design), see [7, 4, 22]; there have also been recent works in Computer Science [16, 5, 51].

² Unless our goal is to learn a kind of micro-preference, which we can define as a person's tendency to be more 'picky' concerning alternatives close to one another in a genre they like.

³ Supplementary materials on Active Learning can be found at: <http://www.DataMilking.org/AL>

23.1.1 Objectives of Active Learning in Recommender Systems

Different RSs have different objectives (see Chapter 8), which necessitate different objectives for their Active Learning components as well. As a result, one AL method may be better suited than another for satisfying a given task [35]. For example, what is important in the recommender system being built (see Chapter 10)? The difficulty of signing-up (user effort)? If the user is happy with the service (user satisfaction)? How well the system can predict a user's preferences (accuracy)? How well the system can express a user's preferences (user utility)? How well the system can serve other users by what it learns from this one (system utility)? System functionality may also be important, such as when a user inquires about a rating for an item of interest the system has insufficient data to predict a rating for, what the system does in response. Does it in such a case give an ambiguous answer, allowing the user to train the system further if they have the interest and the time to do so? Or does it require them to rate several other items before providing a prediction? Perhaps the user has experienced the item (e.g. watched the movie or trailer) and thinks their rating differs substantially from the predicted one [10]. In all these cases how the system responds to the user is important for consideration.

Traditionally AL does not consider the trade-off of exploration (learning user's preferences) and exploitation (utilizing user's preferences), that is, it does not dynamically assign weights to exploitation/exploration depending on system objectives. This trade-off is important because for a new user about which nothing or little is known, it may be beneficial to validate the worth of the system by providing predictions the user is likely to be interested in (exploitation), while long-term users may wish to expand their interests through exploration [38, 41].

Though an objective of the RS will likely be to provide accurate predictions to the user, the system may also need to recommend items of high novelty/serendipity, improve coverage, maximize profitability, or determine if the user is even able to evaluate a given item, to name a few [44, 21, 33]. Multiple objectives may need to be considered simultaneously (see Chapter 24), e.g. minimizing the net acquisition cost of training data *while* maximizing net profit, or finding the best match between the cost of offering an item to the user, the utility associated with expected output, and the alternative utility of inaction [38]. The utility of training may also be important, e.g. predicting ratings for exotic cars may not be so useful if the user is not capable of purchasing them and so should be avoided. It can be seen that the system objective is often much more complex than mere predictive accuracy, and may include the combination of several objectives.

While Recommender Systems in general often have an ill-defined or open-ended objective, namely to predict items a user would be interested in, Conversation-based AL [32, 37, 9], as the name suggests, engages in a conversation with the user as a goal oriented approach. It seeks to, through each iteration of questioning, elicit a response from the user to best reduce the search space for quickly finding what it is the user seeks (Section 23.8).

The New User Problem When a user starts using a RS they expect to see interesting results after a minimal amount of training. Though the system knows little about the user's preferences, it is essential that training points are selected to be rated by the user that will maximize the understanding of what the new user wants [35].

The New Product Problem As new products are introduced into the system, it is important to quickly improve prediction accuracy for these items by selecting users to rate them [24].

Cost of obtaining an output value Different means of obtaining an output value come at different costs. Implicit strategies, such as treating a user click on a suggested item as positive output, or not clicking as negative, are inexpensive in relation to user effort. Conversely, asking the user to explicitly rate an item is more costly, though still dependent on the task. Watching a movie like Star Wars to rate may provide good results but requires substantial user effort [20]; rating a joke requires much less. This often dovetails the exploration/exploitation coupling and trade-offs between obtaining outputs from different inputs should also be considered (e.g. certainty/uncertainty, ease of evaluation, etc.)

Adaptation for different AL methods Though we focus on the traditional objective of reducing predictive error, it is equally plausible to construct a method for maximizing other goals, such as profitability. In this case a model would pick points that most likely increase profit rather than a rating's accuracy.

23.1.2 An Illustrative Example

Let us look at a concrete example of Active Learning in a Recommender System. This is only meant to demonstrate concepts, so it is oversimplified. Please note that the similarity metric may differ depending on the method used; here, movies are assumed to be close to one another if they belong to the same genre. Figure 23.1 shows two charts, the leftmost is our starting state, in which we have already asked the user to rate a movie within the upper right group, which we will say is the Sci-Fi genre. The right chart shows us four possibilities for selecting our next training point: (a), (b), (c), or (d). If we select the training point (a) which is an obscure movie (like *The Goldfish Hunter*), it does not affect our predictions because no other movies (points) are nearby. If we select the training point (b), we can predict the values for the points in the same area, but these predictions are already possible from the training point in the same area (refer to the chart on the left). If training

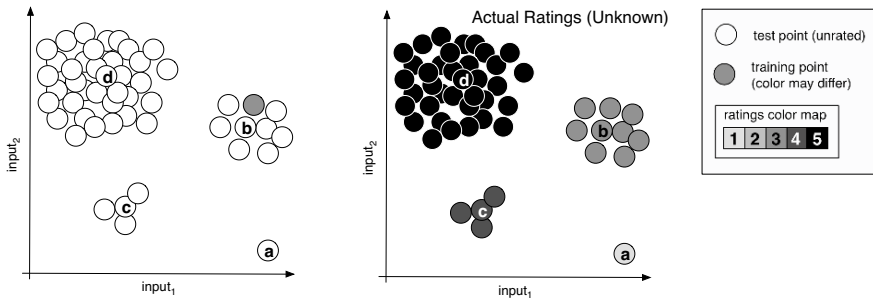


Fig. 23.1: Active Learning: illustrative example (See Section 23.1.2).

point (c) is selected, we are able to make new predictions, but only for the other three points in this area, which happens to be Zombie movies. By selecting training point (d), we are able to make predictions for a large number of test points that are in the same area, which belong to Comedy movies. Thus, selecting (d) is the ideal choice because it allows us to improve accuracy of predictions the most (for the highest number of training points).⁴

23.1.3 Types of Active Learning

AL methods presented in this chapter have been categorized based on our interpretation of their primary motivation/goal. It is important to note, however, that various ways of classification may exist for a given method, e.g. sampling close to a decision boundary may be considered as Output Uncertainty-based since the outputs are unknown, Parameter-based because the point will alter the model, or even Decision boundary-based because the boundary lines will shift as a result. However, since the sampling is performed with regard to decision boundaries, we would consider this the primary motivation of this method and classify it as such.

In addition to our categorization by primary motivation (Section 23.1), we further subclassify a method’s algorithms into two commonly classified types for easier comprehension: instance-based and model-based.

Instance-based Methods A method of this type selects points based on their properties in an attempt to predict the user’s ratings by finding the closest match to other users in the system, without explicit knowledge of the underlying model. Other common names for this type include memory-based, lazy learning, case-based, and non-parametric [2]. We assume that any existing data is accessible, as well as rating predictions from the underlying model.

⁴ This may be dependent on the specific prediction method used in the RS.

Model-based Methods A method of this type selects points in an attempt to best construct a model that explains data supplied by the user to predict user ratings [2]. These points are also selected to maximize the reduction of expected error of the model. We assume that in addition to any data available to instance-based methods, the model and its parameters are also available.

Modes of Active Learning: Batch and Sequential Because users typically want to see the system output something interesting immediately, a common approach is to recompute a user's predicted ratings after they have rated a single item, in a sequential manner. It is also possible, however, to allow a user to rate several items, or several features of an item before readjusting the model. On the other hand, selecting training points sequentially has the advantage of allowing the system to react to the data provided by users and make necessary adjustments immediately. Though this comes at the cost of interaction with the user at each step. Thus a trade-off exists between Batch and Sequential AL: the usefulness of the data vs. the number of interactions with the user.

23.2 Properties of Data Points

When considering any Active Learning method, the following three factors should always be considered in order to maximize the effectiveness of a given point. Supplementary explanations are then given below for the first two. Examples refer to the Illustrative Example (Figure 23.1).

- (R1) *Represented*: Is it already represented by the existing training set? E.g. point (b).
- (R2) *Representative*: Is the point a good candidate for representing other data points? Or is it an outlier? E.g. point (a).
- (R3) *Results*: Will selecting this point result in better prediction ratings or accomplish another objective? E.g. point (d), or even point (c).

(R1) Represented by the Training Data As explained in the introduction to this chapter, asking for ratings of multiple volumes from a trilogy, such as *Star Wars*, is not likely beneficial, as it may not substantially contribute to the acquisition of *new* information about the user's preferences. To avoid obtaining redundant information, therefore, an active learning method should favor items that are not yet well represented by the training set [18].

(R2) Representative of the Test Data It is important that any item selected for being rated by an AL algorithm be as representative of the test items as possible (we consider all items as potentially belonging to the test set), since the accuracy of the

algorithm will be evaluated based on these items. If a movie is selected from a small genre, like Zombie movies from the Illustrative Example (Figure 23.1), then obtaining a rating for this movie likely provides little insight into a user's preferences for other, more prominent genres. In addition, users naturally tend to rate movies from genres they like, meaning that any genre that dominates the training set (which is likely composed of items the user likes) may be representative of only a small portion of all items [38]. In order to increase information obtained, it is important to select representative items which may provide information about the other yet unrated items [18, 47, 53].

23.2.1 Other Considerations

In addition to the three Rs listed in Section 23.2, it may also be desirable to consider other criteria for data points, such as the following.

Cost As touched upon in the introduction to this chapter, obtaining implicit feedback from user selections is cheaper than asking the user to explicitly rate an item [19]. This can be considered a variable cost problem. One approach for tackling this, is to take into account both the cost of labeling an item and the future cost of estimated misclassification were the item to be added to the training set [27]. Moreover, the cost may be unknown beforehand [48].

Ratability A user may not always be able to provide a rating for an item; you cannot properly rate a movie you have not seen! It is suggested therefore that the probability of a user being able to evaluate an item also be considered [20].

Saliency Decision-centric AL places emphasis on items whose ratings are more likely to affect decision-making, and acquires instances that are related to decisions for which a relatively small change in their estimation can change the order of top rated predictions [43]. For example, unless labeling an item would result in displacing or rearranging a list of the top ten recommended movies on a user's home page (the salient items), it may be considered of little use. It is also possible to only consider the effect of obtaining an item's rating on items that are strongly recommended by the system [6].

Popularity It has also been suggested to take into account an item's popularity [35], i.e. how many people have rated an item. This operates on the principle that since a popular item is rated by many people, it may be rather informative. Conversely, an item's rating uncertainty should also be considered since positive items have a tendency to be rated highly by most users, indicating the item may not provide much discriminative power and thus not worth including in the training set.

Best/Worst It has been shown [29] that looking at the best/worst reviews is beneficial when a user makes a decision about an item. Extending this idea to Active Learning we hypothesize with the "best/worst" principle that in order to make a de-

cision about a user's preferences it may also be beneficial to obtain his/her best/worst ratings (as it may capture user preferences well). By asking a user to provide his/her most liked/disliked items, it changes the problem of AL to one in which a user is asked to provide a rating for an item in a known class (e.g. to select a favorite movie from within a liked genre), and the process of obtaining an item is what incurs the cost [30]. This process is called *active class selection*. This is opposite from traditional AL techniques in which the labeling process (and not the items themselves) is what is assumed to incur a cost.

23.3 Active Learning in Recommender Systems

With Traditional AL, users are asked to rate a set of preselected items. This is often at the time of enrollment, though a preselected list may be presented to existing users at a later date as well. It may be argued that since these items are selected by experts, they capture essential properties for determining a user's preferences. Conceptually this may sound promising, but in practice this often leads towards selecting items that best predict the preferences of only an *average* user. Since the idea of RS is to provide personalized recommendations, selecting items to rate in a personalized manner should readily make more sense.

23.3.1 Method Summary Matrix

The following matrix (Table 23.1) provides a summary of the methods overviewed in this chapter. Explicit performance numbers are not supplied because to our knowledge no such comprehensive comparison in fact exists. AL methods *could* be compared on an individual basis, but any results would be inconclusive. This is because authors have a tendency to fix the predictive method and *then* apply one or more compatible AL methods to compare performance. Moreover, AL methods are often designed for a specific predictive method, and may therefore not have good performance when applied to a different method (which creates potentially misleading results), or may not even be applicable if the underlying system is not able to provide the required information, e.g. distribution of rating estimates. For these reasons we have opted to omit performances figures of any kind.

23.4 Active Learning Formulation

Passive Learning (see Figure 23.2) refers to when training data is provided beforehand, or when the system makes no effort to acquire new data (it simply accumulates through user activities over time). *Active Learning*, on the other hand, selects train-

Primary Motivation of Approach	Description/Goal	Possible Considerations
<i>Uncertainty Reduction</i> (Section 23.5)	Reducing uncertainty of: <ul style="list-style-type: none"> rating estimates (Section 23.5.1), decision boundaries (Section 23.5.2), model parameters (Section 23.5.3). 	Reducing uncertainty may not always improve accuracy; the model could simply be certain about the wrong thing (e.g. when the predictive method is wrong).
<i>Error Reduction</i> (Section 23.6)	Reducing the predictive error by utilizing the relation between the error and: <ul style="list-style-type: none"> the changes in the output estimates (Section 23.6.1.1), the test set error (Section 23.6.1.2), changes in parameter estimates (Section 23.6.2.1), the variance of the parameter estimates (Section 23.6.2.2). 	Estimating reduction of error reliably could be difficult and computationally expensive.
<i>Ensemble-based</i> (Section 23.7)	Identifying useful training points based on consensus between: <ul style="list-style-type: none"> models in the ensemble (Section 23.7.1), multiple candidate models (Section 23.7.1). 	The effectiveness depends on the quality of models/candidates, and could be computationally expensive since it is performed with regards to multiple models/candidates.

Table 23.1: Method Summary Matrix.

ing points actively (the input) so as to observe the most informative output (user ratings, behavior, etc.).

Let us define the problem of active learning in a more formal manner. An item is considered to be a multi-dimensional input variable and is denoted by a vector \mathbf{x} (also referred to as a *data point*).⁵ The set of all items is denoted by \mathcal{X} . The preferences of a user u are denoted by a function f_u (also referred to as a *target function*); for brevity, we use f when referring to a target user. A rating of an item \mathbf{x}

⁵ The way in which an item is represented depends on the RS and the underlying predictive method. In Collaborative Filtering based approaches items could be represented through the ratings of the users, or, in content based RSs, items could be represented through their descriptions.

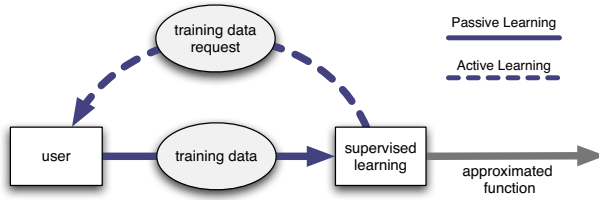


Fig. 23.2: Active learning employs an interactive/iterative process for obtaining training data, unlike passive learning, where the data is simply given.

is considered to be an output value (or *label*) and is denoted as $y = f(\mathbf{x})$. Each item \mathbf{x} could be rated on a finite scale $\mathcal{Y} = \{1, 2, \dots, 5\}$.

In supervised learning, the items and corresponding user ratings are often partitioned into complementary subsets – a training set and a testing set (also called a validation set). The task of supervised learning is then to, given a training set (often supplemented by the ratings of all users), learn a function \hat{f} that accurately approximates a user’s preferences. Items that belong to the training set are denoted by $\mathcal{X}^{(Train)}$, and these items along with their corresponding ratings constitute a training set, i.e. $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{\mathbf{x}_i \in \mathcal{X}^{(Train)}}$. We measure how accurately the learned function predicts the true preferences of a user by the generalization error:

$$G(\hat{f}) = \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(f(\mathbf{x}), \hat{f}(\mathbf{x})) P(\mathbf{x}). \tag{23.1}$$

In practice, however, $f(\mathbf{x})$ is not available for all $\mathbf{x} \in \mathcal{X}$; it is therefore common to approximate the generalization error by the test error:

$$\hat{G}(\hat{f}) = \sum_{\mathbf{x} \in \mathcal{X}^{(Test)}} \mathcal{L}(f(\mathbf{x}), \hat{f}(\mathbf{x})) P(\mathbf{x}), \tag{23.2}$$

where $\mathcal{X}^{(Test)}$ refers to the items in the *test set*, and prediction errors are measured by utilizing a loss function \mathcal{L} , e.g. mean absolute error (MAE):

$$\mathcal{L}_{MAE}(f(\mathbf{x}), \hat{f}(\mathbf{x})) = |f(\mathbf{x}) - \hat{f}(\mathbf{x})|, \tag{23.3}$$

or mean squared error (MSE):

$$\mathcal{L}_{MSE}(f(\mathbf{x}), \hat{f}(\mathbf{x})) = (f(\mathbf{x}) - \hat{f}(\mathbf{x}))^2. \tag{23.4}$$

The active learning criterion is defined so as to estimate the usefulness of obtaining a rating of an item \mathbf{x} and adding it to the training set $\mathcal{X}^{(Train)}$ for achieving a certain objective (Section 23.1.1). For simplicity, let us consider this objective to be the minimization of generalization error of a learned function with respect to the training set. We then denote the active learning criterion as:

$$\widehat{G}(\mathcal{X}^{(Train)} \cup \{\mathbf{x}\}), \quad (23.5)$$

or for brevity, denote it as:

$$\widehat{G}(\mathbf{x}). \quad (23.6)$$

The goal of active learning is to select an item \mathbf{x} that would allow us to minimize the generalization error $\widehat{G}(\mathbf{x})$:

$$\operatorname{argmin}_{\mathbf{x}} \widehat{G}(\mathbf{x}). \quad (23.7)$$

If we consider asking a user to rate an item \mathbf{x}_j or an item \mathbf{x}_k , then we would estimate their usefulness by an active learning criterion, i.e. $\widehat{G}(\mathbf{x}_j)$ and $\widehat{G}(\mathbf{x}_k)$, and select the one that will result in a smaller generalization error. Note that we need to estimate the usefulness of rating an item without knowing its actual rating. To distinguish a candidate item to be rated from the other items we refer to it as \mathbf{x}_a . AL can be applied to any predictive method as long as it provides the required information, such as rating estimates [42] and their distribution [23, 25], closeness to the decision boundary [55, 15], method parameters [49], etc.

\mathbf{x}	input (item)
\mathcal{X}	inputs (items)
y	output (item's rating)
$\mathcal{Y} = \{1, 2, \dots, 5\}$	possible outputs (ratings), i.e. $y \in \mathcal{Y}$
f	user's preferences function (unknown to the system)
$\mathcal{X}^{(Train)}$	training inputs (rated items)
$\mathcal{T} = \{(\mathbf{x}_i, y_i)_{\mathbf{x}_i \in \mathcal{X}^{(Train)}}\}$	training set (items and their ratings)
\widehat{f}	approximated function of user's preferences (from training set)
G	generalization error (predictive accuracy); see (23.1)
\mathbf{x}_a	item considered for rating
$\widehat{G}(\mathbf{x}_a)$	active learning criterion (estimates usefulness of rating an item \mathbf{x}_a)

Fig. 23.3: Summary of Notation.

Regression and Classification The problem of predicting a user's ratings could be treated as both a regression and a classification problem. It is a regression problem since the ratings are discrete numerical values, such as if we consider their ordinal properties, meaning the ratings could be ordered (e.g. a rating of 4 is higher than a rating of 3). On the other hand, we can disregard the numerical properties of the ratings and treat the problem as a classification one by treating ratings as classes/labels.⁶ For example, we can use a nearest-neighbor (NN) approach to do classification, e.g. pick the most frequent label of the neighbors; or we can use NN to do regression, e.g. calculate the mean of the ratings of the neighbors. Throughout

⁶ If the ordinal properties of the labels are considered, it is referred to as Ordinal Classification.

the chapter we use both classification and regression in examples, selecting the one most appropriate for aiding the current explanation.

23.5 Uncertainty-based Active Learning

Uncertainty-based AL tries to obtain training points so as to reduce uncertainty in some aspect, such as concerning output values [28], the model’s parameters [23], a decision boundary [45], etc. A possible drawback to this approach is that reducing uncertainty may not always be effective. If a system becomes certain about user ratings, it does not necessarily mean that it will be accurate, since it could simply be certain about the wrong thing (i.e., if the algorithm is wrong, reducing uncertainty will not help). As an example, if the user has so far rated items positively, a system may mistakenly be certain that a user likes all of the items, which is likely incorrect.

23.5.1 Output Uncertainty

In Output Uncertainty-based methods, an item to label (training point) is selected so as to reduce the uncertainty of rating predictions for test items. In Figure 23.1, with the assumption that the RS estimates the rating of an item based on the cluster to which it belongs (e.g. items in the same movie genre receive the same rating), if a user’s rating for a movie from the Sci-Fi genre (upper-right) has already been obtained, then there is a higher likelihood that the RS may be more certain about the ratings of other movies in the Sci-Fi genre, likely making it more beneficial to obtain a user’s preference for a movie from a genre (cluster) not yet sampled, i.e. a cluster that is still uncertain.

The difference between instance-based and model-based approaches for Output Uncertainty-based AL is primarily in how, for an arbitrary item \mathbf{x} , the rating’s distribution $P(Y_{\mathbf{x}})$ is obtained, where a rating’s distribution is defined as the probability of an item being assigned a certain rating. For model-based methods it is possible to obtain the rating’s distribution from the model itself. Probabilistic models are particularly well suited for this as they directly provide the rating’s distribution [23, 25]. For instance-based methods, collected data is used to obtain the rating’s distribution. As an example, methods utilizing nearest-neighbor techniques can obtain a rating’s distribution based on the votes of its neighbors, where “neighbor” here means a user with similar preferences,⁷ using a formula such as:

$$P(Y_{\mathbf{x}} = y) = \frac{\sum_{nn \in NN_{\mathbf{x},y}} w_{nn}}{\sum_{nn \in NN_{\mathbf{x}}} w_{nn}}, \quad (23.8)$$

⁷ Defining a neighbor as a similar item is also feasible depending on the method.

where NN_x are neighbors that have rated an item x , and $NN_{x,y}$ are neighbors that have given an item x a rating of y , and w_{nn} is the weight of the neighbor (such as similarity).

23.5.1.1 Active Learning Methods

Some AL methods [28] estimate the usefulness of a potential training point in a *local* (greedy) manner by measuring the uncertainty of its output value:

$$\widehat{G}_{Uncertainty_{local}}(\mathbf{x}_a) = -Uncertainty(Y_a). \quad (23.9)$$

Since our goal is to minimize \widehat{G} , rating an item with *high* uncertainty is useful; it will eliminate the uncertainty about the rating of the chosen item. However, labeling an item whose rating is uncertain does not necessarily accomplish the goal of reducing the uncertainty of ratings for other items (e.g. labeling an outlier may only reduce rating uncertainty for a few other similar items, such as when selecting item (c) in the Zombie genre, or even none as in (d), shown in Figure 23.1.

We may thus consider reducing uncertainty in a *global* manner by selecting an item which may reduce the uncertainty about *other* unrated items. One approach [40] for doing this is to define criteria by measuring the uncertainty of ratings over all of the test items $\mathcal{X}^{(Test)}$ with respect to a potential training input item \mathbf{x}_a :

$$\widehat{G}_{Uncertainty}(\mathbf{x}_a) = \frac{1}{|\mathcal{X}^{(Test)}|} \sum_{\mathbf{x} \in \mathcal{X}^{(Test)}} \mathbb{E}_{\mathcal{T}^{(a)}} (Uncertainty(Y_{\mathbf{x}})), \quad (23.10)$$

where $\frac{1}{|\mathcal{X}^{(Test)}|}$ is a normalizing factor, and $\mathbb{E}_{\mathcal{T}^{(a)}} (Uncertainty(Y_{\mathbf{x}}))$ is the expected value of uncertainty with respect to adding an estimated rating y_a of a candidate item \mathbf{x}_a to the training set \mathcal{T} ; i.e. $\mathcal{T}^{(a)} = \mathcal{T} \cup (\mathbf{x}_a, y_a)$.

A possible drawback of this non-local approach is that while with the local approach it is only necessary to estimate the uncertainty of a single output value y_a , for the non-local approach uncertainty needs to be estimated for the output values of *all* the test points *with respect to* a potential training point (\mathbf{x}_a, y_a) ; this may be difficult to estimate accurately and could be computationally expensive.

23.5.1.2 Uncertainty Measurement

Uncertainty of an item's rating (output value) is often measured by its variance, its entropy [28], or by its confidence interval [38]. Variance is maximized when ratings deviate the most from the mean rating, and entropy when all the ratings are equally likely.

Uncertainty of an output value could be calculated by using a definition of variance as follows:

$$Uncertainty(Y_a) = VAR(Y_a) = \sum_{y \in \mathcal{Y}} (y - \bar{Y}_a)^2 P(Y_a = y), \quad (23.11)$$

where \bar{Y}_a is the mean rating of all users for an item \mathbf{x}_a and $P(Y_a = y)$ is the probability of an item's rating Y_a being equal to y , both being calculated based on either nearest-neighbors for instance-based, or obtained from the model for model-based approaches.

Uncertainty could also be measured by entropy as follows:

$$Uncertainty(Y_a) = ENT(Y_a) = - \sum_{y \in \mathcal{Y}} P(Y_a = y) \log P(Y_a = y). \quad (23.12)$$

In [47] a method is proposed for measuring the uncertainty of a rating based on the probability of the most likely rating:

$$Uncertainty(Y_a) = -P(Y_a = y^*), \quad (23.13)$$

where $y^* = \operatorname{argmax}_y P(Y_a = y)$ is the most likely rating.

In [38] the confidence interval is used as a measure of uncertainty for selecting the training input point:

$$c = P(b_l(Y_a) < y_a < b_u(Y_a)), \quad (23.14)$$

where c is the confidence that the actual rating y_a will lie in the interval between the lower bound $b_l(Y_a)$ and the upper bound $b_u(Y_a)$. For example, it is possible for the system to be certain that an item will be assigned a rating between 3 and 5 with a probability $c = 90\%$. Many methods prefer items with a higher upper bound, indicating that an item may be rated highly (good for exploitation), and if the confidence interval is also wide then it may be good for exploration. In some cases where it is desirable to increase the number of items predicted to be more highly rated, it may be beneficial to use the expected change in the lower bound of the confidence interval for selecting an item [38], the higher the expected change the more desirable.

23.5.2 Decision Boundary Uncertainty

In Decision Boundary-based methods, training points are selected so as to improve decision boundaries. Often an existing decision boundary is assumed to be somewhat accurate, so points are sampled close to the decision boundary to further refine it (Figure 23.4). In a way this may also be considered Output Uncertainty-based, since the uncertainty of the points close to the decision boundary may be high. This method operates with the assumption that the decision boundary of the underlying learning method (e.g. Support Vector Machine) is easily accessible. A clear advantage of this method is that given a decision boundary, selecting training examples by their proximity to it is computationally inexpensive.

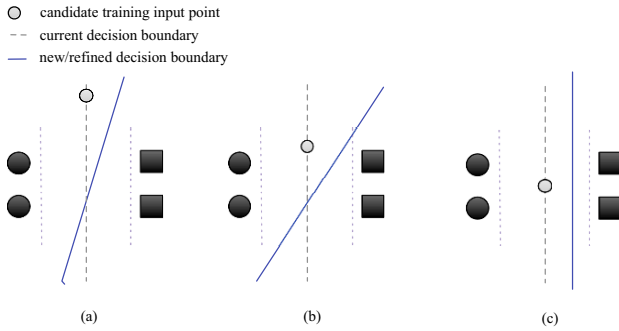


Fig. 23.4: Decision boundary uncertainty.

As discussed in [45], training points may be selected for obtaining a more accurate dividing hyperplane (Figure 23.4 (b)), or if the direction of the hyperplane is already certain, input points may be selected for reducing the size of margin (Figure 23.4 (c)). While it may seem obvious to sample training points closest to the decision boundary [55, 15], there are also methods that select the items furthest away [15] that have potential advantages in scenarios involving several candidate classifiers, which are discussed in Section 23.7. This is because a classifier should be quite certain about any items far from a decision boundary, but if newly acquired training data reveals the classifier to be inaccurate, the classifier may not fit the user’s preferences well, so it should be removed from the pool of candidate classifiers.

23.5.3 Model Uncertainty

Model Uncertainty-based methods select training points for the purpose of reducing uncertainty within the model, more specifically, to reduce uncertainty about the model’s parameters. The assumption is that if we improve the accuracy of the model’s parameters the accuracy of output values will improve as well. If we were to predict a user’s preferences based on membership in different interest groups [23], i.e. a group of people with a similar interest, then training points may be selected so as to determine to which groups the user belongs (Section 23.5.3.1).

23.5.3.1 Probabilistic Models

Probabilistic models are best explained with an example. The aspect model [23], a probabilistic latent semantic model in which users are considered to be a mixture of multiple interests (called aspects) is a good choice for this. Each user $u \in U$ has a probabilistic membership in different interest groups $z \in Z$. Users in the same interest group are assumed to have the same rating patterns (e.g. two users of the same

aspect will rate a given movie the same), so users and items $\mathbf{x} \in \mathcal{X}$ are independent from each other given the latent class variable z . The probability of the user u assigning an item \mathbf{x} the rating y can be computed as follows:

$$P(y|\mathbf{x}, u) = \sum_{z \in \mathcal{Z}} p(y|\mathbf{x}, z) p(z|u). \quad (23.15)$$

The first term $p(y|\mathbf{x}, z)$ is the likelihood of assigning an item \mathbf{x} the rating y by users in class z (approximated by a Gaussian distribution in [23]). It does not depend on the target user and represents the group-specific model. The global-model consists of a collection of group-specific models. The second term $p(z|u)$ is the likelihood for the target user u to be in class z , referred to as a user personalization parameter (approximated by a multinomial distribution in [23]). The user model $\boldsymbol{\theta}_u$ consists of one or more user personalization parameters, i.e. $\boldsymbol{\theta}_u = \{\theta_{u_z} = p(z|u)\}_{z \in \mathcal{Z}}$.

