Introduction to Information Retrieval

Session 8: **Term Weighting and Vector Space**

Instructor: Behrooz Mansouri
Fall 2022, University of Southern Maine

Previous Session

In previous session we learned about:

- ✓ Course Project Part 1
- ✓ Daniel Lawrence gave us a talk about using Library and how librarian use search

Ranked Retrieval

Ranked Retrieval

In Boolean retrieval models, we saw documents either match or don’t

Good for expert users with precise understanding of their needs and the collection

Not good for the majority of users

- Most users incapable of writing Boolean queries
- Most users don’t want to wade through 1000s of results
- This is particularly true of web search

Scoring as the basis of ranked retrieval

We wish to rank documents that are more relevant higher than documents that are less relevant

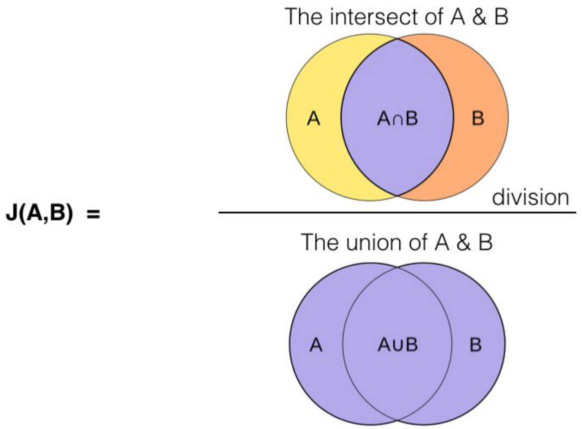
- Assign a score to each query--document pair, say in [0, 1]
- This score measures how well document and query “match”

How can we do this?

Jaccard Coefficient

A commonly used measure of overlap of two sets

- Let A and B be two sets
- Jaccard coefficient: $|A \cap B| / |A \cup B|$



JACCARD (A, A) = 1

JACCARD (A, B) = 0 if $A \cap B = 0$

A and B don’t have to be the same size

Always assigns a number between 0 and 1

What’s wrong with Jaccard?

- It doesn’t consider term frequency

  (how many occurrences a term has)
- Rare terms are more informative than frequent terms
  - Jaccard does not consider this information
  - We need to consider both

Term Frequency

- The term frequency $tf_{t,d}$ of term t in document *d* is defined as the number of times that *t* occurs in *d*
- We can use tf when computing query-document match scores
- $tf_{t,d}$ can be normalized by the document length $$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$
- Sometimes, we may refine the raw term frequency:
  - A document with 10 occurrences of the term is more relevant than a document with one occurrence of the term
  - But not 10 times more relevant
- Relevance does not increase proportionally with term frequency

Log-Frequency Weighting

Sometimes, the log frequency weight of term *t* in *d* is used:

$$w_{t,d} = \begin{cases} 1 + \log tf_{t,d} & , \quad \text{if } tf_{t,d} > 0 \\ 0 & , \quad \text{otherwise} \end{cases}$$

Score for a document-query pair; sum over terms t in both q and d:

$$\text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

The contribution of term frequency to document relevance is essentially a sub-linear function

Document Frequency

Rare terms are more informative than frequent terms

- Recall stop words

Consider a term in the query that is rare in the collection (e.g., serendipity)

- A document containing this term is very likely to be relevant to the query ‘serendipity film’
- We want a high weight for rare terms like serendipity

Document Frequency

Consider a query term that is frequent in the collection (e.g., high, increase, line, film)

A document containing such a term is more likely to be relevant than a document that does not, but it is not a sure indicator of relevance

- Just term frequency is not sufficient
- We wish to capture the notion of rare terms

Inverse Document Frequency (IDF)

df_t (document frequency) = the number of documents that contain the term t

idf_t = inverse document frequency of term t ($\log_2 (N / df_t)$)

- N is the total number of documents
- This is an indication of a term's discrimination power

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

TF-IDF

TF.IDF Weighting

A typical combined term importance indicator is tf-idf weighting:

$$w_{t,d} = tf_{t,d} \times idf_t = (1+\log(tf_{t,d})) \times \log_2 (N/ df_t)$$

IDF: TERM
TF: Document, TERM

$$\text{Score}(d,q) = \sum_{t \in q \cap d} \text{tf}_{t,d} \times \text{idf}_t$$

- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection
- <u>Many other ways of determining term weights</u> have been proposed
- Experimentally, tf-idf has been found to work well

Example: Use TF-IDF Model and Rank the Documents

Q: The cat.

