UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Natural Language Processing

Machine Translation

LESSON 13

prof. Antonino Staiano

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

The slides are based on those from DeepLearning.AI

Machine Translation

- Manipulating vectors enable you to translate one word from one language to another language
  - Word vectors are used to learn to **align** words in two different languages
- For instance, if we have a set of English word vectors and a set of equivalent French word vectors
  - The aim is to learn a mapping from an English vector to the French vector

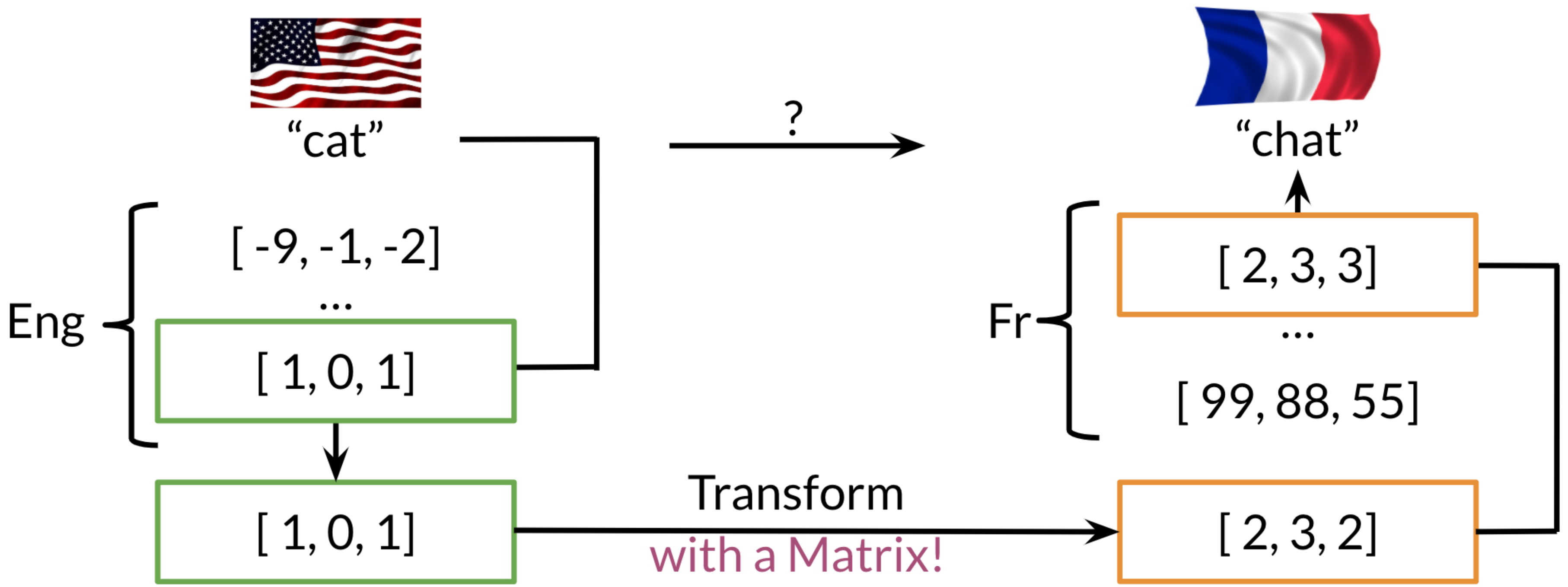
Overview of translation

- English to French translation
  - Generate an extensive list of English words and their associated French words

1. Compute the word embeddings associated with English and word embeddings associated with French
2. Retrieve the English word embedding of a given English word
3. Find a way to transform English word embedding into the same meaning French word embedding
   - By learning a transformation matrix
4. Search for word vectors in the French word vector space that are most similar to the transformed English word embedding
   - The most similar words are candidates words for your translation

Overview of translation

- Translating the English word *cat* in French



Transforming vectors

- How do we define the transformation matrix R to transform English vectors X into corresponding French vector Y ?
  - Formally, XR=Y
- Let's start by a random matrix R
- We first need to get a subset of English words and their French equivalents
  - Get their respective word vectors and stack the word vectors in the respective matrices, X and Y
  - It's mandatory to align the word vectors

$$\mathbf{XR} \approx \mathbf{Y}$$

$$\underbrace{\begin{pmatrix} [\text{ “cat” vector }] \\ [\text{ ... vector }] \\ [\text{ “zebra” vector }] \end{pmatrix}}_{\mathbf{X}} \qquad \underbrace{\begin{pmatrix} [\text{ “chat” vecteur }] \\ [\text{ ... vecteur }] \\ [\text{ “zébresse” vecteur }] \end{pmatrix}}_{\mathbf{Y}}$$

subsets of the full vocabulary

need to train on a subset of the English-French vocabulary and not the entire vocabulary