A traditional AL approach would be to measure the usefulness of the candidate training input point \mathbf{x}_a based on how much it would allow for reduction of the uncertainty about the user model's parameters $\boldsymbol{\theta}_u$ (i.e. the uncertainty about to which interest group z the user u belongs):

$$\widehat{G}_{\theta \text{Uncertainty}}(\mathbf{x}_a) = \text{Uncertainty}(\boldsymbol{\theta}_u), \quad (23.16)$$

$$\text{Uncertainty}(\boldsymbol{\theta}_u) = - \left\langle \sum_{z \in \mathcal{Z}} \theta_{u_z|\mathbf{x}_a, y} \log \theta_{u_z|\mathbf{x}_a, y} \right\rangle_{p(y|\mathbf{x}_a, \boldsymbol{\theta}_u)}, \quad (23.17)$$

where $\boldsymbol{\theta}_u$ denotes the currently estimated parameters of the user u and $\theta_{u_z|\mathbf{x}_a, y}$ a parameter that is estimated using an additional training point (\mathbf{x}_a, y) . Since the goal of the above criterion is to reduce the uncertainty of which interest groups the target user belongs to, it favors training points that assign a user to a *single* interest group. This approach may not be effective for all models, such as with the aspect model, in which a user's preferences are better modeled by considering that a user belongs to *multiple* interest groups [23, 25].

Another potential drawback comes from the expected uncertainty being computed over the distribution $p(y|\mathbf{x}, \boldsymbol{\theta}_u)$ by utilizing the currently estimated model $\boldsymbol{\theta}_u$. The currently estimated model could be far from the true model, particularly when the number of training points is small, but the number of parameters to be estimated is large. Therefore, performing AL based only on a single estimated model can be misleading [25]. Let us illustrate this by the following example shown in Figure 23.5. The four existing training points are indicated by solid line contours, test points by dashed ones. Based on these four training examples, the most likely decision boundary is the horizontal line (dashed), even though the true decision boundary is a vertical line (solid). If we select training input points based only on the estimated model, subsequent training points would likely be obtained from areas along the estimated boundary, which are ineffective in adjusting the estimated decision boundary (horizontal line) towards the correct decision boundary (vertical line). This example illustrates that performing AL for the currently estimated model

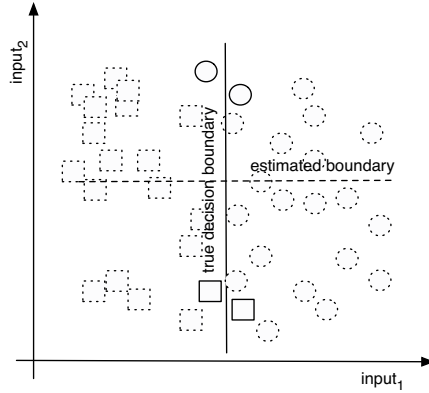


Fig. 23.5: A learning scenario when the estimated model is far from the true model. Training points are indicated by solid contours.

without taking into account the model’s uncertainty can be very misleading, particularly when the estimated model is far from the true model. A better strategy could be to consider model uncertainty by utilizing the model distribution for selecting training input points [25]. This would allow for adjusting the decision boundary more effectively since decision boundaries other than the estimated one (i.e. horizontal line) would be considered for selecting the training input points. This idea is applied to probabilistic models in [25] as follows. The usefulness of the candidate training input point is measured based on how much it allows adjusting the model’s parameters θ_u towards the optimal model parameters θ_u^* :

$$\widehat{G}_{\theta_{Uncertainty}}(\mathbf{x}_a) = \left\langle \sum_{z \in Z} \theta_{u_z}^* \log \frac{\theta_{u_z} | \mathbf{x}_a, y}{\theta_{u_z}^*} \right\rangle_{p(y | \mathbf{x}_a, \theta_{u^*})} \quad (23.18)$$

The above equation corresponds to Kullback–Leibler divergence which is minimized when the estimated parameters are equal to the optimal parameters. The true model θ_{u^*} is not known but could be estimated as the expectation over the posterior distribution of the user’s model i.e. $p(\theta_u | u)$.

23.6 Error-based Active Learning

Error-based Active Learning methods aim to reduce the predictive error, which is often the final goal. Instance-based approaches try to find and utilize the relation between the training input points and the predictive error. Model-based approaches tend to aim at reducing the model error (i.e. the error of model parameters), which is hoped would result in the improvement of predictive error.

23.6.1 Instance-based Methods

Instance-based methods aim at reducing error based on the properties of the input points, such as are listed in Section 23.2.

23.6.1.1 Output Estimates Change (Y-Change)

This approach [42] operates on the principle that if rating estimates do not change then they will not improve. Thus, if the estimates of output values do change, then their accuracy may either increase or decrease. However, it is expected that at least something will be learned from a new training point, so it follows then that in many cases estimates do in fact become more accurate. Assuming that most changes in estimates are for the better, an item that causes many estimates to change will result in the *improvement* of many estimates, and is considered useful.

As an example (Figure 23.6), if a user rates an item that is representative of a large genre, such as the Sci-Fi movie *Star Wars*, then its rating (regardless of its value) will likely cause a change in rating estimates for many other related items (e.g. items within that genre), in other words, rating such a representative item is very informative about the user's preferences. On the other hand, the user rating an item without many other similar items, such as the movie *The Goldfish Hunter*, would change few rating estimates, and supply little information.

To find the expected changes in rating estimates caused by a candidate item's rating, all possible item ratings are considered (since the true rating of a candidate item is not yet known). The difference is calculated between rating estimates for each item for each of its possible ratings, before and after it was added to the training set (refer to the pseudocode in Algorithm 1).

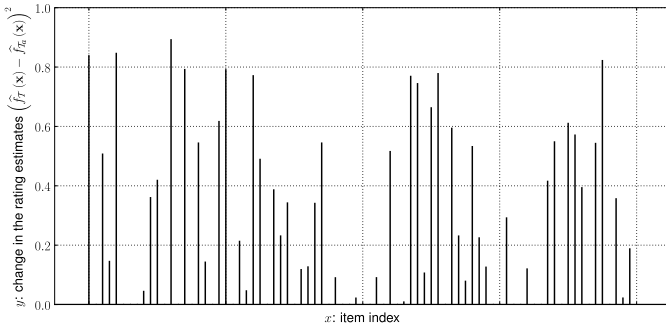
More formally the above criterion could be expressed as:

$$\widehat{G}_{Y\text{change}}(\mathbf{x}_a) = - \sum_{\mathbf{x} \in \mathcal{X}^{(\text{Test})}} \mathbb{E}_{y \in \mathcal{Y}} \mathcal{L}(\widehat{f}_{\mathcal{T}}(\mathbf{x}), \widehat{f}_{\mathcal{T} \cup (\mathbf{x}_a, y)}(\mathbf{x})), \quad (23.19)$$

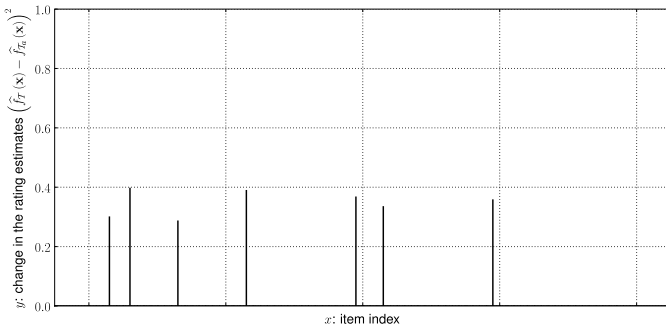
where $\widehat{f}_{\mathcal{T}}(\mathbf{x})$ is the estimated rating for an item \mathbf{x} given the current training set \mathcal{T} , and $\widehat{f}_{\mathcal{T} \cup (\mathbf{x}_a, y)}(\mathbf{x})$ is the rating's estimate after a hypothetical rating y of an item \mathbf{x}_a is added to the training set \mathcal{T} , and \mathcal{L} is the loss function that measures the differences between the rating estimates $\widehat{f}_{\mathcal{T}}(\mathbf{x})$ and $\widehat{f}_{\mathcal{T} \cup (\mathbf{x}_a, y)}(\mathbf{x})$. By assuming that ratings of a candidate item are equally likely and using a mean squared loss function, the above criterion could be written as:

$$\widehat{G}_{Y\text{change}}(\mathbf{x}_a) = - \sum_{\mathbf{x} \in \mathcal{X}^{(\text{Test})}} \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} \left(\widehat{f}_{\mathcal{T}}(\mathbf{x}) - \widehat{f}_{\mathcal{T} \cup (\mathbf{x}_a, y)}(\mathbf{x}) \right)^2 \quad (23.20)$$

where $\frac{1}{|\mathcal{Y}|}$ is a normalizing constant since we assume all possible ratings $y \in \mathcal{Y}$ of an item \mathbf{x}_a .



(a)



(b)

Fig. 23.6: Output estimate-based AL (Section 23.6.1.1). The x -axis corresponds to an item’s index, and the y -axis to the changes in rating estimates with regard to a candidate training point. Training points that cause many changes in rating estimates are considered to be more informative (a).

The advantage of this criterion is that it relies only on the *estimates* of ratings, available from any learning method. It has a further advantage of utilizing all unrated items, something that differentiates it from other methods in which only a small subset of all items (ones that have been rated by the user) are considered. It also works in tandem with any of a variety of learning methods, enabling it to potentially adapt to different tasks.

23.6.1.2 Cross Validation-based

In this approach a training input point is selected based on how well it may allow for approximation of already known ratings, i.e. items in the training set [15]. That is, a

Algorithm 1 Output estimates-based Active Learning (Section 23.6.1.1).

```

#  $\widehat{G}$  estimates predictive error that rating an item  $\mathbf{x}_a$  would achieve
function  $\widehat{G}(\mathbf{x}_a)$ 
  # learn a preference approximation function  $\widehat{f}$  based on the current training set  $\mathcal{T}$ 
   $\widehat{f}_{\mathcal{T}} = \text{learn}(\mathcal{T})$ 
  # for each possible rating of an item  $\mathbf{x}_a$  e.g.  $\{1, 2, \dots, 5\}$ 
  for  $y_a \in \mathcal{Y}$ 
    # add a hypothetical training point  $(\mathbf{x}_a, y_a)$ 
     $\mathcal{T}^{(a)} = \mathcal{T} \cup (\mathbf{x}_a, y_a)$ 
    # learn a new preference approximation function  $\widehat{f}$  based on the new training set  $\mathcal{T}^{(a)}$ 
     $\widehat{f}_{\mathcal{T}^{(a)}} = \text{learn}(\mathcal{T}^{(a)})$ 
    # for each unrated item
    for  $\mathbf{x} \in \mathcal{X}^{(Test)}$ 
      # record the differences between ratings estimates
      # before and after a hypothetical training point  $(\mathbf{x}_a, y_a)$  was added to the training set  $\mathcal{T}$ 
       $\widehat{G} = \widehat{G} + \left( - \left( \widehat{f}_{\mathcal{T}}(\mathbf{x}) - \widehat{f}_{\mathcal{T}^{(a)}}(\mathbf{x}) \right)^2 \right)$ 
  return  $\widehat{G}$ 

```

candidate training point \mathbf{x}_a with each possible rating $y \in \mathcal{Y}$ is added to the training set \mathcal{T} , then an approximation of the user's preferences \widehat{f} is obtained and its accuracy is evaluated (i.e. cross-validated) on the training items $\mathcal{X}^{(Train)}$. It is assumed that when the candidate training item is paired with its correct rating, the cross-validated accuracy will improve the most. The usefulness of the candidate training point is measured by the improvement in the cross-validated accuracy as following:

$$\widehat{G}_{CV\mathcal{T}}(\mathbf{x}_a) = - \max_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}^{(Train)}} \mathcal{L}(\widehat{f}_{\mathcal{T} \cup (\mathbf{x}_a, y)}(x), f(x)), \quad (23.21)$$

where \mathcal{L} is a loss function such as MAE or MSE (Section 23.4), and $f(x)$ is the actual rating of the item \mathbf{x} , and $\widehat{f}_{\mathcal{T} \cup (\mathbf{x}_a, y)}(x)$ is the approximated rating (where a function \widehat{f} is learned from the training set $\mathcal{T} \cup (\mathbf{x}_a, y)$).

A potential drawback is that training points selected by this AL method could be overfitted to the training set.

23.6.2 Model-based

In model-based approaches training input points are obtained as to reduce the model's error, i.e. the error of the model's parameters. A potential drawback of this approach is that reducing the model's error may not necessarily reduce the prediction error which is the objective of AL.

23.6.2.1 Parameter Change-based

Parameter Change-based AL [49] favors items that are likely to influence the model the most. Assuming that changes in the model’s parameters are for the better, i.e. approach the optimal parameters, it is then beneficial to select an item that has the greatest impact on the model’s parameters:

$$\widehat{G}_{\theta change}(\mathbf{x}_a) = - \sum_{\theta} \mathbb{E}_{y \in \mathcal{Y}} \mathcal{L}(\theta_{\mathcal{T}}, \theta_{\mathcal{T} \cup (\mathbf{x}_a, y)}), \tag{23.22}$$

where $\theta_{\mathcal{T}}$ are the model’s parameters estimated from the current training set \mathcal{T} , and $\theta_{\mathcal{T} \cup (\mathbf{x}_a, y)}$ are the model’s parameter estimates after a hypothetical rating y of an item \mathbf{x}_a is added to the training set \mathcal{T} , and \mathcal{L} is the loss function that measures the differences between the parameters.

23.6.2.2 Variance-based

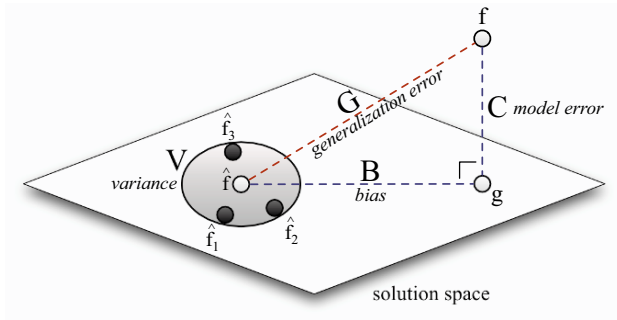


Fig. 23.7: Decomposition of generalization error G into model error C , bias B , and variance V , where g denotes optimal function, \hat{f} is a learned function \hat{f}_i 's are the learned functions from a slightly different training set.

In this approach the error is decomposed into three components: model error C (the difference between the optimal function approximation g , given the current model, and the true function f), bias B (the difference between the current approximation \hat{f} and an optimal one g), and variance V (how much the function approximation f varies). In other words, we have:

$$G = C + B + V. \tag{23.23}$$

One solution [13] is to minimize the variance component V of the error by assuming that the bias component becomes negligible (if this assumption is not satisfied then this method may not be effective). There are a number of methods proposed that

aim to select training inputs for reducing a certain measure of the variance of the model's parameters. The A-optimal design [11] seeks to select training input points so as to minimize the average variance of the parameter estimates, the D-optimal design [26] seeks to maximize the differential Shannon information content of the parameter estimates, and the Transductive Experimental design [56] seeks to find representative training points that may allow retaining most of the information of the test points. The AL method in [51], in addition to the variance component, also takes into account the existence of the model error component.

23.6.2.3 Image Restoration-based

It is also possible to treat the problem of predicting the user's preferences as one of image restoration [34], that is, based on our limited knowledge of a user's preferences (a partial picture), we try to restore the complete picture of the user's likes and dislikes. The AL task is then to select the training points that would best allow us to restore the "image" of the user's preferences. It is interesting to note that this approach satisfies the desired properties of the AL methods outlined in Section 23.2. For example, if a point already exists in a region, then without sampling neighboring points the image in that region could likely be restored. This approach also may favor sampling close to the edges of image components (decision boundaries).

23.7 Ensemble-based Active Learning

Sometimes instead of using a single model to predict a user's preferences, an ensemble of models may be beneficial (see Chapter 21). In other cases only a single model is used, but it is selected from a number of candidate models. The main advantage of this is the premise that different models are better suited to different users or different problems. The preferences of one user, for example, could be better modeled by a stereotype model, while the preferences of another user may be better modeled by a nearest-neighbor model. The training input points for these AL methods must be selected with regards to multiple models (Section 23.7.1) or multiple model candidates (Section 23.7.2).

23.7.1 Models-based

In Models-based approaches, the models form a "committee" of models that act, in a sense, cooperatively to select training input points [50]. Methods tend to differ with respect to: (1) how to construct a committee of models, and (2) how to select training points based on committee members [46]. As [46] explains thoroughly (please refer to it for more details), the Query by Committee approach (QBC) in-

volves maintaining a committee of models which are all trained on the same training data. In essence, they represent competing hypotheses for what the data might look like (as represented by the model). The members of this committee then vote on how to label potential input points (the “query” in “QBC”). The input points for which they disagree the most are considered to be the most informative. The fundamental premise of QBC is minimizing the version space, or the subset of all hypotheses that are consistent with all the collected training data; we want to then constrain the size of this space as much as possible, while at the same time minimizing the number of training input points. Put a different way, QBC “queries” in controversial regions to refine the version space.

There are many ways to construct the committee of models; [46] provides numerous examples. It can, for example, be constructed through simple sampling [50]. With generative model classes, this can be achieved by randomly sampling an arbitrary number of models from some posterior distribution, e.g. using the Dirichlet distribution over model parameters for naive Bayes [31], or sampling Hidden Markov Models (HMMs) using the Normal distribution [14]. The ensemble can be constructed for other model classes (such as discriminative or non-probabilistic models) as well, e.g. query-by-boosting and query-by-bagging [1], which employ the boosting [17] and bagging [8] ensemble learning methods to construct the committees; there has also been research [12] on using a selective sampling algorithm for neural networks that utilizes the combination of the “most specific” and “most general” models (selecting the models that lie at two extremes of the current version space given the current training set).

The “committee is still out” on the appropriate number of models to use, but even small sizes have demonstrated good results [50, 31, 47].

Measuring the disagreement between models is fundamental to the committee approach; there are two main means for calculating disagreement: vote uncertainty [14] and average Kullback-Leibler (KL) divergence [31]. Vote uncertainty selects the point with the largest disagreement between models of the committee. KL divergence is an information-theoretic measure of the difference between two probability distributions. KL divergence selects the input point with the largest average difference between the distributions of the committee consensus and the most differing model.

23.7.2 Candidates-based

Different models are better suited to different users or to different problems (see Chapter 2). So both the choice of the training set (AL) and the choice of the model, called Model Selection (MS), affect the predictive accuracy of the learned function. There is in fact a strong dependency between AL and MS, meaning that useful points for one model may not be as useful for another (Figure 23.9). This section discusses how to perform AL with regards to multiple model candidates and the issues that may arise when doing so.

The concept of *model* has several different meanings. We may refer to a model as a set of functions with some common characteristic, such as a function's complexity, or the type of a function or learning method (e.g. SVM, Naive Bayes, nearest-neighbor, or linear regression). The characteristics of the functions that may differ are often referred to as parameters. Thus, given a model and training data, the task of MS is to find parameters that may allow for accurate approximation of the target function. All of the model's characteristics affect the predictive accuracy, but for simplicity we concentrate only on the complexity of the model.

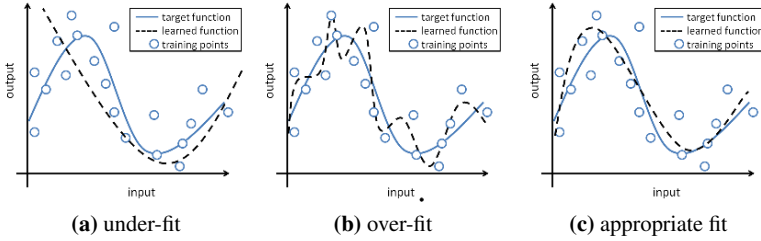


Fig. 23.8: Dependence between model complexity and accuracy.

As illustrated by Figure 23.8, if the model is too simple in comparison with the target function, then the learned function may not be capable of approximating the target function, making it under-fit (Figure 23.8a). On the other hand, if the model is too complex it may start trying to approximate irrelevant information (e.g. noise that may be contained in the output values) which will cause the learned function to over-fit the target function (Figure 23.8b). A possible solution to this is to have a number of candidate models. The goal of model selection (MS) is thus to determine the weights of the models in the ensemble, or in the case of a single model being used, to select an appropriate one (Figure 23.8c):

$$\min_{\mathcal{M}} G(\mathcal{M}). \quad (23.24)$$

The task of AL is likewise to minimize the predictive error, but with respect to the choice of the training input points:

$$\min_{\mathcal{X}^{(Train)}} G(\mathcal{X}^{(Train)}). \quad (23.25)$$

It would be beneficial to combine AL and MS since they share a common goal of minimizing the predictive error:

$$\min_{\mathcal{X}^{(Train)}, \mathcal{M}} G(\mathcal{X}^{(Train)}, \mathcal{M}). \quad (23.26)$$

Ideally we would like to choose the model of appropriate complexity by a MS method and to choose the most useful training data by an AL method. However

simply combining AL with MS in a batch manner, i.e. selecting all of the training points at once, may not be possible due to the following paradox:

- To select training input points by a standard AL method, a model must be fixed. In other words, MS has already been performed (see Figure 23.9).
- To select the model by a standard MS method, the training input points must be fixed and corresponding training output values must be gathered. In other words, AL has already been performed (see Figure 23.10).

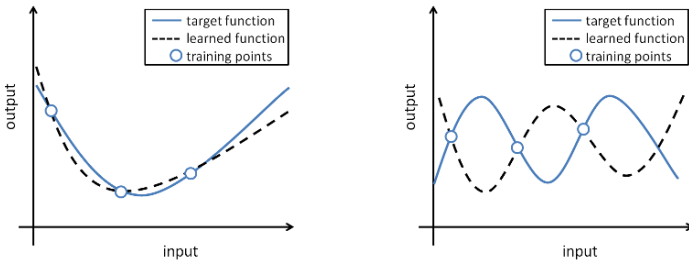
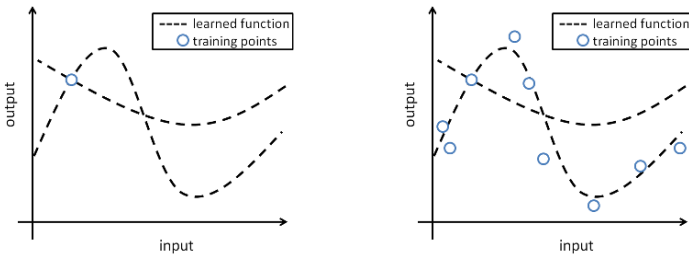


Fig. 23.9: Training input points that are good for learning one model, are not necessarily good for the other.



Unable to determine which model is more appropriate (Model Selection), until training points have been obtained (Active Learning).

Fig. 23.10: Dependence of Model Selection on Active Learning.

As a result Batch AL selects training points for a randomly chosen model, but after the training points are obtained the model is selected once again, giving rise to the possibility that the training points will not be as useful if the initial and final models differ. This means that the training points could be over-fitted to a possibly inferior model, or likewise under-fitted.

With Sequential AL, the training points and models are selected incrementally in a process of selecting a model, then obtaining a training point for this model, and so on. Although this approach is intuitive, it may perform poorly due to *model*

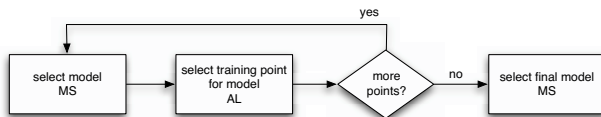


Fig. 23.11: Sequential Active Learning.

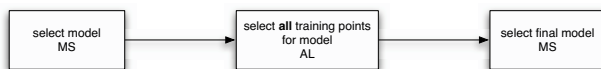


Fig. 23.12: Batch Active Learning.

drift, where a chosen model varies throughout the learning process. As the number of training points increases, more complex models tend to fit data better and are therefore selected over simpler models. Since the selection of training input points depends on the model, the training points chosen for a simpler model in the early stages could be less useful for the more complex model selected at the end of the learning process. Due to model drift, portions of training points are gathered for different models, resulting in the training data being not well suited for any of the models. However, because the selection of the final model is unclear at the onset, one possibility is to select training input points with respect to multiple models [52], by optimizing the training data for all the models:

$$\min_{\mathcal{X}^{(Train)}} \sum_{\mathcal{M}} \widehat{G}(\mathcal{X}^{(Train)}, \mathcal{M}) w(\mathcal{M}), \tag{23.27}$$

where $w(\mathcal{M})$ refers to the weight of the model in the ensemble, or among the candidates. This allows each model to contribute to the optimization of the training data and thus the risk of overfitting the training set to possibly inferior models can be hedged.

23.8 Conversation-based Active Learning

Differing from standard AL in which the goal is to obtain ratings for dissimilar items (for improving prediction accuracy over the entire set), Conversation-based AL is goal oriented with the task of starting general and, through a series of interaction cycles, narrowing down the user’s interests until the desired item is obtained [32, 37, 9], such as selecting a hotel to stay at during a trip. In essence, the goal is to supply the user with the information that best enables them to reduce the set of possible items, finding the item with the most utility. The system therefore aims at making accurate predictions about items with the highest utility for a potentially small group of items, such as searching for a restaurant within a restricted locale.

A common approach is to iteratively present sets of alternative recommendations to the user, and by eliciting feedback, guide the user towards an end goal in which the scope of interest is reduced to a single item. This cycle-based approach can be beneficial since users rarely know all their preferences at the start (becoming self-aware), but tend to form and refine them during the decision making process (exploration). Thus Conversation-based AL should also allow users to refine their preferences in a style suitable to the given task. Such systems, unlike general RSs, also include AL by design, since a user's preferences are learned through active interaction. They are often evaluated by the predictive accuracy, and also by the length of interaction before arriving at the desired goal.

23.8.1 Case-based Critique

One means for performing a conversation with a user is the Case-based Critique approach, which finds cases similar to the user's query or profile and then elicits a critique for refining the user's interests (see Chapters [37] and 13). As mentioned above (Section 23.8), the user is not required to clearly define their preferences when the conversation initiates; this may be particularly beneficial for mobile device-oriented systems. Each step of iteration displays the system's recommendations in a ranked list and allows for user critique, which will force the system to re-evaluate its recommendations and generate a new ranked list. Eliciting a user critique when a feature of a recommended item is unsatisfactory may be more effective in obtaining the end goal than mere similarity-based query revision combined with recommendation by proposing. As an example of a user critique, he/she may comment "I want a less expensive hotel room" or "I like restaurants serving wine."

23.8.2 Diversity-based

While suggesting items to the user that are similar to the user query is important (Section 23.8.1), it may also be worthwhile to consider diversity among the set of proposed items [32]. This is because if the suggested items are too similar to each other, they may not be representative of the current search space. In essence, the recommended items should be as representative and diverse as possible, which should be possible without appreciably affecting their similarity to the user query.

It is particularly important to provide diverse choices while the user's preferences are in their embryonic stages. Once the user knows what it is they want, providing items that match as closely as possible may be pertinent, and the AL technique used should attempt to make this distinction, i.e. if the recommendation space is properly focused, reduce diversity, and if incorrect, increase it.

23.8.3 Query Editing-based

Another possibility is to allow a user to repeatedly edit and resubmit a search query until their desired item is found [9]. Since it is an iterative process, the object is to minimize the number of queries needed before the user finds the item of highest utility. A query's usefulness is estimated based on the likelihood of the user submitting a particular query, along with its satisfiability, accomplished by observing user actions and inferring any constraints on user preferences related to item utility and updating the user's model. As an example, a user may query for hotels that have air-conditioning and a golf course. The RS can determine this to be satisfiable, and further infer that though the user is likely to add a restraint for the hotel being located in the city-center, no hotels match such criteria, so the system preemptively notifies the user that such a condition is unsatisfiable to prevent wasted user effort. The RS may also infer that for a small increase in price there are hotels with a pool and spa and a restaurant. Knowing the user's preferences for having a pool (and not for other options), the system would only offer adding the pool option, since it may increase the user's satisfaction, and not the others since they may overwhelm the user and decrease overall satisfaction.

23.9 Computational Considerations

It is also important to consider the computational costs of AL algorithms. [40] have suggested a number of ways of reducing the computational requirements, summarized (with additions) below.

- Many AL select an item to be rated based on its expected effect on the learned function. This may require retraining with respect to each candidate training item, and so efficient incremental training is crucial. Typically this step-by-step manner has lower cost than starting over with a large set.
- New rating estimates may need to be obtained with respect to each candidate item. Likewise, this could be done in an incremental manner, since only the estimates that change would need to be obtained again.
- It is possible to incrementally update the estimated error only for items likely to be effected by the inclusion of a training point, which in practice is only nearby items or items without similar features. A common approach is to use inverted indices to group items with similar features for quick lookup.
- A candidate training item's expected usefulness can likely be estimated using a subset of all items.
- Poor candidates for training points can be partially pruned through a pre-filtering step that removes poor candidate items based on some criteria, such as filtering books written in a language the user cannot read. A suboptimal AL method may be a good choice for this task.

23.10 Discussion

Though very brief, hopefully the collection of Active Learning methods presented in this chapter has demonstrated that AL is indeed not only beneficial but also desirable for inclusion in many systems, namely Recommender Systems. It can be seen that due to individual characteristics, the AL method selected, in many cases, relies heavily on the specific objectives (Section 23.1.1) that must be satisfied, either due to business constraints, preferred system behavior, user experience, or a combination of these (and possibly others). In addition to AL objectives, it is also prudent to evaluate the computational costs (Section 23.9) of any methods under consideration for use, and their trade-offs. Despite the success that many of the methods discussed have received, there is also something to be said for abstracting the problem, or finding solutions to other problems that though seemingly unrelated, may have strikingly similar solutions (e.g. Image Restoration (Section 23.6.2.3)). We have also touched upon conversation-based systems (Section 23.8) which differ from traditional RSs, but include the notion of AL by design. Depending on the task at hand, such as specific goal oriented assistants, this may also be a nice fit for a Recommender System.

Some issues related to AL have already been well studied in Statistics; this is not the case in Computer Science, where research is still wanting. Recommender Systems are changing at a rapid pace and becoming more and more complex. An example of this is the system that won the Netflix Recommendation Challenge, which combined multiple predictive methods in an ensemble manner (see Chapter 5). Given the high rate of change in predictive methods of RSs, and their complex interaction with AL, there is an ever increasing need for new approaches.

Improving accuracy has traditionally been the main focus of research. Accuracy alone, however, may not be enough to entice the user with RSs. This is because the system implementing AL may also need to recommend items of high novelty/serendipity, improve coverage, or maximize profitability, to name a few [44, 21, 33]. Another aspect that is frequently overlooked by AL researchers is the manner in which a user can interact with AL to reap improvements in performance. Simply presenting items to the user for rating lacks ingenuity to say the least; surely there is a better way? One example of this is a work [3] which demonstrated that by using the right interface even such menial tasks as labeling images could be made fun and exciting. With the right interface alone the utility of an AL system may increase dramatically.

Many issues remain that must be tackled to ensure the longevity of AL in RSs; with a little innovation and elbow grease we hope to see it transform from a “bothersome process” to an enjoyable one of self-discovery and exploration, satisfying both the system objectives and the user at the same time.

