UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Natural Language Processing

Corpora and text processing

LESSON 5

prof. Antonino Staiano

M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

Eliza: A Rogerian psychotherapist

```
User:   I am unhappy.
ELIZA:  DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
User:   I need some help, that much seems certain.
ELIZA:  WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
User:   Perhaps I could learn to get along with my mother.
ELIZA:  TELL ME MORE ABOUT YOUR FAMILY
User:   My mother takes care of me.
ELIZA:  WHO ELSE IN YOU FAMILY TAKES CARE OF YOU
User:   My father.
ELIZA:  YOUR FATHER
User:   You are like my father in some ways.
```

Weizenbaum (1966)

- An early NLP system able to carry on a limited conversation with a user (chatbot)
  - It is based on pattern matching to recognize phrases like "*I need X*" and translate them into suitable output like "*What would it mean to you if you got X?*"

Eliza

- Despite its simplicity a pattern matching method at the basis of Eliza play a crucial role in natural language processing
- The most important tool for describing text pattern is the regular expression

Regular Expressions

Regular expressions

- Regular expression (RE), a language to specify text search strings
  - Is an algebraic notation for characterizing a set of strings
  - Useful in searching for *patterns* in a corpus of texts
- A regular expression function searches through the corpus and returns all texts that match the pattern
  - Unix program *grep*
- The corpus could be a single document or a collection
- A search can be planned to return every match on a line, if multiple matches exist, or just the first match
  - In our example we consider the latter only
- Regular expressions come in many variants. We describe here the so-called **extended** regular expressions

Regular expressions

- Basically, the simplest regular expression is a sequence (concatenation) of characters
- How can we search for any of these?
  - *woodchuck*
  - *woodchucks*
  - *Woodchuck*
  - *Woodchucks*

RE	Example Patterns Matched
`/woodchucks/`	"interesting links to woodchucks and lemurs"
`/a/`	"Mary Ann stopped by Mona's"
`/!/`	"You've left the burglar behind again!" said Nori

Disjunctions

- Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges [A-Z]

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Negation in Disjunction

- Negations [^Ss]
  - Caret (^) means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an upper case letter	Oyfn pripetchik
[^Ss]	Neither 'S' nor 's'	I have no exquisite reason"
[^e^]	Neither *e* nor ^	Look here
a^b	The pattern *a* caret *b*	Look up a^b now

More Disjunctions

- *Woodchuck* is another name for *groundhog*!
- The **pipe** | for disjunction

Pattern	Matches
`groundhog|woodchuck`	`woodchuck`
`yours|mine`	`yours`
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	`Woodchuck`

Regular Expressions

- `? *+.`

Pattern	Matches	
`colou?r`	Optional previous char	`color` `colour`
`oo*h!`	0 or more of previous char	`oh!` `ooh!` `oooh!` `ooooh!`
`o+h!`	1 or more of previous char	`oh!` `ooh!` `oooh!` `ooooh!`
`baa+`		`baa` `baaa` `baaaa` `baaaaa`
`beg.n`		`begin` `begun` `begun` `beg3n`

Regular Expressions: Anchors

- ^ and $

Pattern	Matches
^[A-Z]	Palo Alto
^[^A-Za-z]	1 "Hello"
\.$	The end.
.$	The end? The end!

Example

- Find me all instances of the word "*the*" in a text
  - `/the/`
    - misses capitalized examples
  - `/[tT]he/`
    - Incorrectly returns *other* or *theology*
  - `[^a-zA-Z][tT]he[^a-zA-Z]`

Errors

- The process we just went through was based on fixing two kinds of errors:

1. Matching strings that we should not have matched (there, then, other) **False positives (Type I errors)**

2. Not matching things that we should have matched (The) **False negatives (Type II errors)**

Errors cont.