Finding a good R

- Define a loss function to measure the "quality" of the translation (transformation) w.r.t. the actual French words (vectors)
- Starting with a random R, we can iterate for the optimal R using the gradient descent

```
Initialize R
Loop
```

$$Loss = \|XR - Y\|_F^2$$

$$g = \frac{d}{dR} Loss$$

$$R = R - \alpha g$$

Frobenius norm

- Measures the magnitude of a matrix
  - $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$

$$\|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

$$\|\mathbf{A}_F\| = 4$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}\|_F^2 = \left(\sqrt{2^2 + 2^2 + 2^2 + 2^2}\right)^2$$

$$\|\mathbf{A}\|_F^2 = 16$$

Optimizing R

`Initialize R`

`Loop`

$$Loss = \|XR - Y\|_F^2$$

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \left|a_{ij}\right|^2}$$

$$g = \frac{d}{dR} Loss = \frac{2}{m}\left(\boldsymbol{X}^T(\boldsymbol{XR} - \boldsymbol{Y})\right)$$

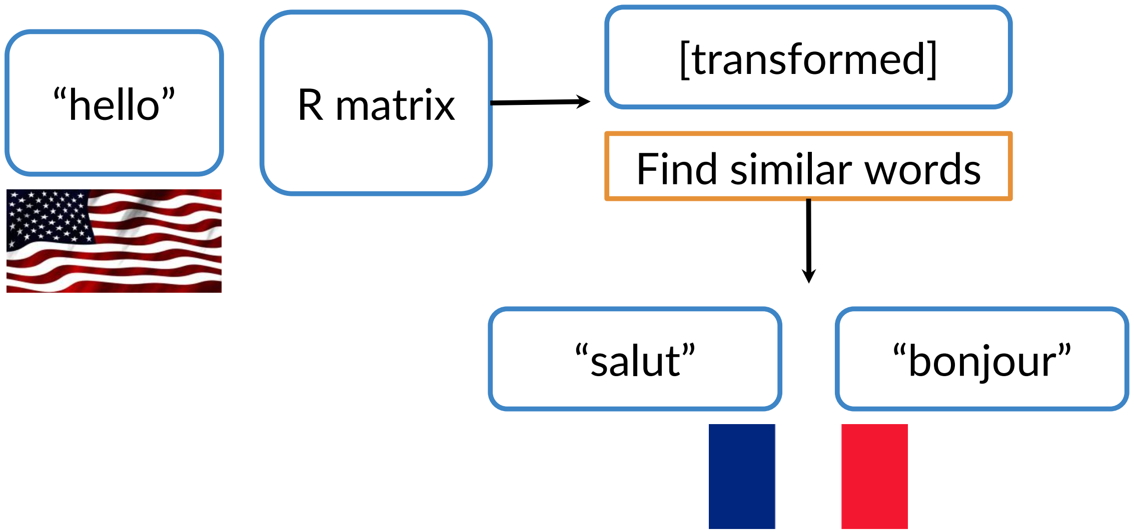
$$R = R - \alpha g$$

Machine Translation

K-nearest neighbors

Finding the translation

- A way to find a matching word after the transformation is trough k-nearest neighbors
- After a transformation through the matrix R, a vector v is in the French word vector space
  - v is not necessarily identical to any word vector in the French vector space
  - One needs to search through the actual French word vectors to find a similar word