Acknowledgments

We would like to express our appreciation to Professor Okamoto, Professor Ueno, Professor Tokunaga, Professor Tomioka, Dr. Sheinman, Dr. Vilenius, Sachi Kabasawa and Akane Odake for their help and assistance, and also to MEXT and JST for their financial support; comments received from reviewers and editors were also indispensable to the writing process.

References

1. Abe, N., Mamitsuka, H.: Query learning strategies using boosting and bagging. In: Proceedings of the Fifteenth International Conference on Machine Learning, vol. 388. Morgan Kaufmann Publishers Inc. (1998)
2. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749 (2005)
3. Ahn, L.V.: Games with a purpose. *Computer* **39**(6), 92–94 (2006). DOI 10.1109/MC.2006.196
4. Bailey, R.A.: Design of Comparative Experiments. Cambridge University Press (2008)
5. Balcan, M.F., Beygelzimer, A., Langford, J.: Agnostic active learning. In: ICML '06: Proceedings of the 23rd international conference on Machine learning, pp. 65–72. ACM, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1143844.1143853>
6. Boutilier, C., Zemel, R., Marlin, B.: Active collaborative filtering. In: Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence, pp. 98–106 (2003). URL citeseer.ist.psu.edu/boutilier03active.html
7. Box, G., Hunter, S.J., Hunter, W.G.: Statistics for Experimenters: Design, Innovation, and Discovery. Wiley-Interscience (2005)
8. Breiman, L., Breiman, L.: Bagging predictors. In: Machine Learning, pp. 123–140 (1996)
9. Bridge, D., Ricci, F.: Supporting product selection with query editing recommendations. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 65–72. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1297231.1297243>
10. Carenini, G., Smith, J., Poole, D.: Towards more conversational and collaborative recommender systems. In: IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces, pp. 12–18. ACM, New York, NY, USA (2003). DOI <http://doi.acm.org/10.1145/604045.604052>
11. Chan, N.: A-optimality for regression designs. Tech. rep., Stanford University, Department of Statistics (1981)
12. Cohn, D.A.: Neural network exploration using optimal experiment design **6**, 679–686 (1994). URL citeseer.ist.psu.edu/article/cohn94neural.html
13. Cohn, D.A., Ghahramani, Z., Jordan, M.I.: Active learning with statistical models. *Journal of Artificial Intelligence Research* **4**, 129–145 (1996)
14. Dagan, I., Engelson, S.: Committee-based sampling for training probabilistic classifiers. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 150–157. Citeseer (1995)
15. Danziger, S., Zeng, J., Wang, Y., Brachmann, R., Lathrop, R.: Choosing where to look next in a mutation sequence space: Active learning of informative p53 cancer rescue mutants. *Bioinformatics* **23**(13), 104–114 (2007)

16. Dasgupta, S., Lee, W., Long, P.: A theoretical analysis of query selection for collaborative filtering. *Machine Learning* **51**, 283–298 (2003). URL citeseer.ist.psu.edu/dasgupta02theoretical.html
17. Freund, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* **55**(1), 119–139 (1997)
18. Fujii, A., Tokunaga, T., Inui, K., Tanaka, H.: Selective sampling for example-based word sense disambiguation. *Computational Linguistics* **24**, 24–4 (1998)
19. Greiner, R., Grove, A., Roth, D.: Learning cost-sensitive active classifiers. *Artificial Intelligence* **139**, 137–174 (2002)
20. Harpale, A.S., Yang, Y.: Personalized active learning for collaborative filtering. In: *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 91–98. ACM, New York, NY, USA (2008). DOI <http://doi.acm.org/10.1145/1390334.1390352>
21. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **22**(1), 5–53 (2004). DOI <http://doi.acm.org/10.1145/963770.963772>
22. Hinkelmann, K., Kempthorne, O.: *Design and Analysis of Experiments*, Advanced Experimental Design. Wiley Series in Probability and Statistics (2005)
23. Hofmann, T.: Collaborative filtering via gaussian probabilistic latent semantic analysis. In: *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 259–266. ACM, New York, NY, USA (2003). DOI <http://doi.acm.org/10.1145/860435.860483>
24. Huang, Z.: Selectively acquiring ratings for product recommendation. In: *ICEC '07: Proceedings of the ninth international conference on Electronic commerce*, pp. 379–388. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1282100.1282171>
25. Jin, R., Si, L.: A bayesian approach toward active learning for collaborative filtering. In: *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 278–285. AUAI Press, Arlington, Virginia, United States (2004)
26. John, R.C.S., Draper, N.R.: D-optimality for regression designs: A review. *Technometrics* **17**(1), 15–23 (1975)
27. Kapoor, A., Horvitz, E., Basu, S.: Selective supervision: Guiding supervised learning with decision-theoretic active learning. In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 877–882 (2007)
28. Kohrs, A., Merialdo, B.: Improving collaborative filtering for new users by smart object selection. In: *Proceedings of International Conference on Media Features (ICMF)* (2001)
29. Leino, J., Räihä, K.J.: Case amazon: ratings and reviews as part of recommendations. In: *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pp. 137–140. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1297231.1297255>
30. Lomasky, R., Brodley, C., Aernecke, M., Walt, D., Friedl, M.: Active class selection. In: *Proceedings of the European Conference on Machine Learning (ECML)*. Springer (2007)
31. McCallum, A., Nigam, K.: Employing em and pool-based active learning for text classification. In: *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 350–358. San Francisco, CA, USA (1998)
32. McGinty, L., Smyth, B.: On the Role of Diversity in Conversational Recommender Systems. *Case-Based Reasoning Research and Development* pp. 276–290 (2003)
33. McNee, S.M., Riedl, J., Konstan, J.A.: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pp. 1097–1101. ACM Press, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1125451.1125659>
34. Nakamura, A., Abe, N.: Collaborative filtering using weighted majority prediction algorithms. In: *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 395–403. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
35. Rashid, A.M., Albert, I., Cosley, D., Lam, S.K., McNee, S.M., Konstan, J.A., Riedl, J.: Getting to know you: learning new user preferences in recommender systems. In: *IUI '02: Pro-*

- ceedings of the 7th international conference on Intelligent user interfaces, pp. 127–134. ACM Press, New York, NY, USA (2002). DOI <http://doi.acm.org/10.1145/502716.502737>
36. Rashid, A.M., Karypis, G., Riedl, J.: Influence in ratings-based recommender systems: An algorithm-independent approach. In: SIAM International Conference on Data Mining, pp. 556–560 (2005)
 37. Ricci, F., Nguyen, Q.N.: Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intelligent Systems* **22**(3), 22–29 (2007). DOI <http://dx.doi.org/10.1109/MIS.2007.43>
 38. Rokach, L., Naamani, L., Shmilovici, A.: Pessimistic cost-sensitive active learning of decision trees for profit maximizing targeting campaigns. *Data Mining and Knowledge Discovery* **17**(2), 283–316 (2008). DOI <http://dx.doi.org/10.1007/s10618-008-0105-2>
 39. Rokach, L. and Maimon, O. and Arbel, R., Selective voting-getting more for less in sensor fusion, *International Journal of Pattern Recognition and Artificial Intelligence* **20** (3) (2006), pp. 329–350.
 40. Roy, N., Mccallum, A.: Toward optimal active learning through sampling estimation of error reduction. In: *In Proc. 18th International Conf. on Machine Learning*, pp. 441–448. Morgan Kaufmann (2001)
 41. Rubens, N., Sugiyama, M.: Influence-based collaborative active learning. In: *Proceedings of the 2007 ACM conference on Recommender systems (RecSys 2007)*. ACM (2007). DOI <http://doi.acm.org/10.1145/1297231.1297257>
 42. Rubens, N., Tomioka, R., Sugiyama, M.: Output divergence criterion for active learning in collaborative settings. *IPSJ Transactions on Mathematical Modeling and Its Applications* **2**(3), 87–96 (2009)
 43. Saar-Tsechansky, M., Provost, F.: Decision-centric active learning of binary-outcome models. *Information Systems Research* **18**(1), 4–22 (2007). DOI <http://dx.doi.org/10.1287/isre.1070.0111>
 44. Schein, A.I., Popescul, A., Ungar, L.H., Pennock, D.M.: Methods and metrics for cold-start recommendations. In: *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 253–260. ACM, New York, NY, USA (2002). DOI <http://doi.acm.org/10.1145/564376.564421>
 45. Schohn, G., Cohn, D.: Less is more: Active learning with support vector machines. In: *Proc. 17th International Conf. on Machine Learning*, pp. 839–846. Morgan Kaufmann, San Francisco, CA (2000). URL citeseer.ist.psu.edu/schohn00less.html
 46. Settles, B.: Active learning literature survey. *Computer Sciences Technical Report 1648*, University of Wisconsin–Madison (2009)
 47. Settles, B., Craven, M.: An analysis of active learning strategies for sequence labeling tasks. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1069–1078. ACL Press (2008)
 48. Settles, B., Craven, M., Friedland, L.: Active learning with real annotation costs. In: *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*, pp. 1–10 (2008)
 49. Settles, B., Craven, M., Ray, S.: Multiple-instance active learning. In: *Advances in Neural Information Processing Systems (NIPS)*, vol. 20, pp. 1289–1296. MIT Press (2008)
 50. Seung, H.S., Opper, M., Sompolinsky, H.: Query by committee. In: *Computational Learning Theory*, pp. 287–294 (1992). URL citeseer.ist.psu.edu/seung92query.html
 51. Sugiyama, M.: Active learning in approximately linear regression based on conditional expectation of generalization error. *Journal of Machine Learning Research* **7**, 141–166 (2006)
 52. Sugiyama, M., Rubens, N.: A batch ensemble approach to active learning with model selection. *Neural Netw.* **21**(9), 1278–1286 (2008). DOI <http://dx.doi.org/10.1016/j.neunet.2008.06.004>
 53. Sugiyama, M., Rubens, N., Mueller, K.R.: Dataset Shift in Machine Learning, chap. A conditional expectation approach to model selection and active learning under covariate shift. MIT Press, Cambridge (2008)
 54. Swearingen, K., Sinha, R.: Beyond algorithms: An hci perspective on recommender systems. *ACM SIGIR 2001 Workshop on Recommender Systems* (2001).

55. Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. In: P. Langley (ed.) Proceedings of ICML-00, 17th International Conference on Machine Learning, pp. 999–1006. Morgan Kaufmann Publishers, San Francisco, US, Stanford, US (2000). URL citeseer.ist.psu.edu/article/tong01support.html
56. Yu, K., Bi, J., Tresp, V.: Active learning via transductive experimental design. In: Proceedings of the 23rd Int. Conference on Machine Learning ICML '06, pp. 1081–1088. ACM, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1143844.1143980>

Chapter 24

Multi-Criteria Recommender Systems

Gediminas Adomavicius, Nikos Manouselis and YoungOk Kwon

Abstract This chapter aims to provide an overview of the class of multi-criteria recommender systems. First, it defines the recommendation problem as a multi-criteria decision making (MCDM) problem, and reviews MCDM methods and techniques that can support the implementation of multi-criteria recommenders. Then, it focuses on the category of *multi-criteria rating recommenders* – techniques that provide recommendations by modelling a user’s utility for an item as a vector of ratings along several criteria. A review of current algorithms that use multi-criteria ratings for calculating predictions and generating recommendations is provided. Finally, the chapter concludes with a discussion on open issues and future challenges for the class of multi-criteria rating recommenders.

24.1 Introduction

The problem of recommendation has been identified as the way to help individuals in a community to find information or items that are most likely to be interesting to them or to be relevant to their needs [4, 39, 72]. Typically, it assumes that there is set *Users* of all the users of a system and set *Items* of all possible items that can be recommended to them. Then, the utility function that measures the appropriateness of recommending item $i \in Items$ to user $u \in Users$ is often defined as $R : Users \times Items \rightarrow R_0$, where R_0 typically is represented by non-negative integers

Gediminas Adomavicius, YoungOk Kwon
Department of Information and Decision Sciences
Carlson School of Management, University of Minnesota, Minneapolis, MN 55455, USA
e-mail: {gedas,kwonx052}@umn.edu

Nikos Manouselis
Greek Research and Technology Network (GRNET S.A.)
56 Messogeion Av., 115 27, Athens, Greece
e-mail: nikosm@grnet.gr

or real numbers within a certain range [4]. It is assumed that this function is not known for the whole $Users \times Items$ space but is specified only on some subset of it. Therefore, in the context of recommendation, we want for each user $u \in Users$ to be able to (a) estimate (or approximate) the utility function $R(u, i)$ for item $i \in Items$ for which $R(u, i)$ is not yet known, and (b) choose one or a set of items i that will maximize $R(u, i)$, i.e.,

$$\forall u \in Users, i = \arg \max_{i \in Items} R(u, i) \quad (24.1)$$

In most recommender systems, the utility function usually considers a single-criterion value, e.g., an overall evaluation or rating of an item by a user. In recent work, this assumption has been considered as limited [2, 4, 48], because the suitability of the recommended item for a particular user may depend on more than one utility-related aspect that the user takes into consideration when making the choice. Particularly in systems where recommendations are based on the opinion of others, the incorporation of multiple criteria that can affect the users' opinions may lead to more accurate recommendations.

Thus, the additional information provided by *multi-criteria ratings* could help to improve the quality of recommendations because it would be able to represent more complex preferences of each user. As an illustration, consider the following example. In a traditional single-rating movie recommender system, user u provides a single rating for movie i that the user has seen, denoted by $R(u, i)$. Specifically, suppose that the recommender system predicts the rating of the movie that the user has not seen based on the movie ratings of other users with similar preferences, who are commonly referred to as "neighbors" [71]. Therefore, the ability to correctly determine the users that are most similar to the target user is crucial in order to have accurate predictions or recommendations. For example, if two users u and u' have seen three movies in common, and both of them rated their overall satisfaction from each of the three movies as 6 out of 10, the two users are considered as neighbors and the ratings of unseen movies for user u are predicted using the ratings of user u' .

In contrast, in a multi-criteria rating setting, users can provide ratings on multiple attributes of an item. For example, a two-criterion movie recommender system allows users to specify their preferences on two attributes of a movie (e.g., story and visual effects). A user may like the story, but dislike the visual effects of a movie, e.g., $R(u, i) = (9, 3)$. If we simply use two ratings with the same weight in making recommendations, rating their overall satisfaction as 6 out of 10 in the single-rating application might correspond to a variety of situations in multi-rating application: $(9, 3)$, $(6, 6)$, $(4, 8)$, etc. Therefore, although the ratings of the overall satisfaction are stated as 6, two users may show different rating patterns on each criterion of an item, e.g., user u gives ratings $(9, 3)$, $(9, 3)$, $(9, 3)$, and user u' gives ratings $(3, 9)$, $(3, 9)$, $(3, 9)$ to the same three movies. This additional information on each user's preferences would help to model users' preferences more accurately, and new recommendation techniques need to be developed to take advantage of this additional information. The importance of studying multi-criteria recommender systems

has been highlighted as a separate strand in the recommender systems literature [2, 4, 48], and recently several recommender systems (as we present later in this chapter) have been adopting multiple criteria ratings, instead of traditional single-criterion ratings. Thus, the aim of this chapter is to provide an overview of systems that use multiple criteria to support recommendation (referred to as *multi-criteria recommender systems*), with a particular emphasis on *multi-criteria rating* ones.

The remainder of this chapter is organized as follows. First, we overview the generic recommendation problem under the prism of multi-criteria decision making (MCDM), and demonstrate the potential of applying MCDM methods to facilitate recommendation in multi-criteria settings. Second, we focus on the particular type of multi-criteria recommender systems that use multi-criteria ratings, referred to as *multi-criteria rating recommenders* because, while it has not been extensively researched, this type of systems has significant potential for better recommendation performance. We survey the state of the art algorithms for this type of recommender systems. Finally, research challenges and future research directions in multi-criteria recommender systems are discussed.

24.2 Recommendation as a Multi-Criteria Decision Making Problem

In order to introduce multiple criteria in the generic recommendation problem, one of the classic MCDM methodologies can be followed. To facilitate the discussion on how MCDM methods and techniques can be used when developing a recommender system, we followed the steps and notations proposed by Bernard Roy (one of the 1960s pioneers in MCDM methods) in the generic modeling methodology for decision making problems [77]. The discussion could also follow some other generic MCDM modeling methodologies [24, 34, 95, 97], since the scope of this section is to provide some initial insights into issues that recommender systems researchers should consider when designing a multi-criteria recommender.

Roy's [77] methodology includes four steps when analyzing a decision making problem:

1. *Defining the object of decision.* That is, defining the set of alternatives (items) upon which the decision has to be made and the rationale of the recommendation decision.
2. *Defining a consistent family of criteria.* That is, identifying and specifying a set of functions that declare the preferences of the decision maker (targeted user) upon the various alternatives. These should cover all the parameters affecting the recommendation decision and be exhaustive and non-redundant.
3. *Developing a global preference model.* That is, defining the function that synthesizes the partial preferences upon each criterion into a model that specifies the total preference of a decision maker regarding a candidate alternative.

4. *Selection of the decision support process.* This covers the design and development of the procedure, methods, or software systems that will support a decision maker when taking a decision about the set of alternatives (items), in accordance to the results of the previous steps.

We briefly review these steps in separate subsections below, and mention how each of them pertains to recommender systems.

24.2.1 Object of Decision

In recommender systems, the object of decision is item i that belongs to the set of all the candidate items. The elements of this set are referred to as *alternatives* or actions in related literature [30]. To express the rationale behind the decision, Roy [77] refers to the notion of the decision “problematics.” Four types of decision problematics are identified:

- *Choice*, which concerns the selection of one or more alternatives that can be considered as more appropriate from all candidate ones;
- *Sorting*, which refers to the classification of the alternatives into a number of pre-defined categories;
- *Ranking*, which involves ranking all the alternatives, from the best one to the worst;
- *Description*, which concerns the description of each alternative in terms of how it performs upon each criterion.

All four types of decision problematics can be considered valid for the recommendation problem:

- Choosing and recommending one or more items as more suitable for a particular user;
- Classifying (or sorting, as Roy defines it) all available items into pre-defined categories according to their suitability, e.g., into “recommended for purchase” and “recommended for viewing” items;
- Ranking all available items from the most suitable to the least suitable ones for a particular user, and presenting a ranked list of recommendations to the user;
- Describing how suitable a particular item is for a specific user, based on how it is evaluated upon each criterion. It corresponds to a full analysis of the item performance upon all criteria, illustrating the suitability of an item for the specific user (that is, in a personalized manner that aims to help the user to make a selection).

24.2.2 Family of Criteria

The performance of alternatives in set *Items* is analyzed upon a set of criteria for each user, in order to model all their characteristics, attributes, effects, or consequences [77, 97]. In recommender systems, the criteria may refer to the multiple features of an item (often the case in content-based recommendations) or to the multiple dimensions upon which the item is being evaluated/rated.

Any criterion c can be represented by function $g_c(i)$ that expresses the preferences of one user (therefore is user-specific), in order for the user to be able to decide between two alternatives i_1 and i_2 , i.e., whether $g_c(i_1) > g_c(i_2)$, in the case that alternative i_1 is preferred to alternative i_2 , or whether $g_c(i_1) = g_c(i_2)$, in the case that the two alternatives are considered equivalent (i.e., perfectly substitutable for the particular user on this criterion). To be able to make rational decisions using multiple criteria, it has to be ensured that the whole set of these functions creates a *consistent* family of criteria [77]. A family of criteria is said to be consistent when it has the following three properties:

1. *Monotonic*: a family of criteria is monotonic only if, for each pair of alternatives i_1 and i_2 , for which $g_{c_1}(i_1) > g_{c_1}(i_2)$ for one criterion c_1 and $g_c(i_1) = g_c(i_2)$ for every other criterion $c \neq c_1$, it can be assumed that alternative i_1 is preferred to alternative i_2 .
2. *Exhaustive*: a family of criteria is exhaustive only if, for each pair of alternatives i_1 and i_2 , for which $g_c(i_1) = g_c(i_2)$ upon each criterion c , we can assume that i_1 and i_2 are equivalent.
3. *Non-redundant*: a family of criteria is non-redundant only if the removal of any one of the criteria leads to the violation of one of the other two properties.

In the remainder of this chapter, unless explicitly specified otherwise, we will assume that we have a consistent family of k criteria, i.e., g_1, g_2, \dots, g_k . The design of a consistent family of criteria for a given recommendation application has been largely ignored in the recommender systems literature and constitutes an interesting and important problem for future research. Four types of criteria are usually found in MCDM [30]:

- *Measurable*, i.e., a criterion that allows its quantified measurement upon some evaluation scale;
- *Ordinal*, i.e., a criterion that defines an ordered set of acceptable values that allow its evaluation using a qualitative or a descriptive scale;
- *Probabilistic*, i.e., a criterion that uses probability distributions to represent uncertainty in its evaluation;
- *Fuzzy*, i.e., a criterion whose evaluation is represented in relation to its possibility to belong in one of the intervals of a qualitative or descriptive evaluation scale.

From a broad perspective, a family of criteria can be used to facilitate the representation of user preferences in recommender systems as well. Therefore, we can

assume that all types of criteria could be potentially engaged in multi-criteria recommender systems, although (as shown later) it seems that some types are used in currently developed systems more often than others.

24.2.3 Global Preference Model

The development of a global preference model provides a way to aggregate the values of each criterion g_c (where $c = 1, \dots, k$) in order to express the preferences between the different alternatives of the set *Items*, depending on the selected decision problematics. In the MCDM literature, a number of methodologies have been developed, which can be classified in different categories according to the form of the global preference model that they use and the process of creating this model. According to [30] and [64], the following categories of global preference modeling approaches can be identified:

- *Value-Focused models*, where a value system for aggregating the user preferences on the different criteria is constructed. In such approaches, marginal preferences upon each criterion are synthesized into a total value function, which is usually called the utility function [33]. These approaches are often referred to as multi-attribute utility theory (MAUT) approaches.
- *Multi-Objective Optimization models*, where criteria are expressed in the form of multiple constraints of a multi-objective optimization problem. In such approaches, usually the goal is to find a Pareto optimal solution for the original optimization problem [102]. They are also sometimes referred to as multi-objective mathematical programming methodologies.
- *Outranking Relations models*, where preferences are expressed as a system of outranking relations between the items, thus allowing the expression of incomparability. In such approaches, all items are pair-wise compared to each other, and preference relations are provided as relations “ a is preferred to b ”, “ a and b are equally preferable”, or “ a is incomparable to b ” [76].
- *Preference Disaggregation models*, where the preference model is derived by analyzing past decisions. Such approaches are sometimes considered as a subcategory of other modeling categories mentioned above, since they try to infer a preference model of a given form (e.g., value function or outranking relations) from some given preferential structures that have led to particular decisions in the past. Inferred preference models aim at producing decisions that are at least identical to the examined past ones [30].

Methodologies from all categories can be used in order to create global preference models for recommender systems, depending on the selected decision problematic and the environment in which the recommender system is expected to operate.

24.2.4 Decision Support Process

In this step, a final decision for a given MCDM problem is made by choosing an appropriate method among the ones defined in each of the previous steps. Like in traditional MCDM, multi-criteria recommendation problems may also need to use different methods for different domains or applications. Note, however, that this MCDM perspective is broad and not very restrictive when modeling multi-criteria recommendation problems, because many existing recommender systems can be thought to fit directly in the MCDM category, since they usually take into account information from multiple sources (e.g., user profiles and item attributes), thus making them *de facto* multi-criteria decision makers. Therefore, later in the chapter, we will focus on a particular category of MCDM recommender systems that can be differentiated from most existing recommender systems.

In Tables 24.1-24.3, we provide an overview of some sample recommender systems that could be broadly classified as MCDM (or multi-criteria recommender) systems based on the work of [48]. This survey covers systems that use one of the MCDM methods discussed in the previous section and, thus, provides insights into the way that existing MCDM approaches can be employed to support the decision-making in recommender systems.

The multi-criteria recommender systems are categorized according to the decision problematic they support (Table 24.1), the types of criteria they use (Table 24.2), and the global preference modelling approach they follow (Table 24.3). Based on Table 24.1, it is interesting to note that most of the existing research focuses on the decision problematic of *ranking* the items (i.e., ranking candidates for recommendation). There are also several systems that support the *sorting* of items into different categories according to their suitability for the user (e.g., recommended vs. non-recommended items). Very few systems support the *choice* and *description* problematic, although clearly there exist some applications in which they would prove relevant. Furthermore, as Table 24.2 illustrates, the families of criteria used are mainly *measurable*: that is, users rate items upon a measurable scale for each criterion. Nevertheless, there are also several systems that engage *fuzzy*, *ordinal*, and *probabilistic* criteria for the expression of user preferences regarding the candidate items. Finally, Table 24.3 indicates that only a few of the multi-criteria recommenders engage in the creation of the global preference model using a *multi-objective optimization* or *outranking relations*. On the contrary, the vast majority uses some *value-focused* model that typically calculates prediction in the form of an additive utility function. There are also some systems that do not synthesize the predictions from the multiple criteria, but rather use the raw vector models as their outcome (e.g., by providing a vector of ratings from all the criteria).

It is important to note that existing systems are sometimes violating the consistency rules that Roy's methodology proposes (e.g., not using an exhaustive set of dimensions). Nevertheless, experimental results often indicate that performance of multi-criteria systems is satisfactory (e.g., see the survey of algorithms that follows) even in cases where no formal modelling methodology has been followed.

This could mean that a modelling inconsistency does not always imply problematic performance, although this is an issue that calls for further investigation.

Table 24.1: Decision problematics supported by existing multi-criteria recommender systems

Choice	Ariely et al. 2004 [6], Falle et al. 2004 [23], Kleinberg and Sandler 2003 [38], Lee et al. 2002 [45], Lee 2004 [44], Price and Messinger 2005 [69], Tewari et al. 2003 [93]
Sorting	Cantador et al. 2006 [12], Choi and Cho 2004 [15], Emi et al. 2003 [22], Guan et al. 2002 [28], Kim and Yang 2004 [36], Liu and Shih 2005 [47], Masthoff 2003 [53], Montaner et al. 2002 [57], Nguyen and Haddawy 1999 [60], Nguyen and Haddawy 1998 [59], Stolze and Rjaibi 2001 [90], Wang 2004 [99], Yu 2002 [100], Yu 2004 [101], Zimmerman et al. 2004 [103]
Ranking	Adomavicius and Kwon 2007 [2], Ardissono et al. 2003 [5], Balabanovic and Shoham 1997 [7], Ghosh et al. 1999 [27], Karacapilidis and Hatzieleftheriou 2005 [32], Kerschberg et al. 2001 [35], Kim et al. 2002 [37], Lakiotaki et al. 2008 [42], Lee and Tang 2007 [43], Lee et al. 2002 [45], Li et al. 2008 [46], Manouselis and Costopoulou 2007b [49], Manouselis and Costopoulou 2007c [50], Manouselis and Sampson 2004 [52], Mukherjee et al. 2001 [58], Noh 2004 [61], Perny and Zucker 1999 [66], Perny and Zucker 2001 [67], Plantie et al. 2005 [68], Ricci and Werthner 2002 [73], Ricci and Nguyen 2007 [74], Sahoo et al. 2006 [79], Schafer 2005 [82], Schickel-Zuber and Faltings 2005 [83], Srikumar and Bhasker 2004 [89], Tang and McCalla 2009 [92], Tsai et al. 2006 [96]
Description	Aciar et al. 2007 [1], Cheetham 2003 [14], Denguir-Rekik et al. 2006 [19], Herrera-Viedma et al. 2004 [29], Schmitt et al. 2002 [84], Schmitt et al. 2003 [85], Stolze and Stroebel 2003 [91]

24.3 MCDM Framework for Recommender Systems: Lessons Learned

While, as mentioned earlier, the recommender systems surveyed in Tables 24.1-24.3 can be considered to be multi-criteria recommender systems according to the MCDM framework, it is important to understand where the existing types of recommender systems fall within this framework and also whether this MCDM framework gives rise to any novel types of recommender systems.