- In NLP we are always dealing with these kinds of errors
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives)

Substitutions

- Substitution in Python and UNIX commands:
  - `s/regexp1/pattern/`
- Example
  - `s/colour/color/`

Capture groups

- Say we want to put angles around all numbers:

  the 35 boxes -> *the <35> boxes*

- capture groups are a way of storing part of the pattern into a "register" so we can refer to it later in the substitution string
- Use parens () to "*capture*" a pattern into a numbered register (1, 2, 3…)
- Use \1 to refer to the contents of the register
  - `s/([0-9]+)/<\1>/`

Capture groups: multiple registers

- Example
- `/the (.*)er they (.*), the \1er we \2/`

- Match: *"the faster they ran, the faster we ran"*
- Not match: *"the faster they ran, the faster we ate"*

But suppose we don't want to capture?

- Parentheses have a double function: grouping terms, and capturing
- Non-capturing groups
  - add a ?: after paren:

```
/(?:some|a few) (people|cats) like some \1/
```

- Match: `some cats like some cats`
- Not match: `some cats like some some`

Lookahead assertions

- `(?= pattern)` is true if pattern matches, but is **zero-width; doesn't advance character pointer**
- `(?! pattern)` true if a pattern does not match
- How to match, at the beginning of a line, any single word that doesn’t start with “Volcano”:
- `/^(?!Volcano)[A-Za-z]+/`

Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing
- For hard tasks, we use machine learning classifiers
  - But regular expressions are still used for pre-processing, or as features in the classifiers
  - Can be very useful in capturing generalizations

Simple Application: ELIZA

- Early NLP system that imitated a Rogerian psychotherapist
  - Joseph Weizenbaum, 1966.

- Uses pattern matching to match, e.g.,:
  - `"I need X"`

  and translates them into, e.g.
  - `"What would it mean to you if you got X?`

Simple Application: ELIZA

Men are all alike
IN WHAT WAY

They're always bugging us about something or other

CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here
YOUR BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time
I AM SORRY TO HEAR YOU ARE DEPRESSED

How ELIZA works

- Eliza works by having a series of RE substitutions
  - Each matches and changes part of the input lines
- `s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/`
- `s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/`
- `s/.* all .*/IN WHAT WAY?/`
- `s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE?/`
- Multiple substitutions can apply
  - They are assigned a rank and applied in order

NLTK library

- NLTK is a collection of modules and corpora, released under an open- source license, that allows students to learn and conduct research in NLP
- The most important advantage of using NLTK is that it is entirely self-contained. Not only does it provide convenient functions and wrappers that can be used as building blocks for common NLP tasks, but it also provides raw and pre-processed versions of standard corpora used in NLP literature and courses
- *NLTK Book*. https://www.nltk.org/book/
- *Natural Language Toolkit*. https://www.nltk.org/

Using NLTK

- NLTK ships with several useful text corpora
  - Brown Corpus
    - Considered to be the first general English corpus for computational linguistic processing tasks
    - 1.000.000 words of American English text printed in 1961
    - 15 genres, e.g., Fiction, News, and Religious text
    - Later, a POS-tagged version was also created
  - Gutenberg Corpus
    - Selection of 14 texts chosen from Project Gutenberg (the largest collection of free e-books)
    - 1.7 million words
      - Michael Hart and Gregory Newby. *Project Gutenberg*. Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics. http://www.gutenberg.org/wiki/Main_Page

Exploring Corpora

- **Task**: Use the NLTK corpus module to read the corpus **austen-persuasion.txt**, included in the Gutenberg corpus collection, and answer the following questions:
  - How many total words does this corpus have ?
  - How many unique words does this corpus have ?
  - What are the counts for the 10 most frequent words ?

Exploring NLTK's bundled corpora

1

```
# import the gutenberg collection
>>> from nltk.corpus import gutenberg
# what corpora are in the collection ?
>>> print gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt',
'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt',
'bryant-stories.txt', 'burgess-busterbrown.txt',
'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt',
'edgeworth-parents.txt', 'melville-moby_dick.txt',
'milton-paradise.txt', 'shakespeare-caesar.txt',
'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt',
'whitman-leaves.txt']
```

2

```
# import FreqDist class
>>> from nltk import FreqDist
# create frequency distribution object
>>> fd = FreqDist()
# for each token in the relevant text, increment its counter
>>> for word in gutenberg.words('austen-persuasion.txt')
...     fd.inc(word)
...
>>> print fd.N() # total number of samples
98171
>>> print fd.B() # number of bins or unique samples
6132
```