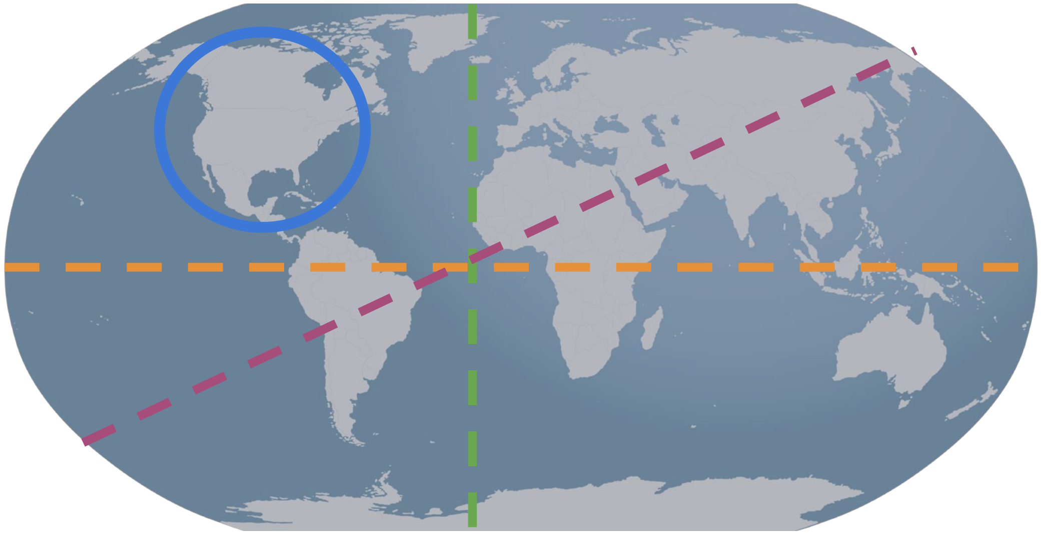


Nearest Neighbors intuition

San Francisco

Friend	Location	Nearest
	Shanghai	2
	Bangalore	3
	Los Angeles	1

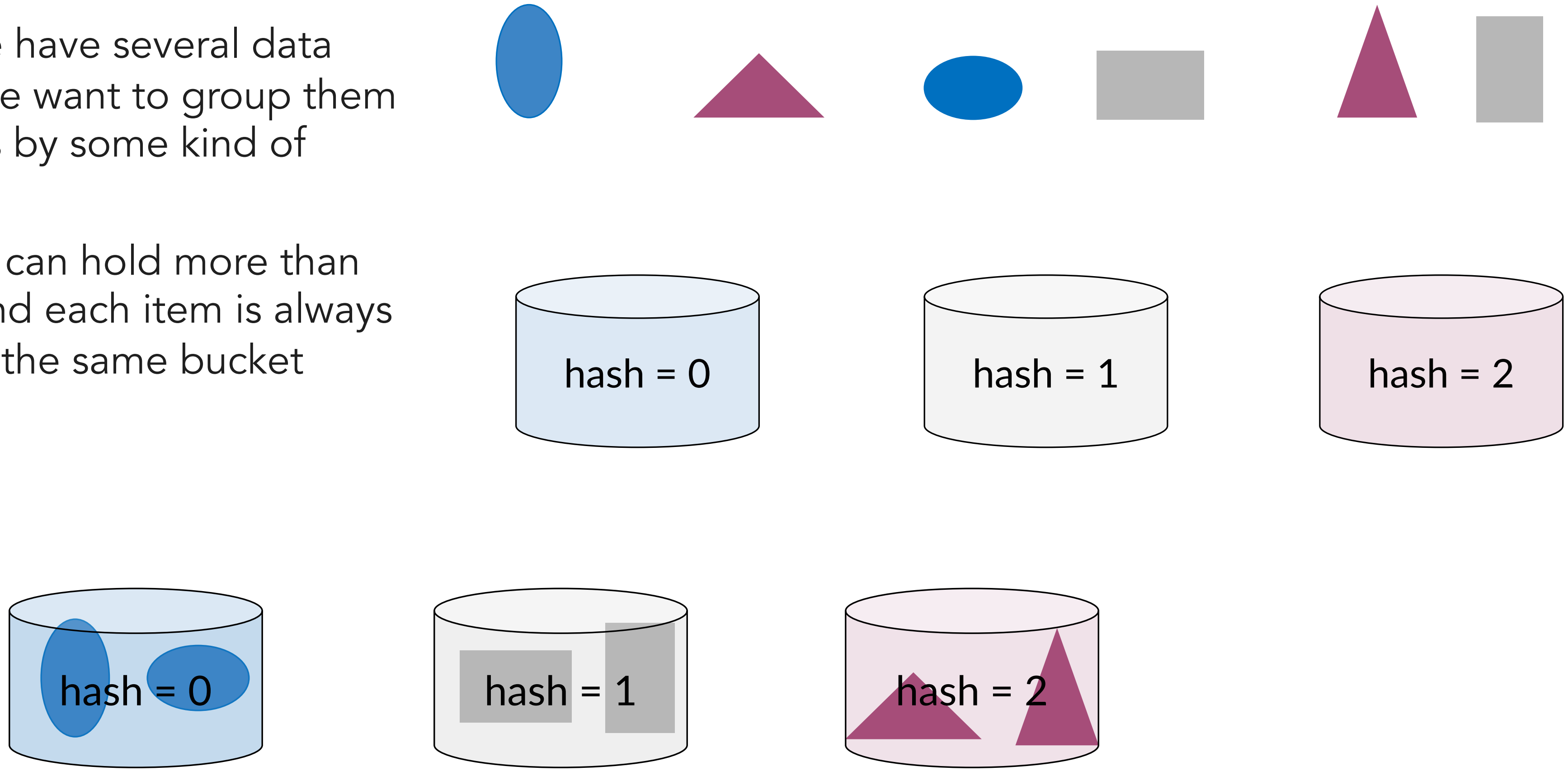
Nearest Neighbors intuition



Hash
tables!

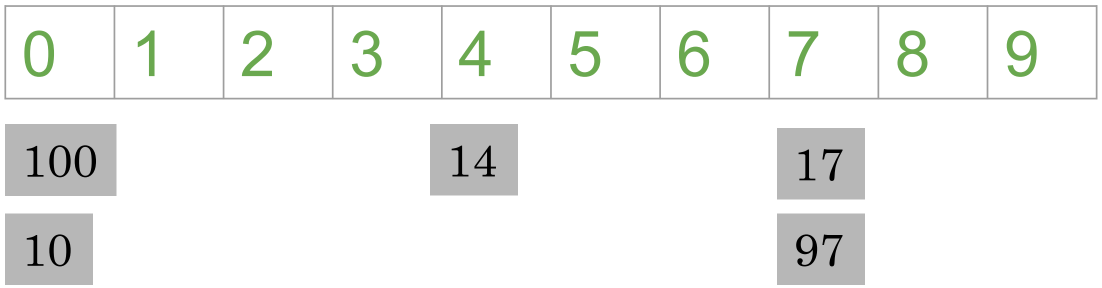
Hash Tables

- Suppose we have several data items and we want to group them into buckets by some kind of similarity
- One bucket can hold more than one item, and each item is always assigned to the same bucket