Recommendation techniques are often classified based on the recommendation approach into several categories: content-based, collaborative filtering, knowledge-based, and hybrid approaches [7]. Content-based recommendation techniques find the best recommendations for a user based on what the user liked in the past [65], and collaborative filtering recommendation techniques make recommendations based on the information about other users with similar preferences [8]. Knowledge-based approaches use knowledge about users and items to find the items

Table 24.2: Criteria types engaged in existing multi-criteria recommender systems

Measurable	Adomavicius and Kwon 2007 [2], Ariely et al. 2004 [6], Balabanovic and Shoham 1997 [7], Cantador et al. 2006 [12], Choi and Cho 2004 [15], Falle et al. 2004 [23], Ghosh et al. 1999 [27], Guan et al. 2002 [28], Kerschberg et al. 2001 [35], Kim and Yang 2004 [36], Kim et al. 2002 [37], Lakiotaki et al. 2008 [42], Lee and Tang 2007 [43], Lee 2004 [44], Lee et al. 2002 [45], Li et al. 2008 [46], Liu and Shih 2005 [47], Manouselis and Costopoulou 2007b [49], Manouselis and Costopoulou 2007c [50], Manouselis and Sampson 2004 [52], Masthoff 2003 [53], Montaner et al. 2002 [57], Mukherjee et al. 2001 [58], Noh 2004 [61], Plantie et al. 2005 [68], Ricci and Werthner 2002 [73], Ricci and Nguyen 2007 [74], Sahoo et al. 2006 [79], Schafer 2005 [82], Schickel-Zuber and Faltings 2005 [83], Schmitt et al. 2003 [85], Schmitt et al. 2002 [84], Srikumar and Bhasker 2004 [89], Stolze and Rjaibi 2001 [90], Tang and McCalla 2009 [92], Tewari et al. 2003 [93], Tsai et al. 2006 [96], Yu 2002 [100], Yu 2004 [101], Zimmerman et al. 2004 [103]
Ordinal	Aciar et al. 2007 [1], Cheetham 2003 [14], Emi et al. 2003 [22], Nguyen and Haddawy 1998 [59], Nguyen and Haddawy 1999 [60]
Fuzzy	Herrera-Viedma et al. 2004 [29], Karacapilidis and Hatzieleftheriou 2005 [32], Perny and Zucker 1999 [66], Perny and Zucker 2001 [67], Stolze and Stroebel 2003 [91], Wang 2004 [99]
Probabilistic	Ardissono et al. 2003 [5], Kleinberg and Sandler 2003 [38], Price and Messinger 2005 [69]

that meet users' requirements [9]. The bottleneck of this knowledge-based approach is that it needs to acquire a knowledge base beforehand, but the obtained knowledge base helps to avoid cold start or data sparsity problems that pure content-based or collaborative filtering systems encounter by relying on solely the ratings obtained by users. Hybrid approaches combine content-based, collaborative filtering, and knowledge-based techniques in many different ways [10]. Upon more in-depth analysis of the representative MCDM recommender systems surveyed in the previous section, we discover that the multi-criteria nature of the majority of these systems can be classified in the following three general categories:

- *Multi-attribute content preference modeling.* Even though these systems typically use single-criterion ratings (e.g., numeric or binary ratings), for any given user these systems attempt to understand and model the commonalities of multi-attribute content among the items the user preferred in the past, and recommend to the user the items that best match this preferred content. For example, in a movie recommender system, these commonalities may be represented by specific genres, actors, directors, etc. that the user's preferred movies have in common.
- *Multi-attribute content search and filtering.* These systems allow a user to specify her general preferences on content-based attributes across all items, through searching or filtering processes (e.g., searching for only "comedy" movies or specifying that "comedy" movies are preferable to "action" movies), and rec-

Table 24.3: Global preference models used in existing multi-criteria recommender systems

Value-focused models	Aciar et al. 2007 [1], Adomavicius and Kwon 2007 [2], Ariely et al. 2004 [6], Balabanovic and Shoham 1997 [7], Cantador et al. 2006 [12], Choi and Cho 2004 [15], Denguir-Rekik et al. 2006 [19], Falle et al. 2004 [23], Ghosh et al. 1999 [27], Guan et al. 200 [28], Herrera-Viedma et al. 2004[29], Karacapilidis and Hatzieleftheriou 2005 [32], Kerschberg et al. 2001 [35], Kim and Yang 2004 [36], Kim et al. 2002 [37], Kleinberg and Sandler 2003 [38],Lakiotaki et al. 2008 [42], Lee 2004 [44], Lee et al. 2002 [45], Li et al. 2008 [46], Liu and Shih 2005 [47], Manouselis and Costopoulou 2007b [49], Manouselis and Costopoulou 2007c [50], Manouselis and Sampson 2004 [52], Masthoff 2003 [53], Montaner et al. 2002 [57], Mukherjee et al. 2001 [58], Noh 2004 [61], Perny and Zucker 1999 [66], Perny and Zucker 2001 [67], Plantie et al. 2005 [68], Ricci and Werthner 2002 [73], Sahoo et al. 2006 [79], Schafer 2005 [82], Schickel-Zuber and Faltings 2005 [83], Schmitt et al. 2003 [85], Schmitt et al. 2002 [84], Srikumar and Bhasker 2004 [89], Stolze and Stroebel 2003 [91], Stolze and Rjaibi 2001 [90],Tang and McCalla 2009 [92], Tsai et al. 2006 [96], Yu 2004 [101], Yu 2002 [100], Zimmerman et al. 2004 [103]
Optimization	Lee and Tang 2007 [43], Price and Messinger 2005 [69], Tewari et al. 2003 [93]
Outranking relations	Emi et al. 2003 [22], Nguyen and Haddawy 1999 [60], Nguyen and Haddawy 1999 [59]
Other preference models	Ardissono et al. 2003 [5], Cheetham 2003 [14], Lee et al. 2002 [45], Ricci and Nguyen 2007 [74], Wang 2004 [99]

ommend to the user the items that are the most similar to her preferences and satisfy specified search and/or filtering conditions.

- *Multi-criteria rating-based preference elicitation.* These systems allow a user to specify her individual preferences by rating each item on multiple criteria (e.g., rating the story of movie *Wanted* as 2 and the visual effects of the same movie as 5), and recommend to the user the items that can best reflect the user's individual preferences based on the multi-criteria ratings provided by this and other users.

Multi-attribute content preference modeling. One way to model user preferences is by analyzing multi-attribute content of items that users purchased or liked. Many multi-criteria recommender systems incorporate these content-based features either directly into the recommendation process (i.e., use a content-based approach) or in combination with collaborative recommendation techniques (i.e., use a hybrid approach). In these systems, users are typically allowed to implicitly or explicitly express their preferences with single-criterion ratings (e.g., item purchase history or single numeric ratings). Using these ratings, recommender systems then can learn users' content-based preferences in an automated fashion by finding the commonalities among the individual content attributes of items that the users purchased or liked, e.g., by identifying favorite content attributes (e.g., "comedy" movies) for each user. As a result, recommendations are made taking into account these fa-

favorite content attributes [7]. Numerous traditional recommender systems that employ content-based, knowledge-based, or hybrid approaches in combination with some multi-attribute preference modeling of users can be found in this category. Several scoring or utility functions have been developed and used to rank the candidate items based on users' content-based preferences, including information retrieval-based and model-based techniques, such as Bayesian classifiers and various machine learning techniques [4]. More details on these techniques are discussed in other chapters 3.

Multi-attribute content search and filtering. In some systems, users can explicitly provide their *general* preferences on multi-attribute content of items that can be used by various searching and filtering techniques to find the most relevant items. For example, in [82] users can identify the movie genre, MPAA rating, and film length that they like and specify which attribute is the most important for their decision in choosing the movies at the current time. Then the recommender system narrows down the possible choices by searching for the items that match these additional explicit user preferences. For example, if a user indicates that she wants to watch “comedy” movies and the movie genre is the most important attribute for her, she will be recommended only comedy movies. Similarly, in [45], users also can provide to the recommender system both the preferred specifications for different content attributes as well as the corresponding importance weights for the different attributes.

Some of knowledge-based recommender systems [35, 37] can also be classified into this category, because users can provide their general preferences by building their own hierarchical taxonomy tree (i.e., where all item features are modeled in a hierarchical manner) and assigning the relative importance level to each component in the tree. As a result, the systems recommend the most relevant items according to users' preferences upon the user-defined multiple attributes of item taxonomy. Furthermore, some of hybrid recommender systems with knowledge-based approach would also fit in this category, particularly case-based reasoning recommender systems, where items are represented with multi-criteria content in a structured way (i.e., using a well-defined set of features and feature values) [88]. These systems allow users to specify their preferences on multi-attribute content of items in their search for items of interest. For example, several case-based travel recommender systems [73, 75] filter out unwanted items based on each user's preferences on multi-attribute content (e.g., locations, services, and activities), and find personalized travel plans for each user by ranking possible travel plans based on the user's preferences and past travel plans of this or similar users. In addition, some case-based recommender systems [9, 70] allow users to “critique” the recommendation results by refining their requirements as part of the interactive and iterative recommendation process, which uses various search and filtering techniques to continuously provide the user with the updated set of recommendations. For example, when searching for a desktop PC, users can critique the current set of provided recommendations by expressing their refined preferences on individual features (e.g.,

cheaper price) or multiple features together (e.g., higher processor speed, RAM, and hard-disk capacity).

Multi-criteria rating-based preference elicitation. This category of recommender systems engage multi-criteria ratings, often by extending traditional collaborative filtering approaches, that show users' *subjective* preferences for various *components* of individual items. For instance, such systems allow users to rate not only the overall satisfaction from a particular movie, but also the satisfaction from the various movie components (factors), such as the visual effects, the story, or the acting. They differ from the above-surveyed systems in that the users do not indicate their preference or importance weight on the visual effects component for movies in general or to be used in a particular user query, but rather *how much* they liked the visual effects of the *particular* movie. One example of such system is the Intelligent Travel Recommender system [73], where users can rate multiple travel items within a "travel bag" (e.g., location, accommodation, etc.) as well as the entire travel bag. Then, candidate travel plans are ranked according to these user ratings, and the system finds the best match between recommended travel plans and the current needs of a user. These and similar types of multi-criteria rating-based systems are the focus of this chapter and more exemplar systems and techniques are provided in the later sections.

In summary, as seen above, many recommender systems that employ traditional content-based, knowledge-based, and hybrid techniques can be viewed as multi-criteria recommender systems, since they model user preferences based on multi-attribute content of items that users preferred in the past or allow users to specify their content-related preferences – i.e., search or filtering conditions for multi-attribute content of items (e.g., identifying the preferred movie genre or providing preferences on multiple pre-defined genre values). However, as mentioned earlier, there is a recent trend in multi-criteria recommendation that studies innovative approaches in collaborative recommendation by engaging multi-criteria ratings. We believe that this additional information on users' preferences offers many opportunities for providing novel recommendation support, creating a unique multi-criteria rating environment that has not been extensively researched. Therefore, in the following sections, we survey the state-of-the-art techniques on this particular type of systems that use individual ratings along multiple criteria, which we will refer to as *multi-criteria rating recommenders*.

24.4 Multi-Criteria Rating Recommendation

In this section, we define the multi-criteria rating recommendation problem by formally extending it from its single-rating counterpart, and provide some further discussion about the advantages that additional criteria may provide in recommender systems.

24.4.1 Traditional single-rating recommendation problem

Traditionally recommender systems operate in a two-dimensional space of *Users* and *Items*. The utility of items to users is generally represented by a totally ordered set R_0 (e.g., non-negative integers or real numbers within a certain range), and recommender systems aim to predict the utility of an item for a user. As mentioned earlier, a utility function R can be formally written as follows:

$$R : Users \times Items \rightarrow R_0 \tag{24.2}$$

The utility function is determined based on user inputs, such as numeric ratings that users explicitly give to items and/or transaction data that implicitly shows users' preferences (e.g., purchase history). The majority of traditional recommender systems use single-criterion ratings that indicate how much a given user liked a particular item in total (i.e., the overall utility of an item by a user). For example, in a movie recommender system, as shown in Table 24.4, user *Alice* may assign a single-criterion rating of 5 (out of 10) for movie *Wanted*, which can be denoted by $R(Alice, Wanted) = 5$. As an illustration, let us assume that the neighborhood-based collaborative filtering technique [71], i.e., one of the most popular heuristic-based recommendation techniques, is used for rating prediction. This technique predicts a user's rating for a given item based on the ratings of other users with similar preferences (i.e., neighbors). Particularly, in this example, the recommender system tries to predict the utility of movie *Fargo* for *Alice* based on the observed ratings. Since *Alice* and *John* show similar rating patterns on the four movies that both of them have previously seen and rated (see Table 24.4), for the purpose of this simple example the rating of movie *Fargo* for user *Alice* is predicted using *John's* rating (i.e., 9), although we would like to note that it is more common to use the ratings of more than one neighbor in a real system.

Table 24.4: Single-rating movie recommender system

Target user	Wanted	WALL-E	Star Wars	Seven	Fargo	Rating to be predicted
User most similar to the target user → Alice	5	7	5	7	?	←
→ John	5	7	5	7	9	← Rating to be used in prediction
Mason	6	6	6	6	5	
:	:	:	:	:	:	

24.4.2 Extending traditional recommender systems to include multi-criteria ratings

With a growing number of real-world applications, extending recommendation techniques to incorporate multi-criteria ratings has been regarded as one of the important issues for the next generation of recommender systems [4]. Examples of multi-criteria rating systems include Zagat's Guide that provides three criteria for restaurant ratings (e.g., food, décor, and service), Buy.com that provides multi-criteria ratings for consumer electronics (e.g., display size, performance, battery life, and cost), and Yahoo! Movies that show each user's ratings for four criteria (e.g., story, action, direction, and visuals). This additional information about users' preferences provided by multi-criteria ratings (instead of a single overall rating) can potentially be helpful in improving the performance of recommender systems.

Some multi-criteria rating systems can choose to model a user's utility for a given item with an overall rating R_0 as well as the user's ratings R_1, \dots, R_k for each individual criterion c ($c = 1, \dots, k$), whereas some systems can choose not to use the overall rating and focus solely on individual criteria ratings. Therefore, the utility-based formulation of the multi-criteria recommendation problem can be represented either with or without overall ratings as follows:

$$R : Users \times Items \rightarrow R_0 \times R_1 \times \dots \times R_k \quad (24.3)$$

or

$$R : Users \times Items \rightarrow R_1 \times \dots \times R_k \quad (24.4)$$

Given the availability of multi-criteria ratings (in addition to the traditional single overall rating) for each item, Tables 24.4 and 24.5 illustrate the potential benefits of this information for recommender systems. While *Alice* and *John* have similar preferences on movies in a single-rating setting (Table 24.4), in a multi-criteria rating setting we could see that they show substantially different preferences on several movie aspects, even though they had the same overall ratings (Table 24.5). Upon further inspection of all the multi-criteria rating information, one can see that *Alice* and *Mason* show very similar rating patterns (much more similar than *Alice* and *John*). Thus, using the same collaborative filtering approach as before, but taking into account multi-criteria ratings, *Alice*'s overall rating for movie *Fargo* would be predicted as 5, based on *Mason*'s overall rating for this movie.

This example implies that a single overall rating may hide the underlying heterogeneity of users' preferences for different aspects of a given item, and multi-criteria ratings may help to better understand each user's preferences, as a result enabling to provide users more accurate recommendations. It also illustrates how multi-criteria ratings can potentially produce more powerful and focused recommendations, e.g., by recommending movies that will score best on the story criterion, if this is the most important one for some user.

Table 24.5: Multi-criteria movie recommender system (ratings for each item: overall, story, action, direction, and visual effects)

Target user		Wanted	WALL-E	Star Wars	Seven	Fargo	
User most similar to the target user	Alice	5,2,2,8,8	7,5,5,9,9	5,2,2,8,8	7,5,5,9,9	2,2,2,2,2	Ratings to be predicted
	John	5,8,8,2,2	7,9,9,5,5	5,8,8,2,2	7,8,8,2,2	9,8,8,10,10	
	Mason	6,3,3,9,9	6,4,4,8,8	6,3,3,9,9	6,4,4,8,8	5,2,2,8,8	Ratings to be used in prediction
	:	:	:	:	:	:	

Therefore, new recommendation algorithms and techniques are needed that can utilize multi-criteria ratings in recommender systems. There are already several systems implementing such algorithms, which we analyze in the next section.

24.5 Survey of Algorithms for Multi-Criteria Rating Recommenders

Recommender systems typically calculate and provide recommendations using the following two-phase process:

- *Prediction:* the phase in which the prediction of a user’s preference is calculated. Traditionally, it is the phase in which a recommender estimates the utility function R for the entire or some part of $Users \times Items$ space based on known ratings and possibly other information (such as user profiles and/or item content); in other words, it calculates the predictions of ratings for the unknown items.
- *Recommendation:* the phase in which the calculated prediction is used to support the user’s decision by some recommendation process, e.g., the phase in which the user gets recommended a set of top- N items that maximize his/hers utility (i.e., N items with highest-predicted ratings).

Multi-criteria rating information can be used in both of these phases in different ways, and a number of approaches have been developed for the prediction or recommendation. Therefore, we classify the existing techniques for multi-criteria rating recommenders into two groups – techniques used during prediction and techniques used during recommendation – and describe these groups in more detail in separate subsections below.

24.5.1 Engaging Multi-Criteria Ratings during Prediction

This section provides an overview of the techniques that use multi-criteria ratings to predict an overall rating or individual criteria ratings (or both). In general, recommendation techniques can be classified by the formation of the utility function into two categories: heuristic-based (sometimes also referred to as memory-based) and model-based techniques [4]. Heuristic-based techniques compute the utility of each item for a user on the fly based on the observed data of the user and are typically based on a certain heuristic assumption. For example, a neighborhood-based technique – one of the most popular heuristic-based collaborative filtering techniques – assumes that two users who show similar preferences on the observed items will have similar preferences for the unobserved items as well. In contrast, model-based techniques learn a predictive model, typically using statistical or machine-learning methods, that can best explain the observed data, and then use the learned model to estimate the utility of unknown items for recommendations. Following this classification, we also present the algorithms of multi-criteria rating recommenders by grouping them into heuristic and model-based approaches.

24.5.1.1 Heuristic approaches

There has been some work done to extend the *similarity* computation of the traditional heuristic-based collaborative filtering technique to reflect multi-criteria rating information [2, 49, 92]. In this approach, the similarities between users are computed by aggregating traditional similarities from individual criteria or using multi-dimensional distance metrics.

In particular, the neighborhood-based collaborative filtering recommendation technique predicts unknown ratings for a given user, based on the known ratings of the other users with similar preferences or tastes (i.e., neighbors). Therefore, the first step of the prediction processes is to choose the similarity computation method to find a set of neighbors for each user. Various methods have been used for similarity computation in single-criterion rating recommender systems, and the most popular methods are correlation-based and cosine-based. Assuming that $R(u, i)$ represents the rating that user u gives to item i , and $I(u, u')$ represents the common items that two users u and u' rated, two popular similarity measures can be formally written as follows:

- *Pearson correlation-based:*

$$sim(u, u') = \frac{\sum_{i \in I(u, u')} (R(u, i) - \overline{R(u)}) (R(u', i) - \overline{R(u')})}{\sqrt{\sum_{i \in I(u, u')} (R(u, i) - \overline{R(u)})^2} \sqrt{\sum_{i \in I(u, u')} (R(u', i) - \overline{R(u')})^2}} \quad (24.5)$$

- *Cosine-based:*

$$sim(u, u') = \frac{\sum_{i \in I(u, u')} R(u, i) R(u', i)}{\sqrt{\sum_{i \in I(u, u')} R(u, i)^2} \sqrt{\sum_{i \in I(u, u')} R(u', i)^2}} \tag{24.6}$$

Multi-criteria rating recommenders cannot directly employ the above formulas, because $R(u, i)$ contains an overall rating r_0 , and k multi-criteria ratings r_1, \dots, r_k , i.e. $R(u, i) = (r_0, r_1, \dots, r_k)^1$. Thus, there are $k+1$ rating values for each pair of (u, i) , instead of a single rating. Two different similarity-based approaches that use $k + 1$ rating values in computing similarities between users have been used. The first approach *aggregates traditional similarities that are based on each individual rating*. This approach first computes the similarity between two users separately on each individual criterion, using any traditional similarity computation, such as correlation-based and cosine-based similarity. Then, a final similarity between two users is obtained by aggregating $k + 1$ individual similarity values. Adomavicius and Kwon [2] propose two aggregation approaches: an average and the worst-case (i.e., smallest) similarity, as specified in (24.7) and (24.8). As a general approach, Tang and McCalla [92], in their recommender system of research papers, compute an aggregate similarity as a weighted sum of individual similarities over several criteria of each paper (e.g., overall rating, value added, degree of being peer-recommended, and learners' pedagogical features such as interest and background knowledge) as specified in (24.9). In their approach, the weight of each criterion c , denoted by w_c , is chosen to reflect how important and useful the criterion is considered to be for the recommendation.

- *Average similarity:*

$$sim_{avg}(u, u') = \frac{1}{k + 1} \sum_{c=0}^k sim_c(u, u') \tag{24.7}$$

- *Worst-case(smallest) similarity:*

$$sim_{min}(u, u') = \min_{c=0, \dots, k} sim_c(u, u') \tag{24.8}$$

- *Aggregate similarity:*

$$sim_{aggregate}(u, u') = \sum_{c=0}^k w_c sim_c(u, u') \tag{24.9}$$

The second approach calculates similarity using *multidimensional distance metrics*, such as Manhattan, Euclidean, and Chebyshev distance metrics [2]. The distance between two users u and u' on item i , $d(R(u, i), R(u', i))$, can be calculated as:

- *Manhattan distance:*

$$\sum_{c=0}^k |R_c(u, i) - R_c(u', i)| \tag{24.10}$$

¹ In some recommender systems, $R(u, i)$ may not contain the overall ratings r_0 in addition to k multi-criteria ratings, i.e., $R(u, i) = (r_1, \dots, r_k)$. In this case, all the formulas in this subsection will still be applicable with index $c \in \{1, \dots, k\}$, as opposed to $c \in \{0, 1, \dots, k\}$.

- *Euclidean distance:*

$$\sqrt{\sum_{c=0}^k |R_c(u, i) - R_c(u', i)|^2} \quad (24.11)$$

- *Chebyshev (or maximal value) distance:*

$$\max_{c=0, \dots, k} |R_c(u, i) - R_c(u', i)| \quad (24.12)$$

The overall distance between two users can be simply an average distance for all common items that both users rated, and it can be formally written as:

$$dist(u, u') = \frac{1}{|I(u, u')|} \sum_{i \in I(u, u')} d(R(u, i), R(u', i)) \quad (24.13)$$

The more similar two users are (i.e., the larger the similarity value between them is), the smaller is the distance between them. Therefore, the following simple transformation is needed because of the inverse relationship of the two metrics:

$$sim(u, u') = \frac{1}{1 + dist(u, u')} \quad (24.14)$$

Manouselis and Costopoulou [49] also propose three different algorithms to compute *similarities* between users in multi-criteria rating settings: similarity-per-priority, similarity-per-evaluation, and similarity-per-partial-utility. The similarity-per-priority algorithm computes the similarities between users based on importance weights $w_c(u)$ of user u for each criterion c (rather than ratings $R(u, i)$). In this way, it creates a neighborhood of users that have the same expressed preferences with the target user. Then, it tries to predict the overall utility of an item for this user, based on the total utilities of the users in the neighborhood. In addition, the similarity-per-evaluation and similarity-per-partial-utility algorithms create separate neighborhoods for the target user for each criterion, i.e., they calculate the similarity with other users per individual criterion, and then predict the rating that the target user would provide upon each individual criterion. The similarity-per-evaluation algorithm calculates the similarity based on the non-weighted ratings that the users provide on each criterion. The similarity-per-partial-utility algorithm calculates the similarity based on the weighted (using $w_c(u)$ of each user u) ratings that the users provide on each criterion.

In such systems, the similarities between users are obtained using multi-criteria ratings, and the rest of the recommendation process can be the same as in single-criterion rating systems. The next step is, for a given user, to find a set of neighbors with the highest similarity values and predict unknown overall ratings of the user based on neighbors' ratings. Therefore, these similarity-based approaches are applicable only to neighborhood-based collaborative filtering recommendation techniques that need to compute the similarity between users (or items).

In summary, multi-criteria ratings can be used to compute the similarity between two users in the following two ways [2]: by (i) aggregating similarity values that are calculated separately on each criterion into a single similarity and (ii) calcu-

lating the distance between multi-criteria ratings directly in the multi-dimensional space. Empirical results using a small-scale Yahoo! Movies dataset show that both heuristic approaches outperform the corresponding traditional single-rating collaborative filtering technique (i.e., that uses only single overall ratings) by up to 3.8% in terms of precision-in-top- N metric, which represents the percentage of truly high overall ratings among those that the system predicted to be the N most relevant items for each user [2]. The improvements in precision depend on many parameters of collaborative filtering techniques, such as neighborhood sizes and the number of top- N recommendations. Furthermore, these approaches can be extended as suggested by Manouselis and Costopoulou [49] by computing similarities using not only known rating information, but also importance weights for each criterion. The latter approaches were evaluated in an online application that recommends e-markets to users, where multiple buyers and sellers can access and exchange information about prices and product offerings, based on users' multi-criteria evaluations on several e-markets. The similarity-per-priority algorithm using Euclidian distance performed the best among their proposed approaches in terms of the mean absolute error (MAE) (i.e., 0.235 on scale of 1 to 7) with a fairly high coverage (i.e., 93% of items can be recommended to users) as compared to non-personalized algorithms, such as arithmetic mean and random, that produce higher MAE (0.718 and 2.063, respectively) with 100% coverage [49].

24.5.1.2 Model-based approaches

Model-based approaches construct a predictive model to estimate unknown ratings by learning from the observed data. Several existing approaches for multi-criteria rating recommenders fall into this category, including aggregation function, probabilistic modeling, and multilinear singular value decomposition (MSVD).

Aggregation function approach. While overall rating r_0 is often considered simply as just another criterion rating in similarity-based heuristic approaches (as illustrated earlier), the aggregation function approach assumes that the overall rating serves as an aggregate of multi-criteria ratings [2]. Given this assumption, this approach finds aggregation function f that represents the relationship between overall and multi-criteria ratings, i.e.,

$$r_0 = f(r_1, \dots, r_k) \quad (24.15)$$

For example, in a movie recommendation application, the story criteria rating may have a very high “priority,” i.e., the movies with high story ratings are well liked overall by some users, regardless of other criteria ratings. Therefore, if the story rating of the movie is predicted high, the overall rating of the movie must also be predicted high in order to be accurate.

The aggregation function approach consists of three steps, as summarized in Fig. 24.1. First, this approach estimates k individual ratings using any recommendation technique. That is, the k -dimensional multi-criteria rating problem is decom-

posed into k single-rating recommendation problems. Second, aggregation function f is chosen using domain expertise, statistical techniques, or machine learning techniques. For example, the domain expert may suggest a simple average function of the underlying multi-criteria ratings for each item based on her prior experience and knowledge. An aggregation function also can be obtained by using statistical techniques, such as linear and non-linear regression analysis techniques, as well as various sophisticated machine learning techniques, such as artificial neural networks. Finally, the overall rating of each unrated item is computed based on the k predicted individual criteria ratings and the chosen aggregation function f .

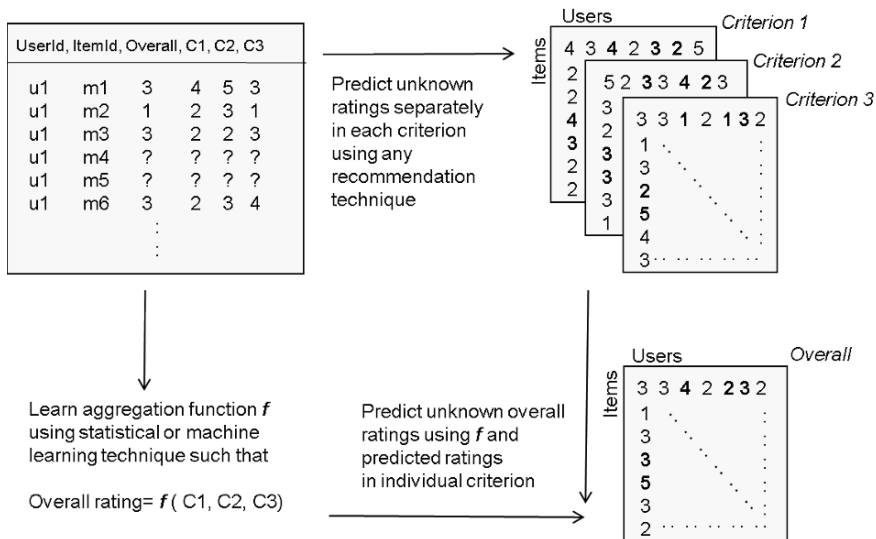


Fig. 24.1: Aggregation function approach (an example of a three-criteria rating system)

While the similarity-based heuristic approaches described earlier apply to only neighborhood-based collaborative filtering recommendation techniques, the aggregation function approach can be used in combination with any traditional recommendation technique, because individual criteria ratings are used for the prediction in the first step. As one example of possible aggregation functions, Adomavicius and Kwon [2] use linear regression and estimate coefficients (i.e., importance weights of each individual criterion) based on the known ratings.

Adomavicius and Kwon [2] also note that the aggregation function can have different scopes: total (i.e., when a single aggregation function is learned based on the

entire dataset), user-based or item-based (i.e., when a separate aggregation function is learned for each user or item).

Empirical analysis using data from Yahoo! Movies shows that the aggregation function approach (using multi-criteria rating information) outperforms a traditional single-rating collaborative filtering technique (using only overall ratings) by 0.3-6.3% in terms of precision-in-top- N ($N = 3, 5,$ and 7) metric [2].

Probabilistic modeling approach. Some multi-criteria recommendation approaches adopt probabilistic modeling algorithms that are becoming increasingly popular in data mining and machine learning. One example is the work of Sahoo et al. [79], which extends the flexible mixture model (FMM) developed by Si and Jin [86] to multi-criteria rating recommenders. The FMM assumes that there are two latent variables Z_u and Z_i (for users and items), and they are used to determine a single rating r of user u on item i , as shown in Fig. 24.2a. Sahoo et al. [79] also discover the dependency structure among the overall ratings (r_0) and multi-criteria ratings ($r_1, r_2, r_3,$ and r_4), using Chow-Liu tree structure discovery [16], and incorporate the structure into the FMM, as shown in Fig. 24.2b.

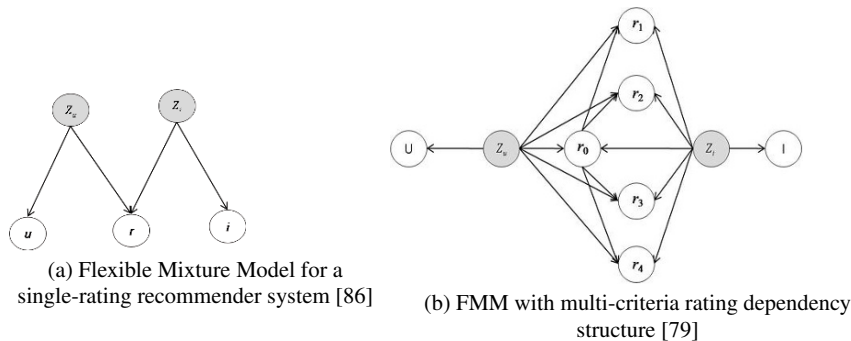


Fig. 24.2: Examples of probabilistic modeling approach in recommender systems

The FMM approach is based on the assumption that the joint distribution of three variables (user u , item i , and rating r) can be expressed using the sum of probabilities over the all possible combinations of the two latent class variables Z_u and Z_i , as follows.

$$P(u, i, r) = \sum_{Z_u, Z_i} P(Z_u)P(Z_i)P(u|Z_u)P(i|Z_i)P(r|Z_u, Z_i) \tag{24.16}$$

In summary, an overall rating of an unknown item for a target user is estimated with the following two steps: learning and prediction. In the first (learning) step, all the parameters of the FMM are estimated using the expectation maximization (EM)

algorithm [18]. Using the obtained parameters, in the second (prediction) step, the overall rating of a given unknown item is predicted as the most likely value (i.e., the rating value with the highest probability). This approach has been extended to multi-criteria ratings, and the detailed algorithm can be found in [79].

Sahoo et al. [79] also compare their model in Fig. 24.2b with the model that assumes independence among multi-criteria ratings conditional on the latent variables, and found that the model with dependency structure performs better than the one with the independence assumption. This finding demonstrates the existence of the “halo effect” in multi-criteria rating systems. The “halo effect” is a phenomenon often studied in psychometric literature, which indicates a cognitive bias whereby the perception of a particular object in one category influences the perception in other categories [94]. In multi-criteria recommender systems, the individual criterion ratings provided by users are correlated due to the “halo effect”, and particularly more correlated to an overall rating than to other individual ratings [79]. In other words, the overall rating given by the user to a specific item seems to affect how the user rates the other (individual) criteria of this item. Thus, controlling for an overall rating reduces this halo effect and helps to make individual ratings independent of each other, as represented in the chow-Liu tree dependency structure (Fig. 24.2b).