3

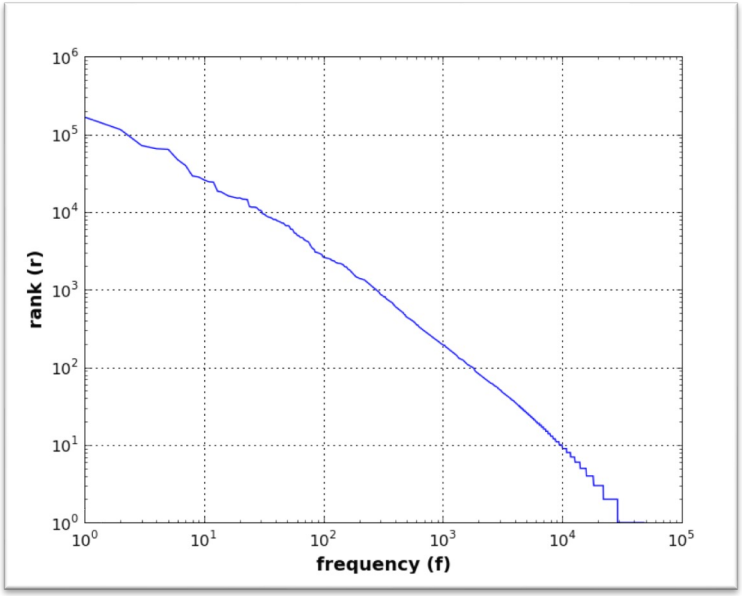
```
# Get a list of the top 10 words sorted by frequency
>>> for word in fd.keys()[:10]:
...     print word, fd[word]
, 6750
the 3120
to 2775
. 2741
and 2739
of 2564
a 1529
in 1346
was 1330
; 1290
```

Using NLTK to plot Zipf's Law

```
>>> from nltk.corpus import gutenberg
>>> from nltk import FreqDist
# For plotting, we need matplotlib (get it from the NLTK download page)
>>> import matplotlib
>>> import matplotlib.pyplot as plt

# Count each token in each text of the Gutenberg collection
>>> fd = FreqDist()
>>> for text in gutenberg.fileids():
...     for word in gutenberg.words(text):
...         fd.inc(word)

# Initialize two empty lists which will hold our ranks and frequencies
>>> ranks = []
>>> freqs = []

# Generate a (rank, frequency) point for each counted token and
# and append to the respective lists, Note that the iteration
# over fd is automatically sorted.
>>> for rank, word in enumerate(fd):
...     ranks.append(rank+1)
...     freqs.append(fd[word])
...

# Plot rank vs frequency on a log–log plot and show the plot
>>> plt.loglog(ranks, freqs)
>>> plt.xlabel('frequency(f)', fontsize=14, fontweight='bold')
>>> plt.ylabel('rank(r)', fontsize=14, fontweight='bold')
>>> plt.grid(True)
>>> plt.show()
```

Using NLTK to plot Zipf's Law

- Does Zipf's Law hold for the Guntenberg Corpus?



Assignment n. 1

- Prepare Jupiter notebooks for explaining text normalization using NLTK library
  - Corpus loading
  - Corpus statistics
  - Tokenization
  - Lemmatization
  - Stemming

Jupiter notebook/Google colab

- Jupyter Notebook
  - A Jupyter notebook lets you write and execute Python code locally in your web browser
  - Interactive, code re-execution, result storage, can interleave text, equations, and images
  - Can add conda environments to Jupyter notebook
- Google Colab
  - https://colab.research.google.com/
  - Google's hosted Jupyter notebook service, runs in the cloud, requires no setup to use, provides free access to computing resources including GPUs
  - Come with many Python libraries pre-installed

Others interesting tasks to try ...

- Language identification
  - Detecting the source language for the input text
    - Python langdetect
- Spell checkers
  - Correct grammatical mistakes in text
    - Python TextBlob based on NLTK
- Punctuation
  - Python string.punctuation
  - NLTK nltk.punkt