Hash function

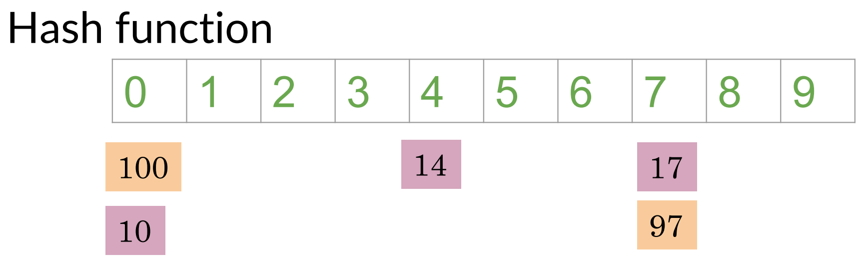
- Think about how we'd like to do this with word vectors
  - Assume that the word vectors have just one dimension, so each word is represented by a single number, such as 100, 14, 17, 10, and 97
  - A function that assigns a hash value is called a hash function
    - Example: Hash table which is a set of buckets, the hash table has 10 buckets



Hash function (vector) ⟶ Hash value

Hash value = vector % number of buckets

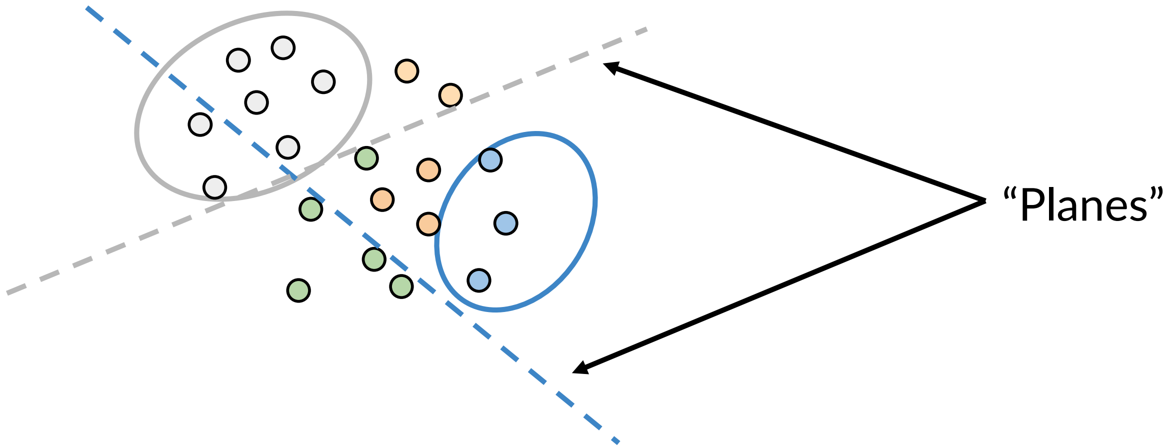
Hash function



- Ideally, you want to have a hash function that puts similar word vectors in the same buckets
  - Locality sensitive hashing
    - A hashing method that assigns items based on where they're located in vector space

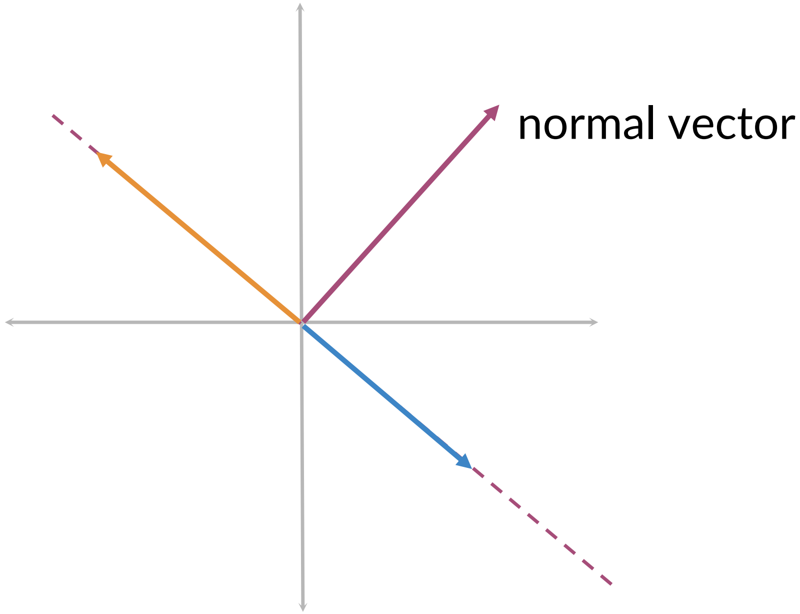
Locality Sensitive Hashing

- First divide the space using these dashed lines, which I'll call planes
  - The blue plane divides the space with blue vectors above it
  - The grey vectors are above the gray plane
- The plane helps us put the vectors into subsets based on their location