Using data from Yahoo! Movies, Sahoo et al. [79] show that multi-criteria rating information is advantageous over a single rating when very little training data is available (i.e., less than 15% of the whole data is used for training). On the other hand, when large training data is available, additional rating information does not seem to add much value. In this analysis, they measure the recommendation accuracy using the MAE metric. However, when they validate this probabilistic modeling approach using precision and recall metrics in retrieving top N items, their model performs better in all cases (i.e., both with small and large datasets) with a maximum of 10% increase. With more training data, the difference between the model with multi-criteria ratings and the traditional single-rating model diminishes in terms of precision and recall metrics.

Multilinear singular value decomposition (MSVD) approach. Li et al. [46] propose a novel approach to improve a traditional collaborative filtering algorithm by utilizing the MSVD technique. Singular value decomposition (SVD) techniques have been extensively studied in numerical linear algebra and have recently gained popularity in recommender systems applications because of their effectiveness in improving recommendation accuracy [26, 41, 81]. In single-rating recommender systems, these techniques identify latent features of items including well-defined item dimensions and uninterpretable dimensions. In particular, using K latent features (i.e., rank- K SVD), user u is associated with a user-factors vector p_u (the user’s preferences on K features), and item i is associated with an item-factors vector q_i (the item’s importance weights on K features). After all the values in user- and item-factors vectors are estimated, the preference of how much user u likes item i , denoted by $R^*(u, i)$, is predicted by taking an inner product of the two vectors, i.e.,

$$R^*(u, i) = p_u^T q_i \quad (24.17)$$

While the SVD techniques are commonly used as a decomposition method for two-dimensional data, the MSVD techniques [17] can be used for multi-dimensional data, such as multi-criteria ratings. In particular, Li et al. [46] use the MSVD to reduce the dimensionality of multi-criteria rating data, and evaluate their approach in the context of a restaurant recommender system, where a user rates a restaurant on 10 criteria (i.e., cuisine, ambience, service, etc.). More specifically, they use MSVD techniques to uncover relationships among users, items, and criteria and then use this information for identifying the nearest neighbors of each user and computing top- N recommendations. The results demonstrate that their approach improves the accuracy of recommendations (as measured by precision-in-top- N) by up to 5%, as compared to the traditional single-rating model.

In summary, the above approaches represent initial attempts to apply sophisticated learning techniques to address multi-criteria recommendation problems, and we expect to see more such techniques in the future.

In the next subsection, we discuss different approaches to recommending items to users, assuming that the unknown multi-criteria ratings have been estimated using any of the techniques discussed above.

24.5.2 Engaging Multi-Criteria Ratings during Recommendation

As mentioned above, multi-criteria recommender systems may choose to model a user's utility for a given item by including both the overall rating and ratings of individual item components/criteria or they may choose to include only ratings of individual criteria. If overall ratings are included as part of the model, the recommendation process in such cases is typically very straightforward: after predicting all unknown ratings, the recommender system uses the overall rating of items to select the most highly predicted items (i.e., the most relevant items) for each user. In other words, the recommendation process is essentially the same as in traditional, single-criterion recommender systems.

However, without an overall rating the recommendation process becomes more complex, because it is less apparent how to establish the total order of the items. For example, suppose that we have a two-criterion movie recommender system, where users judge movies based on their story (i.e., plot) and visual effects. Further, suppose that one movie needs to be chosen for recommendation among the following two alternatives: (i) movie X , predicted as 8 in story and 2 in visuals, and (ii) movie Y , predicted as 5 in story and 5 in visuals. Since there is no overall criterion to rank the movies, it is not easy to judge which movie is better, unless some other modeling approach is adopted, using some non-numerical (e.g., rule-based) way for expressing preferences. Several approaches have been proposed in the recommender systems literature to deal with this problem: some try to design a total order on items and obtain a single global optimal solution for each user, whereas others take one of the possible partial orders of the items and find multiple (Pareto optimal) solutions.

Below we briefly mention related work on multi-criteria optimization, describe several approaches that have been used in the recommender systems literature, and discuss other potential uses of multi-criteria ratings in the recommendation process.

24.5.2.1 Related work: multi-criteria optimization

Multi-criteria optimization problems have been extensively studied in the operations research (OR) literature [21], although not in the context of recommender systems. This multi-criteria optimization approach assists a decision maker in choosing the best alternative when multiple criteria conflict and compete with each other. For example, various points of view, such as financial, human resources-related, and environmental aspects should be considered in organizational decision making. The following approaches are often used to address multi-criteria optimization problems, and can be applied to recommender systems, as discussed in [4]:

- Finding Pareto optimal solutions;
- Taking a linear combination of multiple criteria and reducing the problem to the single-criterion optimization problem;
- Optimizing only the most important criterion and converting other criteria to constraints;
- Consecutively optimizing one criterion at a time, converting an optimal solution to constraints and repeating the process for other criteria.

Below we describe several recommendation approaches that have been used in the recommender systems literature, all of them having roots in multi-criteria optimization techniques.

24.5.2.2 Designing a total order for item recommendations

In the recommender systems literature there has been some work using multi-attribute utility theories from decision science, which can be described as one way to take a linear combination of multiple criteria and find an optimal solution [42].

For example, the approach by Lakiotaki et al. [42] ranks the items by adopting the UTilités Additive (UTA) method proposed by [87]. Their algorithm aims to estimate overall utility U of a specific item for each user by adding the marginal utilities of each criterion c ($c = 1, \dots, k$).

$$U = \sum_{c=1}^k u_c(g_c) \quad (24.18)$$

where g_c is the rating provided on criterion c , and $u_c(g_c)$ is a non-decreasing real-value function (marginal utility function) for a specific user. Since this model uses the ranking information with ordinal regression techniques, Kendall's tau is used as a measure of correlation between two ordinal-level variables to compare an actual

order and the predicted order. The empirical results obtained by using data from Yahoo! Movies show that 20.4% of users obtain a Kendall's tau of 1 indicating a total agreement of the orders between the ones predicted by the recommender system and the ones stated by users, and the mean value of Kendall's tau across all users is 0.74. Their model is also evaluated using the Receiver Operating Curve (ROC), which depicts relative trade-offs between true positives and false positives. The obtained Area Under Curve (AUC) of 0.81, where 1 represents a perfect classifier and 0.5 represents the performance of a random classifier, demonstrates that multi-criteria ratings provide measurable improvements in modeling users' preferences.

Similarly, Manouselis and Costopoulou [49] propose a method that calculates total utility U either by summing the k predicted partial utilities u_c (in their similarity-per-partial-utility algorithm) or by weighting the predicted ratings that the user would give on each criterion c by the user's importance weights w_c (in their similarity-per-evaluation algorithm). In both cases, the total utility of a candidate item is calculated using an aggregate function of the following form:

$$U = \sum_{c=1}^k u_c = \sum_{c=1}^k w_c r_c \quad (24.19)$$

Finally, once the total order on the candidate items is established using any of the above techniques, each user gets recommended the items that maximize this total utility.

24.5.2.3 Finding Pareto optimal item recommendations

This approach discovers several good items among large number of candidates (rather than arriving at a unique solution by solving a global optimization problem) when different items can be associated with multiple conflicting criteria and the total order on items is not directly available. *Data envelopment analysis* (DEA), often also called "frontier analysis", is commonly used to measure productive efficiency of decision making units (DMU) in operations research [13]. DEA computes the efficiency frontier, which identifies the items that are "best performers" overall, taking into account all criteria. DEA does not require *a priori* weights for each criterion, and uses linear programming to arrive more directly at the best set of weights for each DMU. Specifically, in the context of multi-criteria recommender systems, given all the candidate items that are available for recommendation to a given user (including the information about their predicted ratings across all criteria), DEA would be able to determine the reduced set of items (i.e., the frontier) that have best ratings across all criteria among the candidates. These items then can be recommended to the user.

While DEA has not been directly used in multi-criteria rating recommenders, the multi-criteria recommendation problem without overall ratings can also be formulated as a data query problem in the database field, using similar motivation [43]. Lee and Teng [43] utilize *skyline* queries to find the best restaurants across multiple criteria (i.e., food, décor, service, and cost). As Fig. 24.3 shows, skyline queries

identify a few skyline points (i.e., Pareto optimal points) that are not dominated by any others from a large number of candidate restaurants in two-dimensional data space (food and décor). Here, for a given user, a candidate item is considered to be dominated, if there exists another candidate item that has better or equal ratings on all criteria.

Empirical results using multi-criteria ratings of Zagat Survey in [43] show that the recommender system using skyline queries helps to reduce the number of choices that users should consider from their inquiries. For example, when a user searches for buffet restaurants which are located in New York City with a cost of no more than \$30, the system recommends only two restaurants among twelve candidate restaurants, based on the ratings on four criteria. However, this preliminary work needs to be extended in several directions because the skyline queries may not scale well with the increasing number of criteria, resulting in a large number of skyline points with high computational cost.

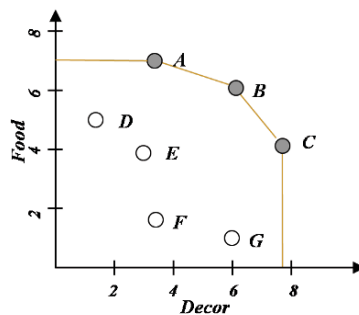


Fig. 24.3: An example of skyline points (the best candidate restaurants) in two-dimensional space

24.5.2.4 Using multi-criteria ratings as recommendation filters

Similar to how content attributes can be used as recommendation filters in recommender systems [45, 82], multi-criteria ratings can be used for similar purposes as well. For example, a user may want to specify that only the movies with an exceptionally good story should be recommended to her at a given time, regardless of other criteria, such as visual effects. Then, only the movies that are highly predicted in the story criterion (say, ≥ 9 out of 10) will be recommended to the user. This approach is similar to how content-based [45, 82] or context-aware [3] recommendation approaches filter recommendations; however, it is also slightly different from

them, because the filtering is done not based on objective content attributes (e.g., $\text{MovieLength} < 120$ minutes) or additional contextual dimensions (e.g., $\text{TimeOfWeek} = \text{weekend}$), but on the subjective rating criteria (e.g., $\text{Story} \geq 9$), the predicted value of which is highly dependent on user's tastes and preferences.

24.6 Discussion and Future Work

Recommender systems represent a vibrant and constantly changing research area. Among the important recent developments, recommender systems have recently started adopting multi-criteria ratings provided by users, and in this chapter we investigated algorithms and techniques for multi-criteria recommender systems. These new systems have not yet been studied extensively, and in this section we present a number of challenges and future research directions for this category of recommender systems.

Managing intrusiveness. The extra information provided by multi-criteria ratings can give rise to an important issue of intrusiveness. For a recommender system to achieve good recommendation performance, users typically need to provide to the system a certain amount of feedback about their preferences (e.g., in the form of item ratings). This can be an issue even in single-rating recommender systems, and some less intrusive techniques to obtain user preferences have been proposed [40, 56, 62]. Multi-criteria rating systems may require a more significant level of user involvement because each user would need to rate an item on multiple criteria. Therefore, it is important to measure the costs and benefits of adopting multi-criteria ratings and find an optimal solution to meet the needs of both users and system designers. Preference disaggregation methods could support the implicit formulation of a preference model based on a series of previous decisions. A characteristic example is the UTA (i.e., UTilités Additive) method, which can be used to extract the utility function from a user-provided ranking of known items [42]. Another example is the ability to obtain each user's preferences on several attributes of an item implicitly from the user's written comments, minimizing intrusiveness [1, 68]. There are also some empirical approaches with less computational complexity [80]. Lastly, performing user studies on multi-criteria recommender systems would further examine the impact of having to submit more ratings on the overall user satisfaction.

Reusing existing single-rating recommendation techniques. A huge number of recommendation techniques have been developed for single-rating recommender systems over the last 10-15 years, and some of them could potentially be extended to multi-criteria rating systems. For example, neighborhood-based collaborative filtering techniques may possibly take into account multi-criteria ratings using the huge number of design options that Manouselis and Costopoulou [51] suggest. As another example, there has been a number of sophisticated hybrid recommendation approaches developed in recent years [11], and some of them could potentially be adopted for multi-criteria rating recommenders. Finally, more sophisticated tech-

niques, e.g., based on data envelopment analysis (DEA) or multi-criteria optimization, could be adopted and extended for choosing best items in the multi-criteria rating settings.

Predicting relative preferences. An alternative way to define the multi-criteria recommendation problem could be formulated as predicting the *relative* preferences of users, as opposed to the *absolute* rating values. There has been some work on constructing the correct relative order of items using ordering-based techniques. For example, Freund et al. [25] developed the RankBoost algorithm based on the well-known AdaBoost method and, in multi-criteria settings, such algorithms could be adopted to aggregate different relative orders obtained from different rating criteria for a particular user. In particular, this is an approach taken by the DIVA system [59, 60].

Constructing the item evaluation criteria. More research needs to be done on choosing or constructing the best set of criteria for evaluating an item. For example, most of current multi-criteria rating recommenders require users to rate an item on multiple criteria at a single level (e.g., story and special effects of a movie). This single level of criteria could be further broken down into sub-criteria, and there could be multiple levels depending on the given problem. For example, in a movie recommender system, special effects could be again divided into sound and graphic effects. More information with multiple levels of criteria could potentially help to better understand user preferences, and various techniques, such as the analytic hierarchy process (AHP), can be used to consider the hierarchy of criteria [78], as Schmitt et al. [85] propose to do in their system. As we consider more criteria for each item, we may also need to carefully examine the correlation among criteria because the choice of criteria may significantly affect the recommendation quality. Furthermore, as mentioned earlier, it is important to have a *consistent* family of criteria for a given recommender system application because then the criteria are monotonic, exhaustive, and non-redundant. In summary, constructing a set of criteria for a given recommendation problem is an interesting and important topic for future research.

Dealing with missing multi-criteria ratings. Multi-criteria recommender systems typically would require the users to provide more data to such systems than their single-rating counterparts, thus increasing the likelihood of obtaining missing or incomplete data. One popular technique to deal with missing data is the expectation maximization (EM) algorithm [18] that finds maximum likelihood estimates for incomplete data. In particular, the probabilistic modeling approach for multi-criteria rating prediction proposed by [79] uses the EM algorithm to predict values of the missing ratings in multi-criteria rating settings. The applicability of other existing techniques in this setting should be explored, and novel techniques could be developed by considering the specifics of multi-criteria information, such as the possible relationships between different criteria.

Investigating group recommendation techniques for multi-criteria settings. Some techniques for generating recommendations to groups can be adopted in

multi-criteria rating settings. According to [31], a group preference model can be built by aggregating the diverse preferences of several users. Similarly, a user's preference for an item in multi-criteria rating settings can be predicted by aggregating the preferences based on different rating criteria. More specifically, there can be many different goals for aggregating individual preferences [55, 63], such as maximizing average user satisfaction, minimizing misery (i.e., high user dissatisfaction), and providing a certain level of fairness (e.g., low variance with the same average user satisfaction). Multi-criteria rating recommenders could investigate the adoption of some of these approaches for aggregating preferences from multiple criteria.

Developing new MCDM modeling approaches. From the MCDM perspective, the recommendation problem is posing novel challenges to the decision modellers. On the one hand, there is a plethora of additional techniques that can be readily adopted and used in such systems, such as including a sensitivity analysis step in the algorithm, as [68] proposes. On the other hand, some studies indicate that recommendation is not a single decision making problem, since there are several decision problems that have to be addressed simultaneously, and each individual has an influence on the recommendation provided to other individuals [54]. Neither it is considered to be a typical group decision making problem nor a negotiation between individuals [66]. Therefore, new MCDM modelling approaches should be proposed and tested for multi-criteria recommendations [20].

Collecting large-scale multi-criteria rating data. Multi-criteria rating datasets that can be used for algorithm testing and parameterization are rare. For this new area of recommender systems to be successful, it is crucial to have a number of standardized real-world multi-criteria rating datasets available to the research community. Some initial steps towards a more standardized representation, reusability, and interoperability of multi-criteria rating datasets have been taken in other application domains, such as e-learning [98].

In this section we discussed several potential future research directions for multi-criteria recommenders that should be interesting to recommender systems community. This list is not meant to be exhaustive; we believe that research in this area is only in its preliminary stages, and there are a number of possible additional topics that could be explored to advance multi-criteria recommender systems.

24.7 Conclusions

In this chapter, we aimed to provide an overview of multi-criteria recommender systems. We first defined the recommendation problem as an MCDM problem and reviewed the MCDM methods and techniques that can support the implementation of multi-criteria recommenders. Then, we focused on the category of *multi-criteria rating recommenders*, i.e., techniques that provide recommendations by modelling a user's utility for an item as a vector of ratings along several criteria. We reviewed current techniques that use multi-criteria ratings for calculating the rating prediction

and generating recommendations, and discussed open issues and future challenges for this class of recommender systems.

This survey provides a systematic view of multi-criteria recommender systems, a roadmap of relevant work, and a discussion of a number of promising future research directions. However, we believe that this is only a first step towards exploring this problem-rich area of recommender systems, and much more research and development are needed to unlock the full potential of multi-criteria recommenders.

Acknowledgements Research of G. Adomavicius was supported in part by the National Science Foundation grant IIS-0546443, USA. Research of N. Manouselis was funded with support by the European Commission and more specifically, the project ECP-2006-EDU-410012 “Organic.Edunet: A Multilingual Federation of Learning Repositories with Quality Content for the Awareness and Education of European Youth about Organic Agriculture and Agroecology” of the eContentplus Programme.

References

1. Aciar, S., Zhang, D., Simoff, S., Debenham, J. Informed Recommender: Basing Recommendations on Consumer Product Reviews. *IEEE Intelligent Systems*, 22(3):39–47, 2007.
2. Adomavicius, G., Kwon, Y. New Recommendation Techniques for Multi-Criteria Rating Systems. *IEEE Intelligent Systems*, 22(3):48–55, 2007.
3. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems (TOIS)*, 23(1):103–145, 2005.
4. Adomavicius, G., Tuzhilin, A. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
5. Ardissono, L., Goy, A., Petrone, G., Segnan, M., Torasso, P. Intrigue: Personalised Recommendation of Tourist Attractions for Desktop and Handset Devices. *Applied Artificial Intelligence*, Special Issue on Artificial Intelligence for Cultural Heritage and Digital Libraries, 17(8):687–714, 2003.
6. Ariely, D., Lynch, J.G.Jr., Aparicio, M. Learning by Collaborative and Individual-based recommendation Agents. *Journal of Consumer Psychology*, 14(1&2):81–95, 2004.
7. Balabanovic, M., Shoham, Y. Fab:Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.
8. Breese, S., Heckerman, D., Kadie, C. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, volume 461, pages 43–52. San Francisco, CA, 1998.
9. Burke, R. Knowledge-Based Recommender Systems. *Encyclopedia of Library and Information Systems*, 69(Supplement 32):175–186, 2000.
10. Burke, R. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.
11. Burke, R. Hybrid Web Recommender Systems. In P. Brusilovsky, A. Kobsa, W., Nejdl (Eds.), *The Adaptive Web: Methods and Strategies of Web, Personalization*, *Lecture Notes in Computer Science*, 4321:377–408, 2007.
12. Cantador, I., Fernandez, M., Castells, P. A Collaborative Recommendation Framework for Ontology Evaluation and Reuse. In *Proc. of the International ECAI Workshop on Recommender Systems*. Riva del Garda, Italy, 2006.

13. Charnes, A., Cooper, W., Rhodes, E. Measuring the efficiency of decision making units. *European Journal of Operational Research*, 2(6):429–444, 1978.
14. Cheetham, W. Global Grade Selector: A Recommender System for Supporting the Sale of Plastic Resin. *Technical Information Series*, GE Global Research, TR 2003GRC261, 2003.
15. Choi, S.H., Cho, Y.H. An utility range-based similar product recommendation algorithm for collaborative companies. *Expert Systems with Applications*, 27(4):549–557, 2004.
16. Chow, C., Liu, C. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.
17. De Lathauwer, L., De Moor, B., Vandewalle, J. A Multilinear Singular Value Decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.
18. Dempster, A., Laird, N., Rubin, D. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.
19. Denguir-Rekik, A., Montmain, J., Mauris, G. A fuzzy-valued Choquet-integral-based multi-criteria decision-making support for marketing and benchmarking activities in e-commerce organizations. In *Proc. of the MCDM 2006*. Chania, Greece, 2006.
20. DIMACS/LAMSADE. Computer science and decision theory: Applications of notions of consensus description of the research issues in the project. In *COST Action ICO602*. <http://dimacs.rutgers.edu/Workshops/Lamsade/propmod.pdf>, 2004.
21. Ehrgott, M. *Multicriteria Optimization*. Springer Verlag, 2005.
22. Emi, Y., Suetoshi, E., Shinohara, I., Toshikazu, K. Development of a recommendation system with multiple subjective evaluation process models. In *Proc. of the 2003 International Conference on Cyberworlds (CW03) 0-7695-1922-9/03*. Singapore, 2003.
23. Falle, W., Stoeffler, D., Russ, C., Zanker, M., Felfernig, A. Using knowledge-based advisor technology for improved customer satisfaction in the shoe industry. In *Proc. of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Technical University of Denmark, Copenhagen, 2004.
24. Figueira, J., Greco, S., Ehrgott, M. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer Verlag, 2005.
25. Freund, Y., Iyer, R., Schapire, R.E., Singer, Y. An efficient boosting algorithm for combining preferences. *The Journal of Machine Learning Research*, 4:933–969, 2003.
26. Funk, S. Netflix update: Try this at home. <http://sifter.org/simon/journal/20061211.html>, 2006.
27. Ghosh, S., Munde, M., Hernandez, K., Sen, S. Voting for movies: the anatomy of a recommender system. In *Proc. of the 3rd Annual Conference on Autonomous Agents*, pages 434–435. ACM, 1999.
28. Guan, S., Ngoo, C.S., Zhu, F. Handy broker: an intelligent product-brokering agent for m-commerce applications with user preference tracking. *Electronic Commerce Research and Applications*, 1(3-4):314–330, 2002.
29. Herrera-Viedma, E., Pasi, G., Lopez-Herrera, A.G. Evaluating the Information Quality of Web Sites: A Qualitative Methodology Based on Fuzzy Computing with Words. *Journal of the American Society for Information Science and Technology*, 57(4):538–549, 2006.
30. Jacquet-Lagrèze, E., Siskos, Y. Preference disaggregation: 20 years of mcda experience. *European Journal of Operational Research*, 130(2):233–245, 2001.
31. Jameson, A., Smyth, B. Recommendation to groups. Brusilovsky P., Kobsa A., and Nejdil W., eds. *Adaptive Web: Methods and Strategies of Web Personalization*, Springer, Berlin, 596–627, 2007.
32. Karacapilidis, N., Hatzieleftheriou, L. A hybrid framework for similarity-based recommendations. *International Journal of Business Intelligence and Data Mining*, 1(1):107–121, 2005.
33. Keeney, R.L. *Value-focused Thinking: A Path to Creative Decisionmaking*. Cambridge MA: Harvard Univ Press, 1992.
34. Keeney, R.L., Raiffa, H. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Cambridge Univ Press, 1976.

35. Kerschberg, L., Kim, W., Scime, A. Websifter /uppercaseII: A Personalizable Meta-Search Agent based on Semantic Weighted Taxonomy Tree. In *Proc. of the International Conference on Internet Computing*, pages 14–20. Las Vegas, Nevada, 2001.
36. Kim, T.H., Yang, S.B. Using Attributes to Improve Prediction Quality in Collaborative Filtering. In *Proc. of the E-Commerce and Web Technologies: 5th International Conference, EC-Web 2004*, Zaragoza, Spain, 2004.
37. Kim, W., Kerschberg, L., Scime, A. Learning for automatic personalization in a semantic taxonomy-based meta-search agent. *Electronic Commerce Research and Applications*, 1(2):150–173, 2002.
38. Kleinberg, J., Sandler, M. Convergent Algorithms for Collaborative Filtering. In *Proc. of the 4th ACM Conference on Electronic Commerce*, pages 1–10. San Diego, CA, 2003.
39. Konstan, J.A. Introduction to Recommender Systems: Algorithms and Evaluation. *ACM Transactions on Information Systems (TOIS)*, 22(1):1–4, 2004.
40. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J. GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40(3):77–87, 1997.
41. Koren, Y. Collaborative Filtering with Temporal Dynamics. In *Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 447–456. ACM New York, NY, USA, 2009.
42. Lakiotaki, K., Tsafarakis, S., Matsatsinis, N. UTA-Rec: A Recommender System Based on Multiple Criteria Analysis. In *Proc. of the 2008 ACM Conference on Recommender Systems*, pages 219–226. ACM New York, NY, USA, 2008.
43. Lee, H.H., Teng, W.G. Incorporating Multi-Criteria Ratings in Recommendation Systems. In *IEEE International Conference on Information Reuse and Integration*, pages 273–278, 2007.
44. Lee, W.P. Towards agent-based decision making in the electronic marketplace: interactive recommendation and automated negotiation. *Expert Systems with Applications*, 27(4):665–679, 2004.
45. Lee, W.P., Liu, C.H., Lu, C.C. Intelligent agent-based systems for personalized recommendations in Internet commerce. *Expert Systems with Applications*, 22(4):275–284, 2002.
46. Li, Q., Wang, C., Geng, G. Improving Personalized Services in Mobile Commerce by a Novel Multicriteria Rating Approach. In *Proc. of the 17th International World Wide Web Conference*. Beijing, China, 2008.
47. Liu, D.R., Shih, Y.Y. Integrating AHP and data mining for product recommendation based on customer lifetime value. *Information & Management*, 42(3):387–400, 2005.
48. Manouselis, N., Costopoulou, C. Analysis and Classification of Multi-Criteria Recommender Systems. *World Wide Web: Internet and Web Information Systems*, 10(4):415–441, 2007.
49. Manouselis, N., Costopoulou, C. Experimental Analysis of Design Choices in Multi-Attribute Utility Collaborative Filtering. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(2):311–332, 2007.
50. Manouselis, N., Costopoulou, C. Towards a Design Process for Intelligent Product Recommendation Services in e-Markets. *Artificial Intelligence and Integrated Information Systems: Emerging Technologies and Applications*, pages 398–417, 2007.
51. Manouselis, N., Costopoulou, C. Overview of design options for neighborhood-based collaborative filtering systems. *Personalized Information Retrieval and Access: Concepts, Methods and Practices*, pages 30–54, 2008.
52. Manouselis, N., Sampson, D. A Multi-criteria Model to Support Automatic Recommendation of e-Learning Quality Approaches. In *Proc. of the 16th World Conference on Educational Multimedia, Hypermedia and Telecommunications (EDMEDIA)*. Lugano, Switzerland, 2004.
53. Masthoff, J. Modeling the Multiple People That Are Me. In *Proc. of the International Conference on User Modelling (UM2003)*, pages 258–262. Johnstown, USA, 2003.
54. Matsatsinis, N.F., Samaras, A.P. MCDA and preference disaggregation in group decision support. *European Journal of Operational Research*, 130(2):414–429, 2001.

55. McCarthy, J. Pocket RestaurantFinder: A situated recommender system for groups. In *Proc. of the Workshop on Mobile Ad-Hoc Communication at the 2002 ACM Conference on Human Factors in Computer Systems*. Minneapolis, MN, 2002.
56. Middleton, S.E., Shadbolt, N.R., Roure, D.C. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):54–88, 2004.
57. Montaner, M., Lopez, B., de la Rosa, J.L., Opinion-Based Filtering Through Trust. In *Proc. of the 6th International Workshop on Cooperative Information Agents VI*, LNCS 2446, pages 164–178, Springer-Verlag, 2002.
58. Mukherjee, R., Dutta, P.S., Jonsdottir, G., Sen, S MOVIES2GO: An Online Voting Based Movie Recommender System. In *Proc. of the 5th International Conference on Autonomous Agents*, pages 114–115. Montreal, Canada, 2001.
59. Nguyen, H., Haddawy, P. DIVA: Applying Decision Theory to Collaborative Filtering. In *Proc. of the AAAI Workshop on Recommender Systems*. Madison, WI, 1998.
60. Nguyen, H., Haddawy, P. The Decision-Theoretic Video Advisor In *Proc. of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 494–501. Stockholm, Sweden, 1999.
61. Noh, S. Implementing Purchasing Assistant Using Personal Profile. In *Proc. IADIS International Conference on Applied Computing*. Lisbon, Portugal, 2004.
62. Oard, D., Kim, J. Modeling information content using observable behavior. In *Proc. of the Annual Meeting-American Society for Information Science*, volume 38, pages 481–488. Washington DC., 2001.
63. O'Connor, M., Cosley, D., Konstan, J.A., Riedl, J. PolyLens: A Recommender System for Groups of Users. In *Proc. of the 7th European Conference on Computer Supported Cooperative Work*, pages 199–218. Kluwer Academic Publishers, 2001.
64. Pardalos, P., Siskos, Y., Zopounidis, C *Advances in Multicriteria Analysis*. Dordrecht: Kluwer Academia Publishers, 1995.
65. Pazzani, M., Billsus, D. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, 27(3):313–331, 1997.
66. Perny, P., Zucker, J.D. Collaborative Filtering Methods based on Fuzzy Preference Relations. In *Proc. of EUROFUSE-SIC 99*, pages 279–285, 1999.
67. Perny, P., Zucker, J.D. Preference-based Search and Machine Learning for Collaborative Filtering: the Film-Conseil Movie Recommender System. *Information, Interaction, Intelligence*, 1(1):1–40, 2001.
68. Plantie, M., Montmain, J., Dray, G. Movies Recommenders Systems: Automation of the Information and Evaluation Phases in a Multi-criteria Decision-Making Process. In *Proc. of the DEXA 2005*, LNCS 3588, pages 633–644. Berlin Heidelberg: Springer-Verlag, 2005.
69. Price, B, Messinger, P.R. Optimal Recommendation Sets: Covering Uncertainty over User Preferences. *Proc. of the National Conference on Artificial Intelligence*, 2005.
70. Reilly, J., McCarthy, K., McGinty, L., Smyth, B. Incremental Critiquing. *Knowledge-Based Systems*, 18(4-5):143–151, 2005.
71. Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., Riedl, J. GroupLens: An open architecture for collaborative filtering of netnews. In *Proc. of the 1994 ACM Conference on Computer Supported Cooperative Work*, pages 175–186, 1994.
72. Resnick, P., Varian, H.R. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997.
73. Ricci, F., Arslan, B., Mirzadeh, N., Venturini, A. ITR: A Case-Based Travel Advisory System. *Advances in Case-Based Reasoning*, LNAI 2416, pages 613–627. Springer-Verlag: Berlin Heidelberg, 2002.
74. Ricci, F., Nguyen, Q.N. Acquiring and Revising Preferences in a Critique-Based Mobile Recommender System. *IEEE Intelligent Systems*, 22(3):22–29, 2007.
75. Ricci, F., Venturini, A., Cavada, D., Mirzadeh, N., Blaas, D., Nones, M. Product Recommendation with Interactive Query Management and Twofold Similarity. *Proc. of the 5th International Conference on Case-Based Reasoning*, Trondheim, Norway, 2003.

76. Roy, B., Bouyssou, D. *Aide multicritère à la décision: méthodes et cas*. Economica Paris, 1993.
77. Roy, B., McCord, M.R. *Multicriteria Methodology for Decision Aiding*. Springer, 1996.
78. Saaty, T.L. *Optimization in Integers and Related Extremal Problems*. McGraw-Hill, 1970.
79. Sahoo, N., Krishnan, R., Duncan, G., Callan, J.P. Collaborative Filtering with Multi-component Rating for Recommender Systems. In *Proc. of the 16th Workshop on Information Technologies and Systems*. Milwaukee, WI, 2006.
80. Sampaio, I., Ramalho, G., Corruble, V., Prudencio, R. Acquiring the Preferences of New Users in Recommender Systems: The Role of Item Controversy. In *Proc. of the 17th European Conference on Artificial Intelligence (ECAI) Workshop on Recommender Systems*, pages 107–110. Riva del Garda, Italy, 2006.
81. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J. Application of Dimensionality Reduction in Recommender System - A Case Study. In *Proc. of the Workshop on Knowledge Discovery in the Web (WebKDD)*, 2000.
82. Schafer, J.B. DynamicLens: A Dynamic User-Interface for a Meta-Recommendation Systems. In *Proc. of the Workshop on the Next Stage of Recommender Systems Research at the ACM Intelligent User Interfaces Conference*, 2005.
83. Schickel-Zuber, V., Faltings, B. Heterogeneous Attribute Utility Model: A new approach for modeling user profiles for recommendation systems. In *Proc. of the Workshop on Knowledge Discovery in the Web (WebKDD)*. Chicago, Illinois, 2005.
84. Schmitt, C., Dengler, D., Bauer, M. The MAUT-Machine: An Adaptive Recommender System. In *Proc. of the Workshop on Adaptivitat und Benutzermodellierung in Interaktiven Softwaresystemen (ABIS)*. Hannover, Germany, 2002.
85. Schmitt, C., Dengler, D., Bauer, M. Multivariate Preference Models and Decision Making with the MAUT Machine. In *Proc. of the 9th International Conference on User Modeling (UM 2003)*, pages 297–302, 2003.
86. Si, L., Jin, R. Flexible Mixture Model for Collaborative Filtering. In *Proc. of the 20th International Conference on Machine Learning*, volume 20, pages 704–711. AAAI Press, 2003.
87. Siskos, Y., Grigoroudis, E., Matsatsinis, N.F. *UTA Methods*. Springer, 2005.
88. Smyth, B. Case-Based Recommendation. *The Adaptive Web: Methods and Strategies of Web Personalization*, 2007.
89. Srikumar, K., Bhasker, B. Personalized Product Selection in Internet Business. *Journal of Electronic Commerce Research*, 5(4):216–227, 2004.
90. Stolze, M., Rjaibi, W. Towards Scalable Scoring for Preference-based Item Recommendation. *IEEE Data Engineering Bulletin*, 24(3):42–49, 2001.
91. Stolze, M., Stroebel, M. Dealing with Learning in eCommerce Product Navigation and Decision Support: The Teaching Salesman Problem. In *Proc. of the 2nd Interdisciplinary World Congress on Mass Customization and Personalization*. Munich, Germany, 2003.
92. Tang T.Y., McCalla, G. The Pedagogical Value of Papers: a Collaborative-Filtering based Paper Recommender. *Journal of Digital Information*, 10(2), 2009.
93. Tewari, G., Youll, J., Maes, P. Personalized location-based brokering using an agent-based intermediary architecture. *Decision Support Systems*, 34(2):127–137, 2003.
94. Thorndike, E.L. A Constant Error in Psychological Ratings. *Journal of Applied Psychology*, 4(1):25–29, 1920.
95. Triantaphyllou, E. *Multi-Criteria Decision Making Methods: A Comparative Study*. Kluwer Academic Pub, 2000.
96. Tsai, K.H., Chiu, T.K., Lee, M.C., Wang, T.I. A Learning Objects Recommendation Model based on the Preference and Ontological Approaches. In *Proc. of the 6th IEEE International Conference on Advanced Learning Technologies (ICALT'06)*, 2006.
97. Vincke, P. *Multicriteria Decision-Aid*. New York: J. Wiley, 1992.
98. Vuorikari, R., Manouselis, N., Duval, E. Using Metadata for Storing, Sharing, and Reusing Evaluations in Social Recommendation: the Case of Learning Resources. *Social Information Retrieval Systems: Emerging Technologies and Applications for Searching the Web Ef-*

- fectively*, in Go D.H. & Foo S. (Eds.), pages 87–107. Hershey, PA: Idea Group Publishing, 2008.
99. Wang, P. Recommendation Based on Personal Preference. *Computational Web Intelligence: Intelligent Technology for Web Applications*, in Zhang Y., Kandel A., Lin T., and Yao Y.(Eds.), World Scientific Publishing Company, 2004.
 100. Yu, C.C. Designing a Web-Based Consumer Decision Support Systems for Tourism Services. In *Proc. of the 4th International Conference on Electronic Commerce*, pages 23–25. Hong Kong, 2002.
 101. Yu, C.C. Innovation, management and strategy: A web-based consumer-oriented intelligent decision support system for personalized e-service. In *Proc. of the 6th International Conference on Electronic Commerce*, pages 429–437, 2004.
 102. Zeleny, M. *Linear Multiobjective Programming*. New York: Springer, 1974.
 103. Zimmerman, J., Kurapati, K., Buczak, A.L., Schaffer, D., Martino, J., Gutta, S. TV Personalization System: Design of a TV Show Recommender Engine and Interface. *Personalized Digital Television: Targeting Programs to Individual Viewers*, in Ardissono L., Kobsa A., Maybury M. (Eds.), Human-Computer Interaction Series 6, 2004.

Chapter 25

Robust Collaborative Recommendation

Robin Burke, Michael P. O’Mahony and Neil J. Hurley

Abstract Collaborative recommender systems are vulnerable to malicious users who seek to bias their output, causing them to recommend (or not recommend) particular items. This problem has been an active research topic since 2002. Researchers have found that the most widely-studied memory-based algorithms have significant vulnerabilities to attacks that can be fairly easily mounted. This chapter discusses these findings and the responses that have been investigated, especially detection of attack profiles and the implementation of robust recommendation algorithms.

25.1 Introduction

Collaborative recommender systems are dependent on the goodwill of their users. There is an implicit assumption – note the word “collaborative” – that users are in some sense “on the same side”, and at the very least, that they will interact with the system with the aim of getting good recommendations for themselves while providing useful data for their neighbors. Herlocker et al. [10] use the analogy of the “water-cooler chat”, whereby co-workers exchange tips and opinions.

However, as contemporary experience has shown, the Internet is not solely inhabited by good-natured collaborative types. Users will have a range of purposes

Robin Burke

Center for Web Intelligence, School of Computer Science, Telecommunication and Information Systems, DePaul University, Chicago, Illinois, USA e-mail: rburke@cs.depaul.edu

Michael P. O’Mahony

CLARITY: Centre for Sensor Web Technologies, School of Computer Science and Informatics, University College Dublin, Ireland e-mail: michael.p.omahony@ucd.ie

Neil J. Hurley

School of Computer Science and Informatics, University College Dublin, Ireland e-mail: neil.hurley@ucd.ie

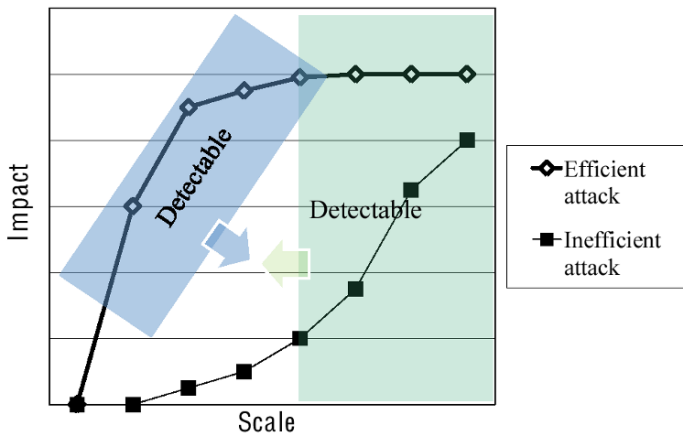


Fig. 25.1: Curves show the theoretical impact of attacks of different degrees of efficiency. The shaded areas shows attacks that can be detected.

in interacting with recommender systems, and in some cases, those purposes may be counter to those of the system owner or those of the majority of its user population. To cite a well-known example, the Google search engine finds itself engaging in more-or-less continual combat against those who seek to promote their sites by “gaming” its retrieval algorithm.

In search engine spam, the goal for an attacker is to make the promoted page “look like” a good answer to a query in all respects that Google cares about. In the case of collaborative recommendation, the goal for an adversary is to make a particular product or item look like a good recommendation for a particular user (or maybe all users) when really it is not. Alternatively, the attacker might seek to prevent a particular product from being recommended when really it is a good choice. If we assume that a collaborative system makes its recommendations purely on the basis of user profiles, then it is clear what an attacker must do – add user profiles that push the recommendation algorithm to produce the desired effect. A single profile would rarely have this effect, and in any case, fielded systems tend to avoid making predictions based on only a single neighbor. What an attacker really needs to do is to create a large number of pseudonomous profiles designed to bias the system’s predictions. Site owners try to make this relatively costly, but there is an inherent tension between policing the input of a collaborative system and making sure that users are not discouraged from entering the data that the algorithm needs to do its work. The possibility of designing user rating profiles to deliberately manipulate the recommendation output of a collaborative filtering system was first raised in [24]. Since then, research has focused on attack strategies, detection strategies to combat attacks and recommendation algorithms that have inherent robustness against attack.

A framework for understanding this research is sketched in Figure 25.1. First, we demonstrate the extent of the problem by modeling *efficient* attacks, attacks that can

with relatively low cost produce a large impact on system output. This enables us to understand the shape of the impact curve for efficient attacks. Research on detection attempts to identify groups of profiles that make up an attack and to eliminate them from the database. Attacks that are not efficient are more difficult to detect, but because they are inefficient, must be very large to have an impact. A large influx of ratings for a particular item is easy to detect with standard system monitoring procedures. Research on detection therefore focuses on how to detect efficient attacks and variants of them, seeking to increase the size of the “detectable” boxes in the diagram, and thereby limiting the impact that an attacker can have. At the same time, researchers have studied a number of algorithms that are intended to be robust against attack, having lower impact curves relative to efficient attacks. With the combination of these techniques, researchers have sought, not to eliminate attacks, but to control their impact to the point where they are no longer cost-effective.

This chapter looks at each of these points in turn. In Section 25.3, we look at research that aims to identify the most efficient and practical attacks against collaborative recommender systems, establishing the shape of the impact curve suggested above. Section 25.5 looks at the problem of detection: in particular, the left-most shaded area for detecting efficient attacks. Lastly, in Section 25.6, we examine attempts to reduce the impact of attacks through robust algorithms.

25.2 Defining the Problem

A collaborative recommender is supposed to change its recommendations in response to the profiles that users add. It is somewhat counter-intuitive to suppose that “robustness” or “stability” is a desirable property in a system that is supposed to be adaptive. The goal of robust recommendation is to prevent attackers from manipulating the system through large-scale insertion of user profiles, a *profile injection* attack.

We assume that any user profile is feasible. That is, we do not want to demand that users’ ratings fit with those that have been entered previously or that they make any kind of objective sense. Users are entitled to their idiosyncratic opinions and there is always the possibility that what is an unusual user today may be more typical tomorrow as new users sign up. So, a profile, taken by itself, cannot constitute an attack. Also, it is important to note that some web phenomena that look like attacks are not considered such within this definition. For example, in the Fall of 2008, numerous videogame fans converged on the page for the game *Spore* on Amazon.com, using it as a vehicle for airing their complaints about the digital rights management software included with the game. Presumably these were a large number of authentic individuals, and while their ratings no doubt skewed the recommendations for *Spore* for some time, their actions would not be considered an attack as we define it here. It is not clear that any automated technique can identify when a real user posts

a rating to make a political statement or as a prank, rather than to reflect an honest preference.¹

For the purposes of this research, an attack is a concerted effort to bias the results of a recommender system by the insertion of a large number of profiles using false identities. Each of the separate identities assumed by the attacker are referred to as an *attack profile*. Once created, these profiles are used to insert preference data into the system. The most dangerous attacks are those that are crafted for maximum impact on the system, so much research has been devoted to finding the most effective and practical attacks against different algorithms.

While random vandalism surely does occur, research in this area has concentrated on attacks designed to achieve a particular recommendation outcome. The objectives of *product push* and *product nuke* attacks are to promote or demote the recommendations made for items, respectively. For example, the goal of an attacker might be to force a system to output poor recommendations for his competitors' products (nuke) while attempting to secure positive recommendations for his own (push).

From the perspective of the attacker, the best attack against a system is one that yields the biggest impact for the least amount of effort. There are two types of effort involved in mounting an attack. The first is the effort involved in crafting profiles. One of the crucial variables here is the amount of knowledge that is required to put together an attack. A *high-knowledge attack* is one that requires the attacker to have detailed knowledge of the ratings distribution in a recommender system's database. Some attacks, for example, require that the attacker know the mean rating and standard deviation for every item. A *low-knowledge attack* is one that requires system-independent knowledge such as might be obtained by consulting public information sources.

We assume that the attacker will have a general knowledge of the type of algorithm being employed to produce recommendations. An attacker that has more detailed knowledge of the precise algorithm in use would be able to produce an *informed attack* that makes use of the mathematical properties of the algorithm itself to produce the greatest impact.

The second aspect of effort is the number of profiles that must be added to the system in order for it to be effective. The ratings are less important since the insertion of ratings can be easily automated. Most sites employ online registration schemes requiring human intervention, and by this means, the site owner can impose a cost on the creation of new profiles. This is precisely why, from an attacker's perspective, attacks requiring a smaller number of profiles are particularly attractive.

¹ It could be argued that even such a technique did exist, it would not be in the interest of a collaborative system to deploy it.

		Items						
		1	2	3	4	5	6	7
Users	<i>a</i>	+	-		+	+		+
	<i>b</i>	-	+	+	-	-		-
	<i>c</i>	+	-	+		-	-	-
	<i>d</i>	-	+	+	-			
	<i>e</i>	-		-	-	-		-
	<i>f</i>	+	-	+	+	+		+
	<i>g</i>		-	+	+	-	-	+
	<i>h</i>	+	-	+	+	+		?
	<i>i</i>	+	-	+		-	-	-
	<i>j</i>	-	+	+	-			-
	<i>k</i>	-		-	-	-		-
	<i>l</i>	+	-	+	+	+		-
	<i>m</i>		-	+	+	-	-	-

Authentic profiles

Target profile

Attack profiles

Fig. 25.2: Simplified system database showing authentic user profiles and a number of attack profiles inserted. In this example, user *h* is seeking a prediction for item 7, which is the subject of a product nuke attack.

25.2.1 An Example Attack

To illustrate the basic idea of a profile injection attack, consider the simplified recommender system database that is presented in Figure 25.2. In this example, the objective is to demote the recommendations that are made for item 7 (i.e. a product nuke attack), and a number of attack profiles (users *i* through *m*) have been inserted into the system to target this item.

In particular, consider the binary recommendation problem in which the task is to predict whether or not user *h* likes item 7. In the first instance, let the attack profiles be ignored and consider only the authentic profiles (users *a* through *g*) as possible neighbours for the target user, *h*. Regardless of the specific recommendation algorithm used, presumably the algorithm would determine that users *a* and *f* have similar tastes to the active user, and since both of these users like item 7, a positive recommendation for the item follows.

When the attack profiles are also considered as possible neighbours, the situation is significantly altered. Several of these attack profiles are also similar to user *h*, and, since all of these profiles rate item 7 poorly, the system is now likely to recommend a negative rating for the item. Thus, the objective of the attack is realised. The next section discusses how these attack profiles must be crafted to work well in a realistic setting.

25.3 Characterising Attacks

A profile-injection attack against a recommender system consists of a set of profiles added to the system by the attacker. A profile consists of a set of rating/item pairs, or alternately, we can think of the profile being a vector of all items, with a rating value for each item, but allowing the *null* value for unrated items. For the attacks that we are discussing, there will always be a target item i_t that the attacker is interested in promoting or demoting. There will generally also be a set of *filler items*, that are chosen randomly from those available. We will denote this set I_F . Some attack models also make use of a set of items that are selected out of the database. The small set usually has some association with the target item (or a targeted segment of users). For some attacks, this set is empty. This will be the set I_S . Finally, for completeness, the set I_\emptyset contains those items not rated in the profile. Since the selected item set is usually small, the size of each profile (total number of ratings) is determined mostly by the size of the filler item set. Some of the experimental results report filler size as a proportion of the size of I (i.e., the set of all items).

25.3.1 Basic Attacks

Two basic attack models, introduced originally in [12], are the random and average attack models. Both of these attacks involve the generation of profiles using randomly assigned ratings to the filler items in the profile.

25.3.1.1 Random Attack

Random attack profiles consist of random ratings distributed around the overall mean assigned to the filler items and a prespecified rating assigned to the target item. In this attack model, the set of selected items is empty. The target item i_t is assigned the maximum rating (r_{max}) or the minimum rating (r_{min}) in the case of push or nuke attacks, respectively.

The knowledge required to mount such an attack is quite minimal, especially since the overall rating mean in many systems can be determined by an outsider empirically (or, indeed, may be available directly from the system). However, this attack is not particularly effective [12, 6].

25.3.1.2 Average Attack

A more powerful attack described in [12] uses the individual mean for each item rather than the global mean (except for the pushed item). In the average attack, each assigned rating for a filler item corresponds (either exactly or approximately) to the mean rating for that item, across the users in the database who have rated it.

As in the random attack, this attack can also be used as a nuke attack by using r_{min} instead of r_{max} . It should also be noted that the only difference between the average attack and the random attack is in the manner in which ratings are computed for the filler items in the profile.

The average attack might be considered to have considerable knowledge cost of order $|I_F|$ (the number of filler items in the attack profile) because the mean and standard deviation of these items must be known. Experiments, however, have shown that the average attack can be just as successful even when using a small filler item set. Thus the knowledge requirements for this attack can be substantially reduced, but at the cost of making all profiles contain the same items, possibly rendering them conspicuous [4].

25.3.2 Low-knowledge attacks

The average attack requires a relatively high degree of system-specific knowledge on the part of attackers. A reasonable defense against such attacks would be to make it very difficult for an attacker to accumulate the required distribution data. The next set of attack types are those for which the knowledge requirements are much lower.

25.3.2.1 Bandwagon Attack

The goal of the bandwagon attack is to associate the attacked item with a small number of frequently rated items. This attack takes advantage of the Zipf's distribution of popularity in consumer markets: a small number of items, bestseller books for example, will receive the lion's share of attention and also ratings. The attacker using this model will build attack profiles containing those items that have high visibility. Such profiles will have a good probability of being similar to a large number of users, since the high visibility items are those that many users have rated. It does not require any system-specific data, because it is usually not difficult to independently determine what the "blockbuster" items are in any product space.

The bandwagon attack uses selected items which are likely to have been rated by a large number of users in the database. These items are assigned the maximum rating value together with the target item i_t . The ratings for the filler items are determined randomly in a similar manner as in the random attack. The bandwagon attack therefore can be viewed as an extension of the random attack.

As we show in Section 25.4, the bandwagon attack is nearly as effective as the average attack against user-based collaborative filtering algorithms², but without the knowledge requirements of that attack. Thus it is more practical to mount. However, as in the case of the average attack, it falls short when used against an item-based algorithm [12].

² Refer to Chapter 4 for details on the user-based and item-based collaborative filtering algorithms.

25.3.2.2 Segment Attack

Mobasher et al. [19] introduced the segment attack and demonstrated its effectiveness against the item-based algorithm. The basic idea behind the segment attack is to push an item to a targeted group of users with known or easily predicted preferences. For example, the producer of a horror movie might want to get the movie recommended to viewers who have liked other horror movies. In fact, the producer might prefer not to have his movie recommender to viewer who do not enjoy the horror genre, since these users might complain and thereby reveal his attack.

To mount this attack, the attacker determines a set of segment items that are likely to be preferred by his intended target audience. Like the bandwagon attack, it is usually fairly easy to predict what the most popular items in a user segment would be. These items are assigned the maximum rating value together with the target item. To provide the maximum impact on the item-based algorithm, the minimum rating is given to the filler items, thus maximising the variations of item similarities.

25.3.3 Nuke Attack Models

All of the attack models described above can also be used for nuking a target item. For example, as noted earlier, in the case of the random and average attack models, this can be accomplished by associating rating r_{min} with the target item instead of r_{max} . However, the results presented in Section 25.4 suggest that attack models that are effective for pushing items are not necessarily as effective for nuke attacks. Thus, researchers have designed additional attack models designed particularly for nuking items.

25.3.3.1 Love/Hate Attack

The love/hate attack is a very simple attack, with no knowledge requirements. The attack consists of attack profiles in which the target item is given the minimum rating value, r_{min} , while other ratings in the filler item set are the maximum rating value, r_{max} . This can be seen as a very low-knowledge version of the Popular Attack below. Surprisingly, this is one of the most effective nuke attacks against the user-based algorithm.

25.3.3.2 Reverse Bandwagon Attack

The reverse bandwagon attack is a variation of the bandwagon attack, discussed above, in which the selected items are those that tend to be rated poorly by many users. These items are assigned low ratings together with the target item. Thus the target item is associated with widely disliked items, increasing the probability that

the system will generate low predicted ratings for that item. This attack was designed to reduce the knowledge required by selecting only a handful of known disliked items. For example, in the movie domain, these may be box office flops that had been highly promoted prior to their openings.

In Section 25.4, we show that although this attack is not as effective as the more knowledge-intensive average attack for nuking items in the user-based system, it is a very effective nuke attack against item-based recommender systems.

25.3.4 *Informed Attack Models*

The low-knowledge attacks above work by approximating the average attack, concentrating on items that are expected to be rated because of their popularity. The average attack in turn is a natural choice for an attacker with a basic intuition about collaborative recommendation, namely that users will be compared on the basis of similarity, so the incentive is to make the profiles similar to the average user. If, on the other hand, the attacker has more detailed knowledge of the precise algorithm, a more powerful attack can be mounted.

25.3.4.1 Popular Attack

Let us assume that the recommender system uses the widely studied user-based algorithm proposed in [27], where similarities between users are calculated using Pearson correlation³. In a similar manner to the bandwagon attack, attack profiles are constructed using popular (i.e. frequently rated) items from the domain under attack.

A high degree of overlap does not, however, guarantee high similarities between attack and authentic profiles. The bandwagon attack used random filler items to generate variation among ratings with the aim of producing at least some profiles that correlate correctly with any given user. The Popular Attack makes use of average rating data and rates the filler items either $r_{min} + 1$ and r_{min} , according to whether the average rating for the item is higher or lower. Linking the rating value to the average rating is likely to result in positive correlations between attack and authentic profiles and furthermore also maximises the prediction shift (see Section 25.4) of attack profiles as computed by the algorithm under consideration (see [25] for details).⁴

³ See [25] for a discussion on informed attacks in cases where alternative similarity metrics are employed. Note that none of the metrics considered provided robustness against attack.

⁴ Note that an optimal push attack strategy is also presented in [18]. In this case, it is concluded that maximising the correlation between authentic and attack profiles is the primary objective. While this conclusion makes sense, it is important to select attack profile ratings that also maximise prediction shift, as is the case with the popular attack described here.

The ratings strategy described above applies to push attacks; this strategy can easily be adjusted for nuke attacks. For example, positive correlations but negative prediction shifts can be achieved by assigning the target item a rating of r_{min} , and ratings of r_{max} and $r_{max} - 1$ to the more- and less-liked selected items.

The knowledge requirement here is intermediate between the bandwagon attack and the average attack. Like the bandwagon attack, the popular items can usually be easily estimated from outside the system; but because there are no filler items, the Popular Attack will need more popular items. The attacker then needs to guess at the relative average preferences between these items in order to provide the correct rating. It might be possible to extract such distinctions from the system itself, or if not, to mine them from external sources; for example, by counting the number of positive and negative reviews for particular items to find general trends.

25.3.4.2 Probe Attack Strategy

Although there are no studies that look at the detectability of the popular attack, it seems likely that it would be easy to detect since all of the attack profiles are identical and also because in many rating databases $r_{min} + 1$ and r_{min} ratings are relatively rare.

A less conspicuous strategy is to obtain items and their ratings from the system itself via the Probe Attack. To perform this strategy, the attacker creates a seed profile and then uses it to generate recommendations from the system. These recommendations are generated by the neighboring users and so they are guaranteed to be rated by at least some of these users and the predicted ratings will be well-correlated with these users' opinions. One could imagine probing narrowly in order to influence a small group as in the segment attack, or probing more broadly to construct an average attack. In a sense, the probe attack provides a way for the attacker to incrementally learn about the system's rating distribution.

This strategy also has another advantage over the popular attack, since less domain knowledge is required by an attacker. Only a small number of seed items need to be selected by the attacker, thereafter the recommender system is used to identify additional items and ratings. In the experiments conducted in Section 25.4, seed items are selected and assigned ratings in a similar manner as in the popular attack.

25.4 Measuring Robustness

Collaborative recommendation algorithms can be categorised into two general classes, which are commonly referred to as memory-based and model-based algorithms [2]. Memory-based algorithms utilise all available data from a system database to compute predictions and recommendations. In contrast, model-based algorithms operate by first deriving a model from the system data, and this model is subsequently used in the recommendation process.

A wide range of collaborative recommendation algorithms have been proposed in the literature, and a comprehensive analysis of the robustness of all of these algorithms is beyond the scope of this chapter. Here, we focus on two widely implemented and studied algorithms, the *user-based* and *item-based* algorithms [27, 32]. The reader is referred to [21, 20, 31] for a robustness analysis of some other collaborative recommendation algorithms.

25.4.1 Evaluation Metrics

Since the objective of push and nuke attacks is to promote and demote target items, we need to evaluate how successfully they do so. Evaluation metrics for robustness need to capture the differences in the predicted ratings and recommended status (i.e. whether or not the target item is included in a top N recommended list) of target items pre- and post-attack.

Many researchers have used average prediction shift to evaluate the changes in predicted ratings. Let U_T and I_T be the sets of users and items, respectively, in the test data. For each user-item pair (u, i) , the prediction shift denoted by $\Delta_{u,i}$ can be measured as $\Delta_{u,i} = p'_{u,i} - p_{u,i}$, where p and p' are the pre- and post-attack predictions, respectively. A positive value means, for example, that the attack has succeeded in making a pushed item more positively rated. The average prediction shift for an item i over all users can be computed as $\Delta_i = \sum_{u \in U_T} \Delta_{u,i} / |U_T|$. Similarly the average prediction shift for all items tested can be computed as $\bar{\Delta} = \sum_{i \in I_T} \Delta_i / |I_T|$.

Prediction shift is a good indicator that an attack is having the desired effect of making a pushed (or nuked) item appear more (or less) desirable. However, it is possible that a pushed item, for example, could be strongly shifted on average but still not make it onto a recommendation list. Such a situation could arise if the item's initial average prediction is so low that even a strong boost is insufficient. To capture the impact of an attack on prediction lists, another metric has been proposed: hit ratio. Let R_u be the set of top N recommendations for user u . If the target item appears in R_u , for user u , the scoring function H_{ui} has value 1; otherwise it is zero. Hit ratio for an item i is given by $HitRatio_i = \sum_{u \in U_T} H_{ui} / |U_T|$. Average hit ratio can then be calculated as the sum of the hit ratio for each item i following an attack on i across all items divided by the number of items: $\overline{HitRatio} = \sum_{i \in I_T} HitRatio_i / |I_T|$.

Many experimenters make use of the publicly available MovieLens 100K dataset⁵. This dataset consists of 100,000 ratings made by 943 users on 1,682 movies. Ratings are expressed on an integer rating scale of 1 to 5 (the higher the score, the more liked an item is). Results below should be assumed to be relative to this dataset unless otherwise stated.

⁵ <http://www.cs.umn.edu/research/GroupLens/data/>.

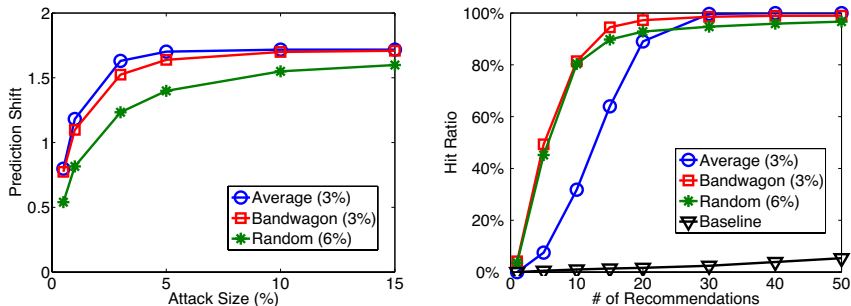


Fig. 25.3: Prediction shift (left) and hit ratio (right) for product push attacks mounted against the user-based collaborative recommendation algorithm. Hit ratio results relate to a 10% attack size.

25.4.2 Push Attacks

To get a sense for the impact that a push attack can have, we will look at results originally reported in [20]. In these figures, the user-based algorithm is subjected to various attacks of different sizes (attack size is measured as a percentage of the total number of authentic profiles in the system; thus an attack of 1% equates to the insertion of 10 attack profiles into the MovieLens dataset). Figure 25.3 (left) shows the average attack (3% filler size), the bandwagon attack (using one frequently rated item and 3% filler size), and the random attack (6% filler size). These parameters were selected as they are the versions of each attack that were found to be most effective. Not surprisingly, the most knowledge-intensive average attack achieved the best performance in terms of prediction shift. This attack works very well. It is capable of moving an average-rated movie (3.6 is the mean) to the top of the five point scale. The performance of the bandwagon attack was quite comparable, despite having a minimal knowledge requirement. In addition, the bandwagon attack was clearly superior to the random attack, which highlights the significance of including the selected items that are likely to be rated by many users.