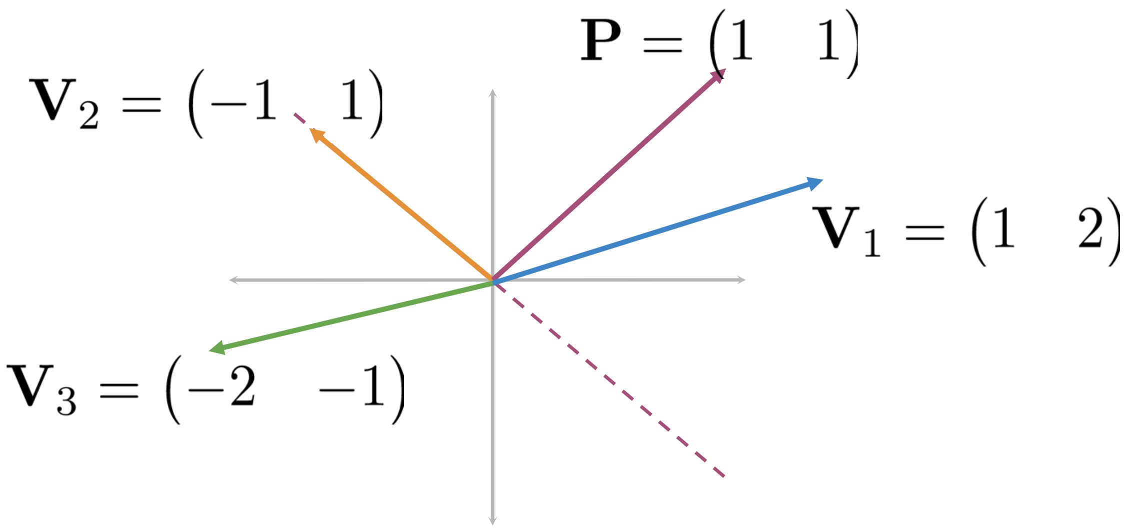
Planes

- A plane is the magenta dashed line
  - It represents all the possible vectors lying on that plane (e.g., the blue or orange vectors)
  - We can define a plane with the normal vector (e.g., magenta) to that plane
    - It is perpendicular to any vectors that lie on the plane



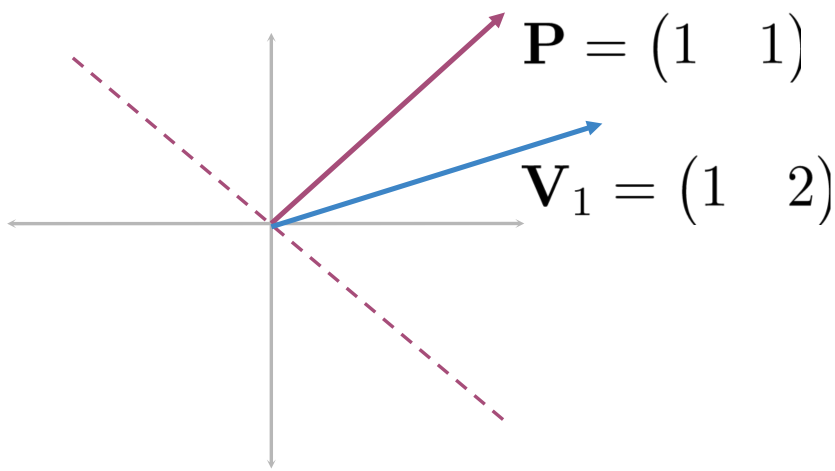
Finding the side of the plane

- How do we find the side of the plane where a vector lie, mathematically?



Side of the Plane

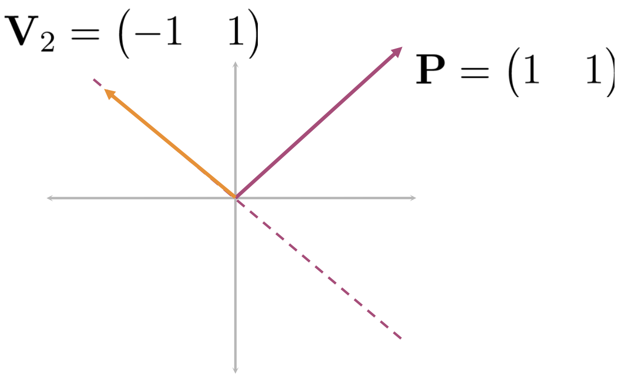
- Let's focus on vector V_1
  - Consider the dot product