Interestingly, Figure 25.3 (right) shows that the largest hit ratios were achieved by the bandwagon attack, indicating that prediction shift does not necessarily translate directly into top N recommendation performance. This result is particularly encouraging from the attacker's perspective, given that the required knowledge to implement such attacks is low. Note that all attacks significantly outperform the pre-attack hit ratio results (indicated by "baseline" in the figure).

The item-based algorithm was shown in [12] to be relatively robust against the average attack. The segment attack, introduced in [19], was specifically crafted as a limited-knowledge attack for the item-based algorithm. It aims to increase the column-by-column similarity of the target item with the users preferred items. If the

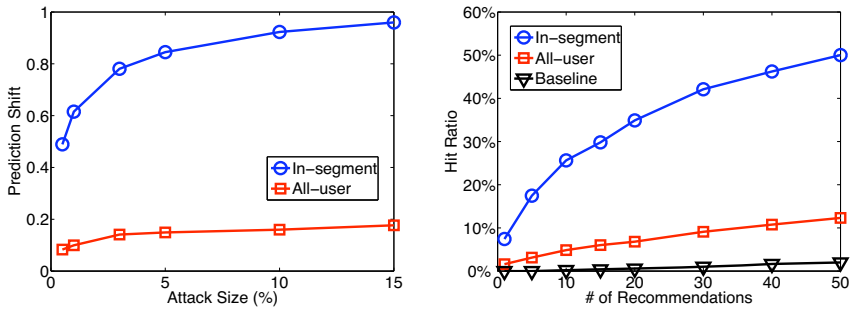


Fig. 25.4: Prediction shift (left) and hit ratio (right) for product push attacks mounted against the item-based collaborative recommendation algorithm. Hit ratio results relate to a 10% attack size.

target item is considered similar to something that the user likes, then its predicted rating will be high – the goal of the push attack. The task therefore for the attacker is to associate her product with popular items that are considered to be similar. The users who have a preference for these similar items are considered the target segment. The task for the attacker in crafting a segment attack is therefore to select items similar to the target item for use as the segment portion of the attack profile I_S . In the realm of movies, we might imagine selecting films of a similar genre or those containing the same actors.

In [19], user segments are constructed by looking at popular actors and genres. For the results shown in Figure 25.4, the segment is all users who gave above average ratings (4 or 5) to any three of the five selected horror movies, namely, *Alien*, *Psycho*, *The Shining*, *Jaws*, and *The Birds*. For this set of five movies, the researchers selected all combinations of three movies that had at least 50 users support, and chose 50 of those users randomly and averaged the results.

The power of the segmented attack is demonstrated in the figure, which contrasts the horror movie fans against the set of all users. While the segmented attack shows some impact against all users, it is clearly very successful in pushing the attacked movie precisely to those users defined by the segment. Further, in the context of the item-based algorithm, the performance of this attack compares very favourably to that of the high-knowledge average attack. For example, the average attack achieved a hit ratio of 30% against all users for top N lists of size 10 and an attack size of 10%. In contrast, the segmented attack achieved approximately the same hit ratio for the same size top N list, but using an attack size of only 1%.

It should also be noted that, although designed specifically as an attack against the item-based algorithm, the segment attack is also effective against the user-based algorithm. Due to limitations of space, we do not show these results here – refer to [20] for details.

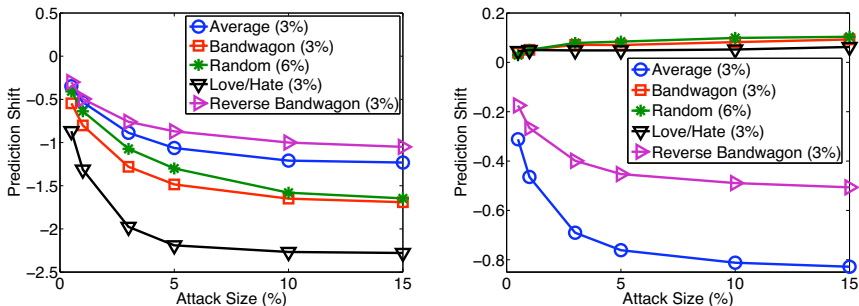


Fig. 25.5: Prediction shifts achieved by nuke attacks against the user-based (left) and item-based (right) algorithms.

25.4.3 Nuke Attacks

It might be assumed that nuke attacks would be symmetric to push attacks, with the only difference being the rating given to the target item and hence the direction of the impact on predicted ratings. However, our results show that there are some interesting differences in the effectiveness of models depending on whether they are being used to push or nuke an item. In particular, the rating distribution should be taken into account: there are in general relatively few low ratings in the MovieLens database, so low ratings can have a big impact on predictions. Furthermore, if we look at the top N recommendations, the baseline (the rate at which an average movie makes it into a recommendation list) is quite low, less than 0.1 even at a list size of 50. It does not take much to make an item unlikely to be recommended.

In the love/hate attack, the randomly selected 3% of filler items were assigned the maximum rating while the target item was given the minimum rating. For the reverse bandwagon attack (designed to attack the item-based algorithm), items with the lowest average ratings that meet a minimum threshold in terms of the number of user ratings in the system are selected as the selected item set, as described in detail in Section 25.3. The experiments were conducted using $|I_S| = 25$ with a minimum of 10 users rating each movie.

Results are shown in Figure 25.5 for all attack models. Despite the minimal knowledge required for the love/hate attack, this attack proved to be the most effective against the user-based algorithm. Among the other nuke attacks, the bandwagon attack actually surpassed the average attack, which was not the case with the push results discussed above.

The asymmetry between these results and the push attack data is somewhat surprising. For example, the love/hate attack produced a positive prediction shift slightly over 1.0 for a push attack of 10% against the user-based algorithm, which is much less effective than even the random attack. However, when used to nuke

an item against the user-based algorithm, this model was by far the most effective model we tried, with a prediction shift of almost twice that of the average attack. For pushing items, the average attack was the most successful, while it proved to be one of the least successful attacks for nuking items. The bandwagon attack, on the other hand, performed nearly as well as the average attack in pushing items, and had superior overall performance for nuking, despite its lower knowledge requirement.

Overall, the item-based algorithm proved to be far more robust. The average attack was the most successful nuke attack here, with reverse bandwagon close behind. The asymmetries between push and nuke continue as we examine the item-based results. The random and love/hate attacks were poor performers for push attacks, but as nuke attacks, they actually failed completely to produce the desired effect. Reverse bandwagon (but not bandwagon) proved to be a reasonable low-knowledge attack model for a nuke attack against the item-based algorithm.

25.4.4 *Informed Attacks*

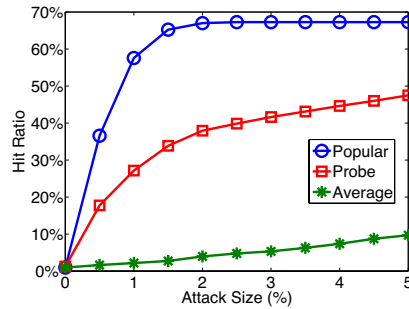
Finally, we turn to the evaluation of the informed attack strategies against the user-based algorithm. In particular, we compare the performance of the informed popular and probe push attacks to the average attack as seen above.

The attacks were implemented as follows. Popular attack profiles consisting of a total of 100 items (including the target item) were selected and assigned ratings as described in Section 25.3. For the probe attack, 10 seed items were selected at random from the 100 most frequently rated items from the system. Thereafter the system was interrogated to discover additional profile items and ratings. In total, probe attack profiles consisted of 100 items. Likewise, the benchmark average attack profiles consisted of 100 items, which corresponds to a filler size of approximately 1.7%. For the purposes of comparison, the 100 most frequently-rated items were chosen for average attack profiles (and not selected randomly, as before).

Figure 25.6 shows the hit ratios achieved by the three attacks. It is clear from the figure that the impact of the informed attacks was significantly greater than that of the average attack. For example, for an attack size of only 2%, the hit ratios achieved by the popular, probe and average attacks were 65%, 34% and 3%, respectively, for top N lists of size 10. Thus the advantage of creating attacks that consider particular features of the algorithm under attack is clearly demonstrated.

The main drawback associated with the informed attacks lies in the high degree of domain knowledge that is required in order to select the appropriate items and ratings with which to create the attack profiles. As discussed in Section 25.3, however, such knowledge is often made directly available to attackers by recommender system applications. Further, the knowledge required can often be obtained from other sources, e.g. by examining best seller lists and the number of positive and negative reviews received by items, etc. Even in situations where such data is only partially available, previous work demonstrates that these informed attacks retain their strong performance [26].

Fig. 25.6 Hit ratios achieved by the popular, probe and average push attacks against the user-based algorithm.



25.4.5 Attack impact

It is clear from the research summarized above that the memory-based algorithms that form the core of collaborative recommendation research and practice are highly vulnerable to manipulation. An attacker with fairly limited knowledge can craft attacks that will make any item appear well liked and promote it into many users’ recommendation lists. The “efficient” attacks that have been developed clearly are a threat to the stability and usability of collaborative systems and thus we see the justification for the low-scale / high-impact portion of the theoretical curve shown in Figure 25.1.

To respond to this threat, researchers have examined two complementary responses. The shaded “detection” areas in Figure 25.1 point towards the first response, which is to detect the profiles that make up an attack and eliminate them. The second approach is to design algorithms that are less susceptible to the types of attacks that work well against the classic algorithms.

25.5 Attack Detection

Figure 25.7 summarises the steps involved in attack detection. This is a binary classification problem, with two possible outcomes for each profile, namely, *Authentic*, meaning that the classifier has determined that the profile is that of a genuine system user or *Attack*, meaning that the classifier has determined that this is an instance of an attack profile. One approach to the detection problem, followed by work such as [7, 1], has been to view it as a problem of determining independently for each profile in the dataset, whether or not it is an attack profile. This is the ‘single profile’ input shown in Figure 25.7. The input is a single rating vector \mathbf{r}_u , for some user u from the dataset. Before processing by the classifier, a *feature extraction* step may extract a set of features, $\mathbf{f}_u = (f_1, \dots, f_k)$ from the raw rating vector \mathbf{r}_u . The classifier takes \mathbf{f}_u as input and outputs, “Attack” or “Authentic”. If the classifier is a *supervised* classifier, then a training phase makes use of annotated dataset of profiles, i.e. a set of profiles labelled as Authentic or Attack, in order to learn the classifier parameters.

Because most attack scenarios consist of groups of profiles working *in concert* to push or nuke a particular item, work such as [16, 23] has suggested that there is benefit to considering groups of profiles *together* when making the classification. This is represented by the ‘Group of Profiles’ input, in which the classifier considers an entire group of profiles, possibly after some feature extraction, and outputs a label for each profile in the group. Note that not all steps may take place in any particular scenario. For instance, there may be no feature extraction, in which case, $\mathbf{f} = \mathbf{r}$ and if *unsupervised* classifiers are used, then there is no need for a training phase.

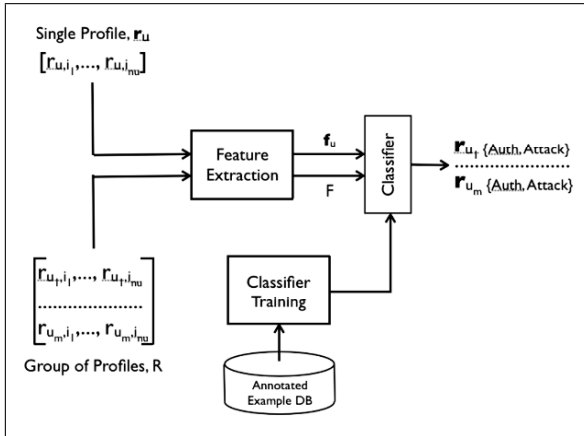


Fig. 25.7: The detection process.

25.5.1 Evaluation Metrics

To compare different detection algorithms, we are interested primarily in measures of classification performance. Taking a ‘positive’ classification to mean the labeling of a profile as Attack, a confusion matrix of the classified data contains four sets, two of which – the true positives and true negatives – consist of profiles that were correctly classified as Attack or Authentic, respectively; and two of which – the false positives and false negatives – consist of profiles that were incorrectly classified as Attack or Authentic, respectively. Various measures are used in the literature to compute performance based on the relative sizes of these sets. Unfortunately, different researchers have used different measures, making direct comparison of results sometimes difficult.

Precision and *recall* are commonly used performance measures in information retrieval. In this context, they measure the classifier’s performance in identifying at-

tacks. Each measure counts the number of attack profiles correctly classified. Recall which is also called *sensitivity* presents this count as a fraction of the total number of actual attacks in the system. Precision, which is also called the *positive predictive value* (PPV), presents this count as a fraction of the total number of profiles labelled as Attack:

$$\begin{aligned} recall \equiv sensitivity &= \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false negatives}}, \\ precision \equiv PPV &= \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false positives}}. \end{aligned} \quad (25.1)$$

Analogous measures can be given for performance in identifying authentic profiles. *Specificity* presents the count of authentic profiles correctly classified as a fraction of the total number of authentic profiles in the system. *Negative predictive value* (NPV), presents the count as a fraction of the total number of profiles labelled Authentic:

$$\begin{aligned} specificity &= \frac{\# \text{ true negatives}}{\# \text{ true negatives} + \# \text{ false positives}}, \\ NPV &= \frac{\# \text{ true negatives}}{\# \text{ true negatives} + \# \text{ false negatives}}. \end{aligned} \quad (25.2)$$

In detection results below, we use the terms *precision*, *recall*, *specificity* and *NPV*.

25.5.1.1 Impact on Recommender and Attack Performance

The misclassification of authentic profiles results in the removal of good data from the ratings database, which has the potential to impact negatively on the overall performance of the recommender system. One way to assess this impact is to compute the MAE of the system before and after detection and filtering. On the positive side, the removal of attack profiles reduces attack performance. Assuming the attack is a push or nuke attack, the degree to which attack performance is affected can be assessed by computing the prediction shift on the targeted item before and after detection and filtering.

25.5.2 Single Profile Detection

The basis of individual profile detection is that the distribution of ratings in an attack profile is likely to be different to that of authentic users and therefore each attack profile can be distinguished by identification of these differences. As such, individual profile detection is an instance of a *statistical detection* problem. It should be noted that it is in the interest of the attacker to minimise the statistical differences between attack and authentic profiles, in order to minimise the probability of de-

tection. On the other hand, a cost-effective attack is likely to consist of unusually influential profiles – e.g., a targeted pushed item will have unusually high ratings and filler items may have been chosen to support the influence of the profile towards high ratings for the target. As a result, distinctive characteristics are likely to exist and may be manifested in many ways, including an abnormal deviation from the system average rating, or an unusual number of ratings in a profile [1].

25.5.2.1 Unsupervised Detection

An unsupervised individual profile detection algorithm is described in [7]. Detection is based on certain common generic attributes of attack profiles, for example that there is a higher than usual rating deviation from mean in such profiles and that such profiles are likely to have a higher than usual similarity to their closest neighbours. Measures of these attributes are proposed and these are applied to compute a probability that a profile is an attack profile.

25.5.2.2 Supervised Detection

Supervised detection algorithms have focussed on the selection of attributes of attack profiles from which to build a feature vector for input to a classifier. Generally, such features have been selected by observation of *generic* attributes that are common across attack profiles of a number of different attack strategies and also *model specific* attributes that are common across profiles that have been generated for a specific type of attack.

In [5] profile attributes based to those proposed in [7] and others along similar lines were developed into features for inclusion in a feature vector input to a supervised classifier. Moreover, other features based on the statistics of the filler and target items in the user profile, rather than the entire profile, were proposed. For example, the *filler mean variance* feature is defined as the variance of the ratings in the filler partition of the profile and is used to detect average attacks; the *filler mean target difference* feature, defined as the difference between the means of the target items and the means of the filler items, is used to detect bandwagon attacks.

The authors looked at three supervised classifiers: kNN, C4.5, and SVM. The kNN classifier uses detection attributes of the profiles to find the $k = 9$ nearest neighbors in the training set using Pearson correlation for similarity to determine the class. The C4.5 and SVM classifiers are built in a similar manner such that they classify profiles based on the detection attributes only. The results for the detection of a 1% average attack over various filler sizes are reproduced in Figure 25.8. SVM and C4.5 have near perfect performance on identifying attack profiles correctly, but on the other hand, they also misclassify more authentic profiles than kNN. SVM has the best combination of recall and specificity across the entire range of filler sizes for a 1% attack.

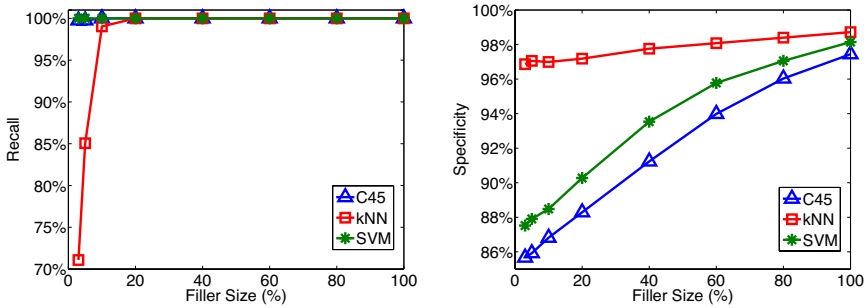


Fig. 25.8: Recall (left) and specificity (right) vs filler size for three classifiers trained on a 1% average attack.

The effect of misclassification of authentic profiles is assessed by examining the MAE of the system before and after detection and filtering. The increase in MAE is observed to be less than 0.05 on a rating scale of 1–5. Finally the effectiveness of the attack as measured by the prediction shift on the targeted item is shown to be significantly reduced when detection is used. All three classifiers reduce the range of attacks that are successful, particularly at low attack sizes. The SVM algorithm, in particular, dominates for attack sizes less than 10%, allowing no resulting prediction shift over that entire range.

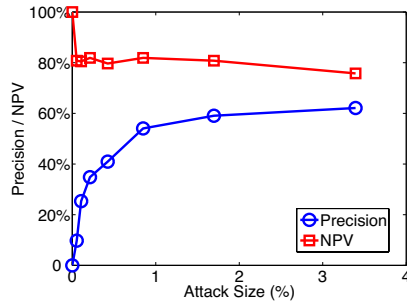
25.5.3 Group Profile Detection

A number of unsupervised algorithms that try to identify groups of attack profiles have been proposed [23, 33, 18]. Generally, these algorithms rely on clustering strategies that attempt to distinguish clusters of attack profiles from clusters of authentic profiles.

25.5.3.1 Neighbourhood Filtering

In [23] an unsupervised detection and filtering scheme is presented. Rather than filtering profiles from the dataset in a preprocessing step, in this method, filtering is applied to the profiles in the active user's neighbourhood during prediction for a particular item. This approach has the advantage of identifying just those attack profiles that are targeting the active item. The strategy is based on an algorithm proposed in [8] in the context of reputation reporting systems that aims to provide a reputation estimate for buyers and sellers engaged in on-line marketplaces that is robust to malicious agents who attempt to fraudulently enhance their own reputations. The ap-

Fig. 25.9 Precision and NPV for the neighbourhood filtering algorithm vs attack size.



proach involves the clustering of neighbourhoods into two clusters. Analysing the statistics of the clusters, a decision is made as to whether an attack is present and, if so, which cluster contains the attack profiles. *All* profiles in the cluster are removed.

Clustering is performed using the Macnaughton-Smith [13] divisive clustering algorithm. The rating distributions for the active item over each of the clusters are then compared. Since the goal of an attacker is to force the predicted ratings of targeted items to a particular value, it is reasonable to expect that the ratings for targeted items that are contained in any attack profiles are centered on the attack value, which is likely to deviate significantly from the mean of the authentic neighbours' ratings. Thus an attack is deemed to have taken place if the difference in the means for the two clusters is sufficiently large. The cluster with the smaller standard deviation is determined to be the attack cluster.

Results for this algorithm (using *precision* and *NPV*) applied to an informed nuke attack on the Movielens dataset are reproduced in Figure 25.9. The fraction of authentic users contained in the cluster identified as the cluster of authentic users is at least 75% for all attack sizes tested, so attack profiles are being effectively filtered from the system. However, particularly for small attack sizes, a significant proportion of the attack cluster is made up of authentic users. The cost of removing malicious profiles is to also lose authentic profiles that may have contributed to the accuracy of the prediction. Results show that filtering a system that has not been attacked leads to an increase of around 10% in the MAE.

25.5.3.2 Detecting attacks using Profile Clustering

In [18] the observation is made that attacks consist of multiple profiles which are highly correlated with each other, as well as having high similarity with a large number of authentic profiles. This insight motivates the development of a clustering approach to attack detection, using Probabilistic Latent Semantic Analysis (PLSA) and Principal Component Analysis (PCA).

In the PLSA model [11], an unobserved factor variable $Z = \{z_1, \dots, z_k\}$ is associated with each observation. In the context of collaborative recommendation, an observation corresponds to a rating for some user-item pair and ratings are predicted

using

$$\Pr(u, i) = \sum_{i=1}^k \Pr(z_i) \Pr(u|z_i) \Pr(i|z_i).$$

The parameters of this expression are chosen to maximise the likelihood of the observed data, using the Expectation Maximisation algorithm. As discussed in [21], the parameters $\Pr(u|z_i)$ can also be used to produce a clustering of the users by assigning each user u to each cluster C_i such that $\Pr(u|z_i)$ exceeds a certain threshold μ or to the cluster that maximises $\Pr(u|z_i)$ if μ is never exceeded.

It is noted in [18] that all or most attack profiles tend to be assigned to a single cluster. Identifying the cluster containing the attack profiles provides an effective strategy for filtering them from the system. Using the intuition that clusters containing attack profiles will be 'tighter' in the sense that the profiles are very similar to each other, the average Mahalanobis distance over the profiles of each cluster is calculated and that with the minimum distance is selected for filtering. Experiments show that PLSA based attack detection works well against strong attacks. However, for weaker attacks the attack profiles tend to be distributed across different clusters.

A second strategy to exploit the high similarity between attack profiles proposed in [18] is to base a clustering on a PCA of the covariance matrix of the user profiles. Essentially this strategy attempts to identify a cluster where the sum of the pair-wise covariances between profiles in the cluster is maximised. PCA has been widely used as a dimension reduction strategy for high-dimensional data. Identifying profiles with dimensions, the method is explained intuitively in [18] as a method of identifying those highly-correlated dimensions (i.e. profiles) that would safely be removed by PCA. Alternatively, a cluster C can be defined by an indicator vector \mathbf{y} such that $y(i) = 1$ if user $u_i \in C$ and $y(i) = 0$ otherwise. With S defined as the covariance matrix, the sum of the pair-wise covariances of all profiles in C , may be written as the quadratic form

$$\mathbf{y}^T \mathbf{S} \mathbf{y} = \sum_{i \in C, j \in C} S(i, j).$$

Moreover, for the normalised eigenvectors \mathbf{x}_i of S , associated with eigenvector λ_i such that $\lambda_1 \leq \dots \leq \lambda_m$, the quadratic form evaluates as

$$\mathbf{y}^T \mathbf{S} \mathbf{y} = \sum_{i=1}^m (\mathbf{y} \cdot \mathbf{x}_i)^2 (\mathbf{x}_i^T \mathbf{S} \mathbf{x}_i) = \sum_{i=1}^m (\mathbf{y} \cdot \mathbf{x}_i)^2 \lambda_i.$$

With this observation, the method described in [18] may be understood as a method that seeks the binary vector \mathbf{y} that maximises the quadratic form by choosing \mathbf{y} so that it has small correlation with those 3–5 eigenvectors corresponding to the smallest eigenvalues and hence correlates strongly with the eigenvectors corresponding to large eigenvalues.

Precision and recall results for the PLSA and PCA clustering strategies are reproduced in Figure 25.10 for an average attack of size 10%. Similar results have been obtained for random and bandwagon attacks. The PLSA and PCA clustering

strategies require that the size of the filtered cluster be specified and, in these results, the cluster size is taken to be the actual number of inserted attack profiles. This point should be taken into account in comparing the results with those obtained with the neighbourhood filtering strategy (Figure 25.9), in which no such control on the cluster size was applied. The 80% maximum recall obtained for the PLSA strategy is due to the fact that the wrong cluster is selected approximately 20% of the time. The PCA clustering strategy shows very good performance, even in the case of attacks consisting of a mixture of random, average and bandwagon profiles.

The UnRAP algorithm [3] also uses clustering to distinguish attack profiles. This algorithm uses a measure called the H_v score which has proved successful in identifying highly correlated biclusters in gene expression data. In the context of attack detection, the H_v score measures for each user, a sum of the squared deviations of its ratings from the user mean, item mean and overall mean ratings:

$$H_v(u) = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_i - \bar{r}_u + \bar{r})^2}{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2},$$

where \bar{r}_i is the mean over all users of the ratings for item i , \bar{r}_u is the mean over all items of the ratings for user u and \bar{r} is the mean over users and items.

A H_v score is assigned to all users in the database and users are sorted according to this score. The top $r = 10$ users with highest score are identified as potential attackers and are examined to identify a target item. The target is identified as that which deviates most from the mean user rating. Next, a sliding window of r users is passed along the sorted user list, shifting the window by one user each iteration. The sum of the rating deviation for the target item is calculated over the window and a stopping point is reached when this sum reaches zero. The users traversed during this process become candidate attack profiles, which are then further filtered by removing any that have not rated the item or whose rating deviation is in the opposite direction to the attack. Precision results for this method on an average attack are reproduced in Figure 25.10, compared with the PCA clustering strategy. In general, the authors report that this method performs well particularly for mid-size attacks, in which other methods show a dip in performance.

25.5.4 Detection findings

For both supervised and unsupervised detection, it has proved possible to achieve reasonably good performance against the attack types discussed in 25.3. Perhaps this is not so surprising, since the assumption is that these attacks are crafted according to a fairly regular pattern and thereby vary substantially from the real users of the system. The extent to which real-life attacks against recommender systems correspond to these idealized models is not known, since e-commerce companies

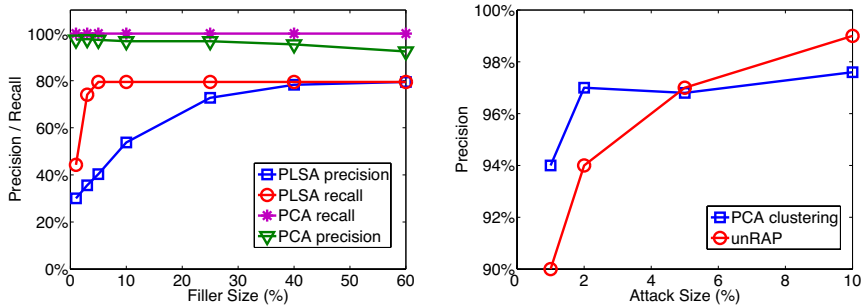


Fig. 25.10: Precision and recall for the PLSA and PCA clustering strategies vs filler size for a 10% average attack (left). Precision vs attack size for PCA clustering and UnRAP on an average attack, with filler size=10% (right).

have been reluctant to reveal vulnerabilities that they have identified in their own systems.

Going back to the framework in Figure 25.1, these findings give us some optimism that the shaded area at the upper left exists. That is, it is possible to detect attacks that are crafted to be optimal against the well-known memory-based algorithms. It remains an open question to what extent these detection measures extend downward and to the right, into regions where attacks differ from the optimal and have correspondingly less impact, but still remain a source of profit for the attacker.

25.6 Robust Algorithms

An alternative (or perhaps a complement) to filtering and detection is to develop recommendation algorithms that are intrinsically robust to attack. To date, researchers have largely tried to identify algorithms robust against the attacks that work well on the memory-based algorithms. An open question is whether new attacks can be tailored to exploit vulnerabilities in algorithms that have shown high robustness to standard attacks.

25.6.1 Model-based Recommendation

It has been shown in [21] that model-based recommendation algorithms provide a greater degree of robustness to attack strategies that have proven highly effective on memory-based algorithms. Moreover, this robustness does not come at a significant cost in terms of recommendation accuracy. This work has been followed up in [17,

15], which surveys model-based attack resistant algorithms and proposes a robust matrix factorisation strategy.

A model-based recommendation strategy based on clustering user profiles is analysed in [21]. In this strategy, similar users are clustered into segments and the similarity between the target user and a user segment is calculated. For each segment, an aggregate profile, consisting of the average rating for each item in the segment is computed and predictions are made using the aggregate profile rather than individual profiles. To make a recommendation for a target user u and target item i , a neighbourhood of user segments that have a rating for i and whose aggregate profile is most similar to u is chosen. A prediction for item i is made using the k nearest segments and associated aggregate profiles, rather than the k nearest neighbours. Both k -means clustering and PLSA-based clustering, as described in Section 25.5.3.2, are evaluated. The prediction shift achieved by an average attack on these algorithms, compared with the standard kNN algorithm, is shown in Figure 25.11 (left). The model-based algorithms are considerably more robust and not significantly less accurate, since, according to [21], PLSA and k -means clustering achieve an MAE of 0.75 and 0.76 using 30 segments, in comparison to a value of 0.74 for kNN.

25.6.2 Robust Matrix Factorisation (RMF)

One model-based approach to collaborative recommendation which has proven very successful recently, is the application of matrix factorisation approaches based on singular value decomposition (SVD) and its variants. Recent work in [15, 18] has suggested a robust factorisation strategy in which the clustering strategy of Section 25.5.3.2 is used in conjunction with the training phase of the factorisation procedure. For example, the PLSA clustering strategy can be applied in conjunction with the PLSA recommendation algorithm. In [15], it is proposed that after elimination of attack clusters, the $\Pr(z_i|u)$ distribution of the remaining clusters should be renormalised and the last few steps of training should be re-run, to maintain the predictive accuracy of the standard PLSA algorithm and significantly reduce prediction shift.

Another strategy proposed in [18] is in the context of the application of *Generalized Hebbian Learning* algorithm to compute a rank-1 SVD factorisation:

$$R \approx GH,$$

where R is the rating matrix and G and H are matrices of rank 1. Again, the algorithm is modified so that the contribution of the suspicious users towards the prediction model is zero, once suspicious users have been identified. Results from this strategy are reproduced in Figure 25.11 (right). The MAE for the attacked algorithm is shown when the number of suspicious users r is set to the exact number of attack profiles inserted, and when it is given a fixed value of 7% of the user base. Also shown for

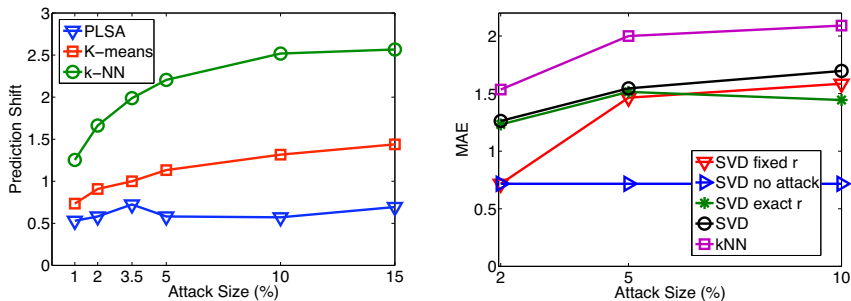


Fig. 25.11: Prediction shift vs attack size for an average attack at 5% filler for segment recommendation (left). MAE on the attacked item vs attack size for filler size of 10% using RMF (right).

reference is the MAE on the kNN algorithm and standard SVD, with and without attack.