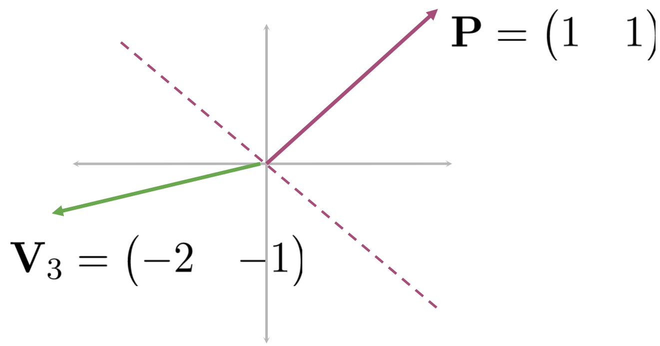
$$\mathbf{P}\mathbf{V}_1^T = 3$$

Side of the Plane

- Now, consider the vectors V_2 and V_3

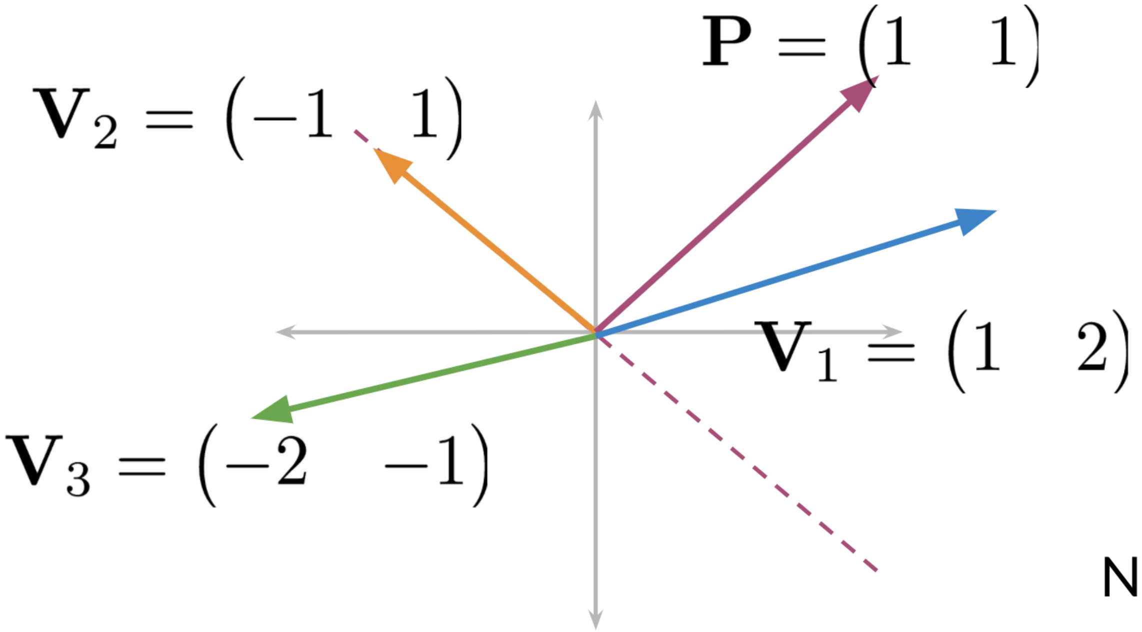


$$\mathbf{P}\mathbf{V}_2^T = 0$$



$$\mathbf{P}\mathbf{V}_3^T = -3$$

Side of the Plane



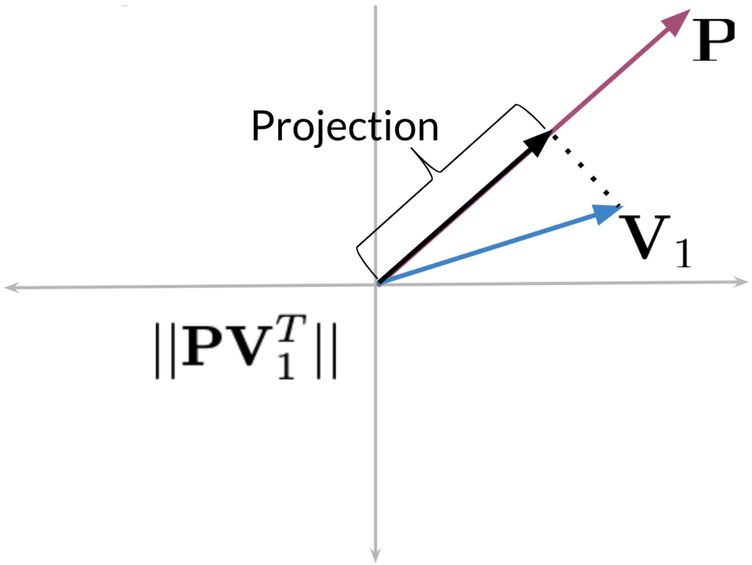
$$\mathbf{P}\mathbf{V}_1^T = 3$$
$$\mathbf{P}\mathbf{V}_2^T = 0$$
$$\mathbf{P}\mathbf{V}_3^T = -3$$

Notice the signs?

- When the dot product is positive, the vector is on one side of the plane
- If the dot product is negative, the vector is on the opposite side of the plane
- If the dot product is zero, the vector is on the plane

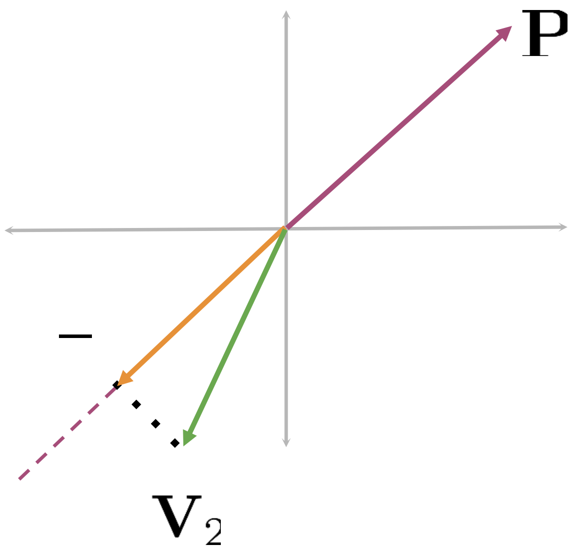
Visualizing a dot product

- Consider the plane represented by vector P
  - The dot product between P and V_1 is a positive number
    - It's the length of the projection of V_1 onto P



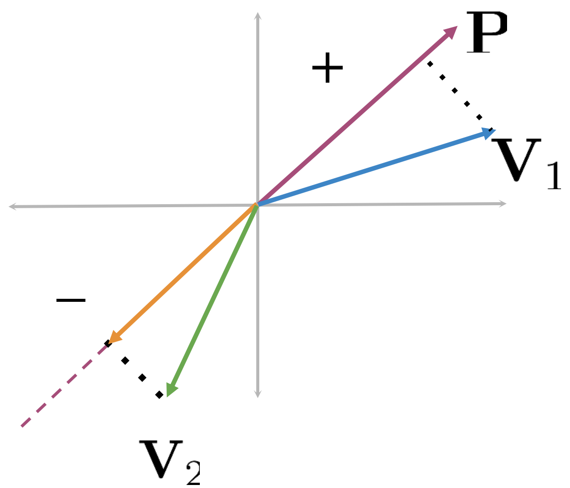
Visualizing a dot product

- The green vector projected onto P, points on the parallel and opposite direction of P
  - The dot product is a negative number



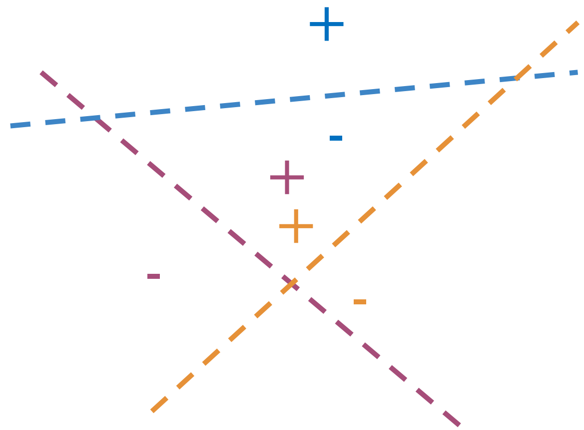
Visualizing a dot product

- The sign of the dot product indicates the direction of the projection with respect to the normal vector
  - If it is positive or negative tells us whether the vectors V_1 or V_2 are on one side of the plane or the other
    - The sign indicates the direction

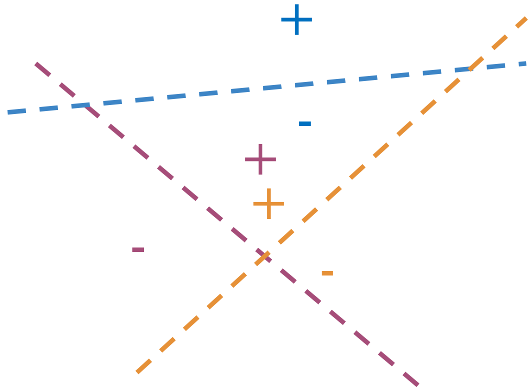


Multiple planes

- How do we get a single hash value from multiple planes?
  - i.e., identify where a data point is given several planes
- We aim at dividing the vector space into manageable regions
  - Goal: determining a single hash value identifying a particular region within the vector space



Multiple planes, single hash value



$$\boxed{\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1}$$

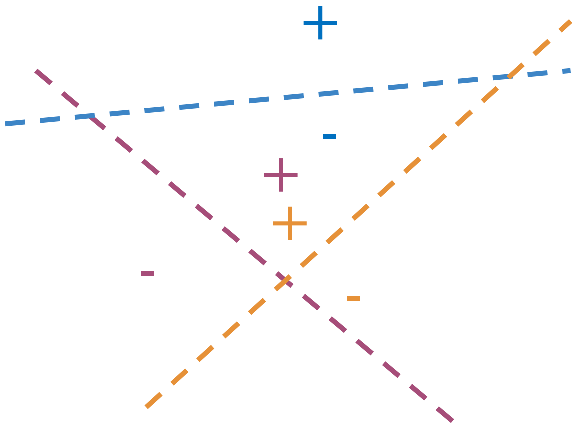
$$\boxed{\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1}$$

$$\boxed{\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0}$$

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$
$$= 1 \times 1 + 2 \times 1 + 4 \times 0$$
$$= 3$$