Theoretical results are also emerging to support the robustness of particular classes of model-based algorithm. In [35], a manipulation-resistant class of collaborative filtering algorithm is proposed for which robustness is proved, in the sense that the effect of any attack on the ratings provided to an end-user diminishes with increasing number of products rated by the end-user. Here, effectiveness is measured in terms of a measure of the average distortion introduced by the attack to the ratings provided to the user. The class of algorithms for which the proof holds is referred to as a *linear* probabilistic collaborative filtering. In essence, the system is modelled as outputting a probability mass function (PMF) over the possible ratings and in linear algorithms, the PMF of the attacked system can be written as a weighted sum of the PMF obtained considering only genuine profiles and that obtained considering only attack profiles. Robustness is obtained, because, as the user supplies more ratings, the contribution of the genuine PMF to the overall PMF begins to dominate. The authors show that, while nearest neighbour algorithms are not linear in this sense, some well-known model-based algorithms such as the naive-bayes algorithm are asymptotically linear.

25.6.3 Other Robust Recommendation Algorithms

Attack profiles are ineffective if they do not appear in the neighborhoods of authentic users. By avoiding similarity as a criterion for neighbour selection, the recommendation algorithm can be made robust to attacks where the attack profiles are designed to have high similarity with authentic users. In [23] it is argued that the goal of neighbour selection is to select the most *useful* neighbours on which to base

the prediction. While similarity is one measure of usefulness, the notion of neighbour utility can be extended to include other performance measures. A selection criterion is proposed based on a notion of *inverse popularity*. It is shown that, with this selection strategy, the same overall system performance in terms of MAE is maintained. Moreover, cost-effective attacks that depend on popular items to build highly influential profiles are rendered much less effective.

In [31], a robust algorithm is presented based on association rule mining. Considering each user profile as a transaction, it is possible to use the *Apriori* algorithm to generate association rules for groups of commonly liked items. The support of an item set $X \subset I$ is the fraction of user profiles that contain this item set. An association rule is an expression of the form $X \Rightarrow Y(\sigma_r, \alpha_r)$, where σ_r is the support of $X \cup Y$ and α_r is the *confidence* for the rule, defined as $\sigma(X \cup Y)/\sigma(X)$. The algorithm finds a recommendation for a user u by searching for the highest confidence association rules, such that $X \subseteq P_u$ is a subset of the user profile and Y contains some item i that is unrated by u . If there is not enough support for a particular item, that item will never appear in any frequent item set and will never be recommended. This algorithm proves robust to the average attack. For attack sizes below 15%, only 0.1% of users are recommended an attacked item by the association rule algorithm, compared to 80 – 100% of users for the kNN algorithm. The trade-off is that coverage of the association rule algorithm is reduced in comparison to kNN . However, the algorithm is not robust against the segment attack.

25.6.4 The Influence Limiter and Trust-based Recommendation

In [28, 29] a recommendation algorithm is presented for which robustness bounds can be calculated. The algorithm introduces two key additional features to the recommendation process, an *influence limiter* and a reputation system. The idea behind the algorithm is to weight the contribution of each user towards a prediction by using a global measure of reputation. The reputation value is boosted when a profile correctly estimates a rating for a neighbor and is reduced which it fails to do so. Within this recommendation model, the authors prove a non-manipulation result that shows that any attack strategy involving up to n attack users, the negative impact due to the attacker is bounded by a small amount. They also show that a user seeking to maximize influence has a strict incentive to rate honestly. Other properties of this algorithm, such as its accuracy, are still under study.

The influence limiter is just one algorithm that takes into account *trust* and *reputation* (see Chapter 20) in order to build recommendations. In recent years, there has been increasing focus on incorporating trust models into recommender systems [14, 22, 9]. In [14], trust propagation is used to increase the coverage of recommender systems while preserving accuracy. In [22] it is argued that the reliability of a profile to deliver accurate recommendations in the past should be taken into account by recommendation algorithms. An algorithm that uses trust as a means of filtering profiles prior to recommendation so that only the top k most trustworthy

profiles participate in the prediction process is presented in [9]. The trust associated with a user for making predictions for an item is computed based on the users' accuracy on predicting their own ratings for that item. The robustness achieved by such algorithms is a function of how difficult it would be for an attacker to become trusted.

25.7 Conclusion

Collaborative recommender systems are meant to be adaptive – users add their preferences to these system and their output changes accordingly. Robustness in this context must mean something different than the classical computer science sense of being able to continue functioning in the face of abnormalities or errors. Our goal is to have systems that adapt, but that do not present an attractive target to the attacker. An attacker wishing to bias the output of a robust recommender system would have to make his attack sufficiently subtle that it does not trigger the suspicion of an attack detector, sufficiently small that it does not stand out from the normal pattern of new user enrollment, and sufficiently close to real user distribution patterns that it is not susceptible to being separated out by dimensionality reduction. If this proves a difficult target to hit and if the payoff for attacks can be sufficiently limited, the attacker may not find the impact of his attack sufficiently large relative to the effort required to produce it. This is the best one can hope for in an adversarial arena.

It is difficult to say how close we have come to this ideal. If an attacker is aware that such detection strategies are being applied, then the attack can be modified to avoid detection. For example, [23] shows that if the attacker is aware of the criteria used to decide if an attack profiles exist in the user's neighbourhood, then the attacker can construct profiles which, although somewhat less effective than the standard attacks, can circumvent detection. In [34] the effectiveness of various types of attack profile obfuscation are evaluated. The general finding is that obfuscated attacks are not much less effective than optimal ones and much harder to detect. More research is needed in this area.

Similar issues apply in the context of attack resistant recommendation algorithms. While model-based algorithms show robustness to attacks that are effective on memory-based algorithms, it is possible to conceive of new attacks that target model-based algorithms. [31], for example, shows that association rule based recommendation is vulnerable to segment attacks.

Another way to view the problem is as a game between system designer and attacker. For each system that the designer creates, an optimal attack against it can be formulated by the attacker, which then requires another response from the designer, etc. What we would like to see is that there are diminishing returns for the attacker, so that each iteration of defense makes attacking more expensive and less effective. One benefit of a detection strategy is that a system with detection cannot be more vulnerable to attack than the original system, since in the worst case, the attacks are not detected. We do not yet know if the robust algorithms that have been proposed

such as RMF have some as-yet-undiscovered flaw that could make them vulnerable to a sophisticated attack, perhaps even more vulnerable than the algorithms that they replace.

Acknowledgements

Neil Hurley would like to acknowledge the support of Science Foundation Ireland, grant number 08/SRC/I1407: Clique: Graph and Network Analysis Cluster. Michael O'Mahony is supported by Science Foundation Ireland under grant 07/CE/I1147: CLARITY: Centre for Sensor Web Technologies.

References

1. A. Williams, C., Mobasher, B., Burke, R.: Defending recommender systems: detection of profile injection attacks. *Service Oriented Computing and Applications* pp. 157–170 (2007)
2. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence* pp. 43–52 (1998)
3. Bryan, K., O'Mahony, M., Cunningham, P.: Unsupervised retrieval of attack profiles in collaborative recommender systems. In: *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pp. 155–162. ACM, New York, NY, USA (2008). DOI <http://doi.acm.org/10.1145/1454008.1454034>
4. Burke, R., Mobasher, B., Bhaumik, R.: Limited knowledge shilling attacks in collaborative filtering systems. In *Proceedings of Workshop on Intelligent Techniques for Web Personalization (ITWP'05)* (2005)
5. Burke, R., Mobasher, B., Williams, C.: Classification features for attack detection in collaborative recommender systems. In: *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pp. 17–20 (2006)
6. Burke, R., Mobasher, B., Zabicki, R., Bhaumik, R.: Identifying attack models for secure recommendation. In: *Beyond Personalization: A Workshop on the Next Generation of Recommender Systems* (2005)
7. Chirita, P.A., Nejdl, W., Zamfir, C.: Preventing shilling attacks in online recommender systems. In *Proceedings of the ACM Workshop on Web Information and Data Management (WIDM'2005)* pp. 67–74 (2005)
8. Dellarocas, C.: Immunizing on-line reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00)* pp. 150–157 (2000)
9. Fug-uo, Z., Sheng-hua, X.: Analysis of trust-based e-commerce recommender systems under recommendation attacks. In: *ISDPE '07: Proceedings of the The First International Symposium on Data, Privacy, and E-Commerce*, pp. 385–390. IEEE Computer Society, Washington, DC, USA (2007). DOI <http://dx.doi.org/10.1109/ISDPE.2007.55>
10. Herlocker, J., Konstan, J., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd International ACM SIGIR Conference on Research and Development in Information Retrieval* pp. 230–237 (1999)
11. Hofmann, T.: Collaborative filtering via gaussian probabilistic latent semantic analysis. In: *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research*

- and development in information retrieval, pp. 259–266. ACM, New York, NY, USA (2003). DOI <http://doi.acm.org/10.1145/860435.860483>
12. Lam, S.K., Riedl, J.: Shilling recommender systems for fun and profit. In Proceedings of the 13th International World Wide Web Conference pp. 393–402 (2004)
 13. Macnaughton-Smith, P., Williams, W.T., Dale, M., Mockett, L.: Dissimilarity analysis – a new technique of hierarchical sub-division. *Nature* **202**, 1034–1035 (1964)
 14. Massa, P., Avesani, P.: Trust-aware recommender systems. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 17–24. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1297231.1297235>
 15. Mehta, B., Hofmann, T.: A survey of attack-resistant collaborative filtering algorithms. *Bulletin of the Technical Committee on Data Engineering* **31(2)**, 14–22 (2008). URL <http://sites.computer.org/debull/A08June/mehta.pdf>
 16. Mehta, B., Hofmann, T., Fankhauser, P.: Lies and propaganda: Detecting spam users in collaborative filtering. In: Proceedings of the 12th international conference on Intelligent user interfaces, pp. 14–21 (2007)
 17. Mehta, B., Hofmann, T., Nejdl, W.: Robust collaborative filtering. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 49–56. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1297231.1297240>
 18. Mehta, B., Nejdl, W.: Unsupervised strategies for shilling detection and robust collaborative filtering. *User Modeling and User-Adapted Interaction* **19(1-2)**, 65–97 (2009). DOI <http://dx.doi.org/10.1007/s11257-008-9050-4>
 19. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Effective attack models for shilling item-based collaborative filtering system. In Proceedings of the 2005 WebKDD Workshop (KDD'2005) (2005)
 20. Mobasher, B., Burke, R., Bhaumik, R., Williams, C.: Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology* **7(4)** (2007)
 21. Mobasher, B., Burke, R.D., Sandvig, J.J.: Model-based collaborative filtering as a defense against profile injection attacks. In: AAI. AAI Press (2006)
 22. O'Donovan, J., Smyth, B.: Is trust robust?: an analysis of trust-based recommendation. In: IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces, pp. 101–108. ACM, New York, NY, USA (2006). DOI <http://doi.acm.org/10.1145/1111449.1111476>
 23. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: An evaluation of neighbourhood formation on the performance of collaborative filtering. *Artificial Intelligence Review* **21(1)**, 215–228 (2004)
 24. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: Promoting recommendations: An attack on collaborative filtering. In: A. Hameurlain, R. Cicchetti, R. Traunmüller (eds.) *DEXA, Lecture Notes in Computer Science*, vol. 2453, pp. 494–503. Springer (2002)
 25. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: An evaluation of the performance of collaborative filtering. In Proceedings of the 14th Irish International Conference on Artificial Intelligence and Cognitive Science (AICS'03) pp. 164–168 (2003)
 26. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: Recommender systems: Attack types and strategies. In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05) pp. 334–339 (2005)
 27. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., J.Riedl: Grouplens: An open architecture for collaborative filtering of netnews. In Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94) pp. 175–186 (1994)
 28. Resnick, P., Sami, R.: The influence limiter: provably manipulation-resistant recommender systems. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 25–32. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1297231.1297236>
 29. Resnick, P., Sami, R.: The information cost of manipulation-resistance in recommender systems. In: RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems,

- pp. 147–154. ACM, New York, NY, USA (2008). DOI <http://doi.acm.org/10.1145/1454008.1454033>
30. Rokach, L.: Mining manufacturing data using genetic algorithm-based feature set decomposition, *Int. J. Intelligent Systems Technologies and Applications*, 4(1):57-78 (2008).
 31. Sandvig, J.J., Mobasher, B., Burke, R.: Robustness of collaborative recommendation based on association rule mining. In: *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pp. 105–112. ACM, New York, NY, USA (2007). DOI <http://doi.acm.org/10.1145/1297231.1297249>
 32. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference* pp. 285–295 (2001)
 33. Su, X.F., Zeng, H.J., Chen, Z.: Finding group shilling in recommendation system. In: *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pp. 960–961. ACM, New York, NY, USA (2005). DOI <http://doi.acm.org/10.1145/1062745.1062818>
 34. Williams, C., Mobasher, B., Burke, R., Bhaumik, R., Sandvig, J.: Detection of obfuscated attacks in collaborative recommender systems. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)* (2006)
 35. Yan, X., Roy, B.V.: Manipulation-resistnat collaborative filtering systems. In: *RecSys '09: Proceedings of the 2009 ACM conference on Recommender systems*. ACM, New York, NY, USA (2009)

Index

- F_1 -measure, 60
- Accuracy, 59, 517
- Accuracy, confidence, effort (ACE), 517
- Active learning, 735
- Adaptive educational hypermedia (AEH), 389, 393
- Adjusted cosine (AC), 125
- Affective state, 677, 678, 688, 698
- Affinity propagation, 63
- Aggregation, 678, 682, 698
- Aggregation function, 704, 708, 712
- Aggregation property, 713
 - Associative, 713
 - Homogeneous, 713
 - Idempotent, 713
 - Lipschitz continuous, 713
 - Shift-invariant, 713
 - Strictly monotone, 713
 - Symmetric, 713
- Aggregation weight, 708–710, 712, 714, 717, 719, 726
- Alternating least squares, 152
- Ambient intelligence, 679, 697
- Application, 335
- Application infrastructure, 339
- Apriori, 65
- Arc consistency, 204
- Arithmetic mean, 714, 718
- Artificial neural network (ANN), 54
- Association rule mining, 64
- Attack detection, 820
- Attribute weights, 559, 564
- Autoregressive moving average (ARMA), 326
- Average commute time, 139
- Average first-passage time, 139
- Average reciprocal hit-rank (ARHR), 109
- Backtracking, 203
- Bag-of-words, 308
- Baseline predictor, 148
- Bayesian belief network (BBN), 53
- Bayesian classifier, 52
- Bias, 148
- Card sorting, *see* validation of recommendations
- Case-based reasoning, 779
- Case-based recommender system, 188
- Choquet integral, 708, 710, 714, 718, 719
- Churn, 373, 375
- Classification, 48
- Clustering, 61, 557
- Cold-start problem, 131, 310, 677, 693, 695, 696, 699
- Collaborative filtering (CF) recommender system, 12, 111, 145, 298, 368–370, 375, 376, 380, 626, 706, 708, 709, 713–715, 722, 726, 784
 - Advantages, 111
 - Item-based, 314
 - item-item, 161
 - Memory-based, 184
 - Model-based, 112, 184
 - Neighborhood-based, 111
 - User-user, 161
 - Weaknesses, 656
- Collaborative web search, 579, 588
- Community-based recommender system, 13
- Community-based web search, 591
- Concept drift, 156
- Confidence, 517
- Configurable product, 210
- Conjunctive query, 204
- Constraint propagation, 203

- Constraint satisfaction problem, 203
- Constraint-based recommender system, 188
- Consumer buying behavior, 211
- Content knowledge, 370, 373, 375–377
- Content-based recommender system, 11, 110, 298, 371, 376, 377, 634, 706, 708, 709, 711, 778
 - Bayesian, 110
 - Limitations, 110
 - Rocchio algorithm, 110
- ContentWise, 299
- Context, 211, 219
- Context-aware recommender system, 218, 230
 - Context generalization, 234
 - Contextual modeling, 238
 - Contextual post-filtering, 237
 - Contextual pre-filtering, 233
 - Ensemble method, 243
 - Reduction-based approach, 233
- Contextual dimension, 225
- Contextual information, 219, 223, 228
- Contextual preference elicitation, 231
- Contextual segment, 235
- Conversational recommender system, 195, 419
- Conversion rate, 207
- Correlation coefficient, 162
- Cosine similarity, 42, 313
- Cosine vector (CV), 124
- Coverage, 494
- Critique
 - Platform, 438
 - Redundancy, 424
 - Relevance, 426
 - Representation, 422
 - Retrieval, 430
- Critique-based recommender system, 419
- Critiquing, 196, 419
- Cross-validation, 43
- Curse of dimensionality, 44
- CWAdvisor, 197

- Data, 344
- Data acquisition, 737
- Data envelopment analysis (DEA), 793
- Data expressiveness, 347
- Data feature, 146
- Data quality, 346
- Data quantity, 347
- DBSCAN, 63
- Debugging, 192
- Decision accuracy, 517, 523
- Decision tree, 50, 93
- Demographic recommender system, 12, 707, 711, 723
- Device, 339
- Diary studies, *see* validation of recommendations
- Dimensionality reduction, 44
- Dissimilarity measure, 555
- Distance measure, 41
- Diversity, 528
- Domain knowledge ontology, 377

- Effectiveness, 483, 488
- Efficiency, 490
- Embodied interface agent, 467
 - Humanoid virtual agent, 467
 - Voice interface, 469
- Embodied Interface Agents
 - Anthropomorphism, 467, 470
- Ensemble, 58, 243, 756
 - Bagging, 58
 - Boosting, 58
- Entropy, 50
- Euclidean distance, 41
- Evaluation, 456, 463, 464, 466, 467, 637, 664
 - Criteria, 446
 - Critiquing, 445
 - Datasets, 446
 - Offline, 261
 - Online, 266
- Evaluation measure
 - Average reciprocal hit-rank (ARHR), 109
 - Mean absolute error (MAE), 109
 - Precision, 109
 - Recall, 109
 - Root mean squared error (RMSE), 109
- Example critiquing, 435, 525
- Explanation, 209, 481
 - Case-based, 503
 - Collaborative-based, 500
 - Content-based, 501
 - Demographic-based, 505
 - Interface, 536
 - Knowledge-based, 504
- Explicit feedback, 146
- Explicit profiling, 584
- Exploitation, 737
- Exploration, 736, 737, 748, 763
- Exploration vs. exploitation, 28
- Exponential diffusion kernel, 137

- Fastweb, 299
- Filtering method, 463
 - Collaborative, 463
 - Compensatory recommender system, 463
 - Content, 463
 - Feature-based system, 463

- Hybrid, 463
 - Needs-based system, 463
 - Non-compensatory recommender systems, 463
- Finite state model, 190
- First-rater problem, 310
- Flexible mixture model, 789
- FolkRank, 632
- Folksonomy, 79, 616, 618
- Frequency-weighted pearson correlation (FWPC), 129
- Frequent itemset, 65
- Fuzzy measure, 718, 719
- Fuzzy set, 711, 712, 717

- Geometric mean, 710, 714
- Gini index, 50
- Gradient descent, 152
- Graphical shopping interface, 562, 570
- Group model, 678, 681, 698
- Group recommender system, 419, 676, 677, 679
 - Applications, 680
 - Classification, 681
- Guideline
 - System design, 520

- Heterogeneity, 372, 375–377, 379
- Hierarchical Bayesian models, 240
- Hierarchical clustering, 64
- Human-computer interaction (HCI), 17, 456, 462, 463
- Human-recommender system interaction, 463, 468
- Hybrid recommender system, 13, 368, 371, 707, 711

- Implicit dataset, 298
- Implicit feedback, 146, 150
- Implicit profiling, 584, 585
- Individual knowledge, 370, 376
- Information gain, 50
- Information overload, 2
- Information retrieval, 581
- Input characteristics, 464, 466
 - Preference elicitation method, 464
- Inspiration seeking, 562
- Interaction effort, 420
- Interaction style, 374, 379
- Interactive television, 679, 697
- IP television, 298
- IP television (IPTV), 299
- Item, 8, 147
- Item similarity, 707, 709

- Item-based recommendation
 - Advantages, 118
- Itemset, 64

- Jaccard coefficient, 42

- K-means, 62
- Katz measure, 137
- Kernel trick, 57
- Knowledge acquisition bottleneck, 191
- Knowledge engineering, 191
- Knowledge source, 368, 369, 371, 372, 375, 378
- Knowledge-based recommender system, 12, 187, 371, 376–378, 707, 711, 724, 726, 779

- L2 Norm, 42
- Latent factor model, 151
- Latent-semantic analysis, 312
- Learning context, 392
- Learning network, 394
- Learning rate, 493
- Lift factor, 298, 326
- Likeability, 460
- Linear models
 - Generalized, 560
- Local consistency, 204
- Long tail, 323
- Low fidelity prototype, *see* validation of recommendations

- Machine learning, 80, 90
- Mahalanobis distance, 41
- Map based visualization, 554, 563
- Matrix factorization, 45, 151, 630
- Mean absolute error (MAE), 109
- Mean average error (MAE), 59
- Mean squared difference (MSD), 126
- Memory-based, 814, 820, 828
- Metadata, 311
- Minkowski distance, 41
- Misclassification error, 50
- Missing values, 555, 558, 560
- Mobile recommender system, 230
- Model selection, 757, 758
- Model-based, 787, 814, 828
- Movie, 298
- Multi-attribute decision problem (MADP), 515
- Multi-attribute utility theory (MAUT), 204, 515
- Multi-criteria decision making, 771
 - Preference modeling, 774
 - Roy's methodology, 771

- Multi-criteria optimization, 792
- Multi-criteria rating, 773, 781, 782, 784, 791
 - Consistency of criteria, 773
- Multi-criteria recommender system, 768
 - Algorithms, 783
 - Heuristic approaches, 784
 - Model-based approaches, 787
 - Similarity metrics, 784
- Multi-mode recommender system, 620, 634
- Multidimensional data, 226
- Multidimensional scaling, 553, 555, 562, 564
- Multiple criteria, 677, 693
- Multiple product alternative, 419

- Naive Bayes classifier, 52
- Neighborhood method, 161
- Neighborhood pre-filtering
 - Negative filtering, 130
 - Threshold filtering, 130
 - Top-*N* filtering, 130
- Neighborhood-based recommendation
 - Advantages, 112
 - Dimensionality reduction, 132
 - Graph-based methods, 135
 - Limitations, 131
 - Neighborhood selection, 129
 - Rating normalization, 121
 - Similarity weight, 124
 - User-based classification, 116
 - User-based prediction, 115
 - Weight significance, 127
 - Weight variance, 128
- Neighborhood-based recommender system
 - Item-based classification, 117
 - Item-based prediction, 117
- Neptun, 299
- Net consumer, 606
- Net producer, 605
- Netflix prize, 146
 - Data, 149
- New item problem, 310
- News recommender, 356
- Non-negative matrix factorization (NNMF), 47
- Norm, 727

- Ontology, 81, 88
- Open innovation, 209
- Ordered weighted averaging (OWA), 710, 714, 717, 719, 723
- Organization interface, 536
- Output characteristics, 465
- Over-specialization, 43

- PageRank, 583, 633

- Passive learning, 742
- Pearson coefficient, 162
- Pearson correlation (PC), 42, 124
- Perceptron, 54
- Performance criteria, 338
- Persuasion, 484
- Persuasiveness, 455, 456, 468, 470, 487
- Power mean, 708, 716
- Precision, 59, 109
- Prediction quality, 17
- Preference
 - Aggregation, 436
 - Conflict, 431, 531
 - Elicitation, 419, 521
 - Feedback, 419
 - Inconsistency, 431
 - Revision, 514, 530
 - Stability, 374, 376
- Principal components analysis (PCA), 44, 553, 558
 - Nonlinear, 568
- Privacy, 698
- Process characteristics, 465
- Product catalog map, 554, 563, 564, 568
- Product data extraction, 207
- Product nuke, 808
- Product push, 808
- Product retrieval system, 419
- Product search, 419
- Production environment, 298
- Profile
 - Long-term, 584
 - Short-term, 584

- Quasi-arithmetic mean, 714, 716
- Query management, 203
- Query relaxation, 201
- Query tightening, 203

- Rapid prototyping, 191
- Rating, 77, 78, 147, 349
 - Binary, 9
 - Explicit, 305
 - Implicit, 305
 - Missing, 148
 - Multi-criteria, 773
 - Numerical, 9
 - Ordinal, 9
 - Temporal dynamic, 154, 180
 - Temporal effect, 154, 180
 - Unary, 9
- Rating normalization
 - Mean-centering, 121
 - Z-score, 122

- Recall, 59, 109, 298, 325
- Receiver operating characteristic (ROC) curve, 60
- Recommendation
 - Diversity, 26
 - Non-personalized, 2
 - Personalized, 2
 - Query language, 248
 - Sequence, 27
 - Taxonomy, 368
 - Time value, 28
- Recommendation approach
 - collaborative filtering, 111
 - Content-based, 110
 - Itemrank, 138
- Recommendation by proposing, 562
- Recommendation dialog, 422
- Recommendation presentation, 466
- Recommendation problem
 - Best item, 108
 - Top- N , 108
- Recommendation task, 203
- Recommendation technique, 10
- Recommendation type, 337
- Recommender design, 334
- Recommender environment, 335
- Recommender knowledge base, 188
- Recommender purpose, 336
- Recommender role, 336
- Recommender system, 1
 - Case-based, 519
 - Conversational, 19
 - Critiquing-based, 520
 - Cross domain, 27
 - Data, 7
 - Distributed, 27
 - Emerging topics, 23
 - Function, 4
 - Mobile, 27
 - Perceived accuracy, 537
 - Preference-based, 513, 515, 519
 - Privacy preserving, 26
 - Proactive, 26
 - Rating-based, 519
 - Scalability, 26
 - Utility-based, 519
- Recommender user interface, 190
- Regression model
 - Poisson, 560
- Regularization, 148
- Regularized kernel matrix factorization, 47
- Relaxation, 202
- Relevance feedback, 75, 76, 92
- Repairing requirement, 202
- Risk, 290, 373, 375–377
- Robust algorithm, 828
- Robustness, 290, 807, 814
- Root mean squared error (RMSE), 59, 109, 150
- Rote-learner, 48
- Rule-based classifier, 51
- Sampling, 42, 757
 - Random, 43
- Satisfaction, 484, 686, 689, 697–699
 - Individual, 690
- Scrutability, 375, 376, 485
- Search engine, 579
- Self-organizing map, 550
- Semantic analysis, 81, 85
- Semantic web, 208
- Semi-structured data, 345
- Sequence order, 681, 686, 698
- Serendipity, 79, 97, 323
- Shrinkage, 128, 162
- Significance weighting, 127
- Similarity measure, 162
 - Ajusted cosine (AC), 125
 - Average commute time, 139
 - Average first-passage time, 139
 - Cosine vector (CV), 124
 - Exponential diffusion kernel, 137
 - Frequency-weighted pearson correlation (FWPC), 129
 - Katz, 137
 - Mean squared difference (MSD), 126
 - Number of paths, 136
 - Pearson correlation (PC), 124
 - Shortest path, 136
 - Spearman rank correlation (SRC), 126
 - Von Neumann diffusion kernel, 137
- Singular value decomposition (SVD), 45, 151, 790
- Skyline query, 793
- Social environment, 344
- Social information retrieval, 388
- Social knowledge, 369, 370, 375, 376
- Social network analysis (SNA), 406
- Social recommender system, 646
- Social search, 579, 590
- Social tagging recommender system, 616
- Social tagging system (STS), 615, 617, 621
- Soft navigation, 531
- Source characteristics, 455, 456, 458, 459, 462, 463, 467–470
 - Authority, 460, 469, 470
 - Credibility, 456–459
 - Humor, 461, 463, 470

- Likeability, 460, 469
- Physical attractiveness, 461, 462
- Similarity, 459, 462, 469
 - similarity, 458
- Speech Style, 461
- Sparsity, 310
- Spearman rank correlation (SRC), 126
- Stochastic gradient descent, 152
- Structured data, 346
- Subjective qualitative evaluation, *see* validation
 - of recommendations
- Sugeno integral, 718, 723
- Supervised classification, 48
- Support vector machine (SVM), 56, 242
- SVD++, 153

- T-conorm, 720
- T-norm, 720
- Tag-Aware recommender system, 621
- Tag-aware recommender system, 617, 627
- Technology enhanced learning (TEL), 388
 - Evaluation, 389, 404
 - Implementation, 400
 - Recommender system, 399
- Tensor factorization, 630
- Term frequency-inverse document frequency (TF-IDF), 110
- Testing, 192
- Time decay function, 180
- TimeSvd++, 159
- Top-rated items, 322
- Tradeoff, 532
- Transaction, 9
- Transparency, 483, 698
- Treemap, 551
- Trust, 18, 456, 457, 485
- Trust acquisition
 - explicit, 659
 - implicit, 662
- Trust aggregation, 654
- Trust metrics, 650
- Trust propagation, 650
- Trust-based collaborative filtering, 660

- Trust-enhanced recommender systems, 645
- Trustworthiness, 458
- Tweaking, 197, 526

- Unstructured data, 345
- Unsupervised classification, 48
- User, 8, 147, 340
- User characteristics, 340
- User effort, 517
- User expectation, 342
- User generated content, 94
- User interaction, 761
- User neighborhood, 706, 708, 709
- User profile, 27, 75, 80
- User requirement, 187
- User response
 - Binary, 108
 - Explicit, 108
 - Implicit, 108
 - Scalar, 108
 - Unary, 108
- User satisfaction, 491
- User similarity, 706–710, 715
- User task, 389, 390
- User-centric design, 351
- Utility function, 516
- Utility-based recommender system, 517, 707–709, 711, 714

- Validation of algorithms, 350
- Validation of recommendations, 351
- Vector space model, 81
- Video-on-demand, 298, 301
- Visualization
 - Map based, 549
 - Product, 549
- Von Neumann diffusion kernel, 137

- Web 2.0, 615, 640
- Wikipedia, 79, 89
- WordNet, 79, 88

- Z-score, 122