Multiple planes, single hash value

- Generalizing
  - H = number of planes

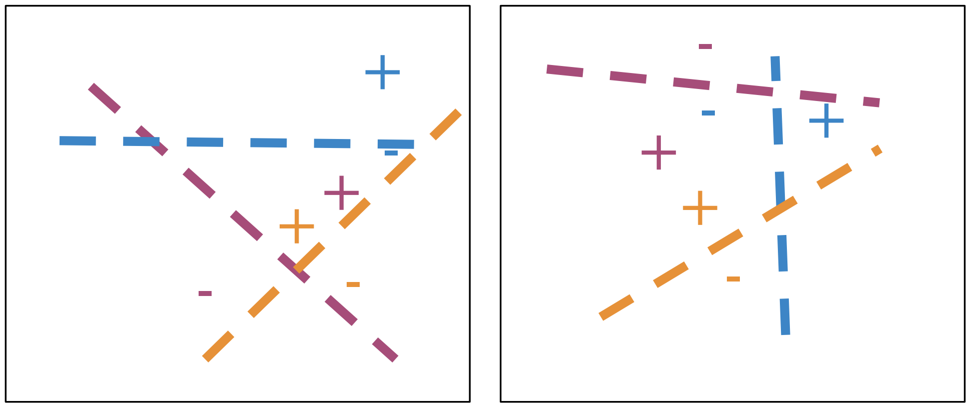


$$sign_i \geq 0, \rightarrow h_i = 1$$
$$sign_i < 0, \rightarrow h_i = 0$$

$$\text{hash} = \sum_{i}^{H} 2^i \times h_i$$

Approximate K-NN

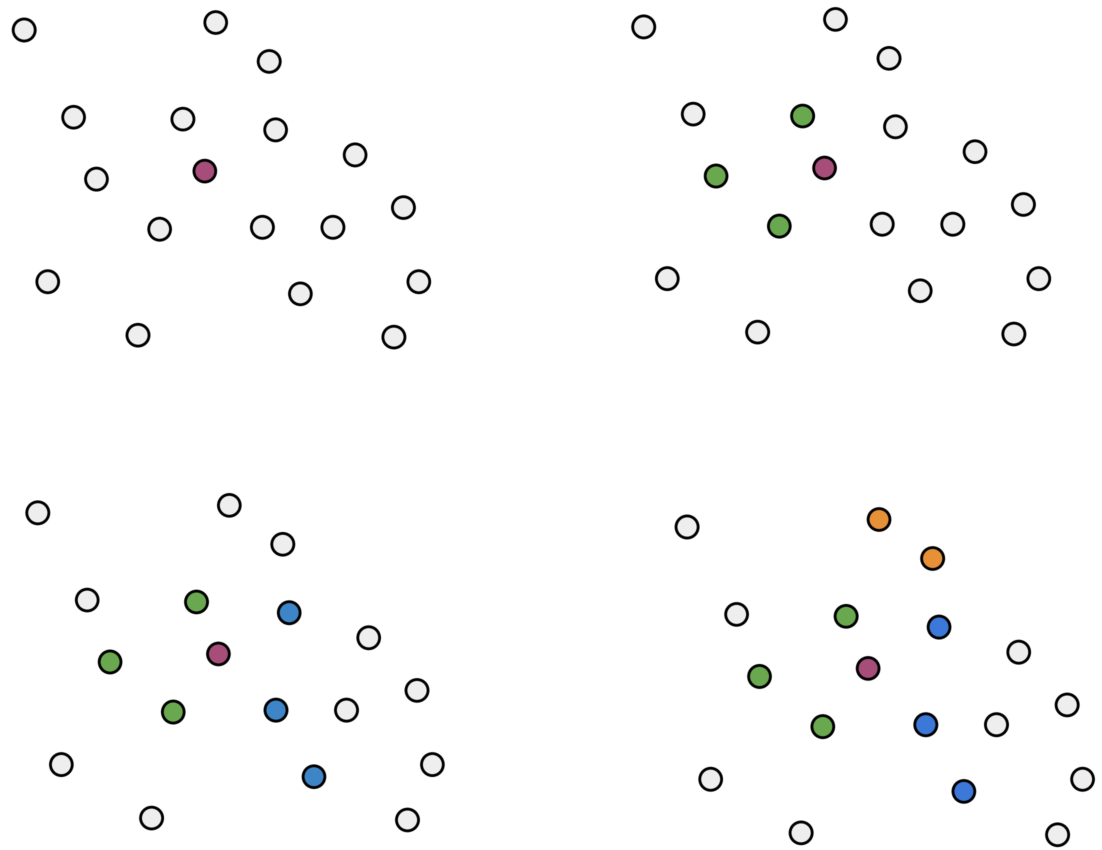
- Multiple sets of planes for approximate NN
  - Random planes



Is this one the best? or Is this one the best?

  - Idea: Create multiple sets of random planes
    - Multiple independent sets of hash tables

Multiple sets of random planes



Approximate nearest neighbors

Searching documents

Document representation

- For document search, the first task is how to represent documents as vectors instead of words as vectors

I love learning!	[?, ?, ?]
I	[1, 0, 1]
	+
love	[-1, 0, 1]
	+
learning	[1, 0, 1]
	=
I love learning!	[1, 0, 3]

- Document search
  - K-NN

Summarizing

- Transform vector
- K nearest neighbors
- Hash tables
- Divide vector space into regions
- Locality sensitive hashing
- Approximated nearest neighbors

Machine translation

Document search