D1: The cat is on the mat.

D2: My dog and cat are the best.

D3: The locals are playing.

Example: Use TF-IDF Model and Rank the Documents

Q: The cat.

D1: The cat is on the mat.

D2: My dog and cat are the best.

D3: The locals are playing.

TF(“the”, D1) = 2/6 = 0.33

TF(“the”, D2) = 1/7 = 0.14

TF(“the”, D3) = 1/4 = 0.25

TF(“cat”, D1) = 1/6 = 0.17

TF(“cat”, D2) = 1/7 = 0.14

TF(“cat”, D3) = 0/4 = 0

IDF(“the”) = log(3/3) = log(1) = 0

IDF(“cat”) = log(3/2) = 0.18

TF-IDF(“the”, D1) = 0.33 * 0 = 0

TF-IDF(“the, D2) = 0.14 * 0 = 0

TF-IDF(“the”, D3) = 0.25 * 0 = 0

TF-IDF(“cat”, D1) = 0.17 * 0.18= 0.0306

TF-IDF(“cat, D2) = 0.14 * 0.18= 0.0252

TF-IDF(“cat”, D3) = 0 * 0 = 0

Example: Use TF-IDF Model and Rank the Documents

Q: The cat.

D1: The cat is on the mat.

D2: My dog and cat are the best.

D3: The locals are playing.

TF-IDF(“the”, D1) = 0.33 * 0 = 0

TF-IDF(“the, D2) = 0.14 * 0 = 0

TF-IDF(“the”, D3) = 0.25 * 0 = 0

TF-IDF(“cat”, D1) = 0.17 * 0.18= 0.0306

TF-IDF(“cat, D2) = 0.14 * 0.18= 0.0252

TF-IDF(“cat”, D3) = 0 * 0 = 0

TF-IDF score for D1 = 0 + 0.0306 = 0.0306

TF-IDF score for D2 = 0 + 0.0252 = 0.0252

TF-IDF score for D3 = 0 + 0 = 0

Vector Space Model (VSM)

Weight Matrix with TF-IDF Weights

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in R^{|V|}$

Document as Vectors

We have a |V|-dimensional vector space

- Terms are axes of the space
- Documents are points or vectors in this space

Very **high-dimensional**: hundreds of millions of dimensions when you apply this to a web search engine

This is a very sparse vector - most entries are zero

Queries as Vectors

Key idea 1: do the same for queries

Represent them as vectors in the high-dimensional space

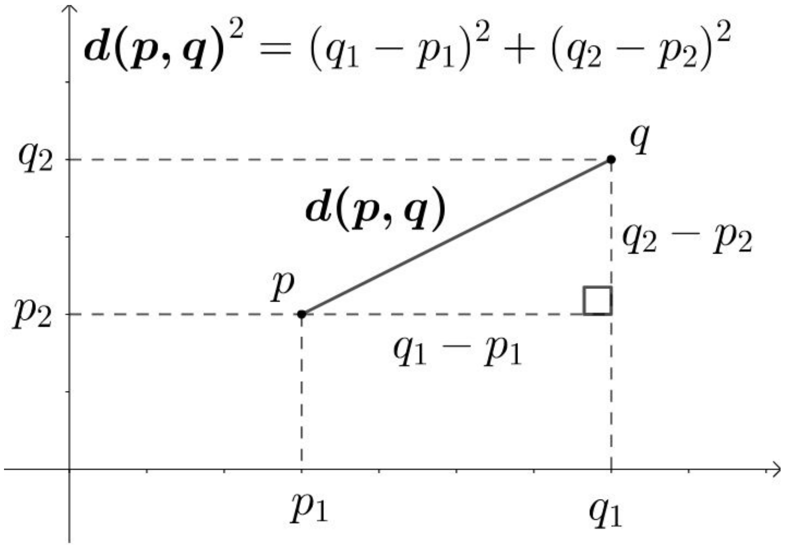
Key idea 2: Rank documents according to their proximity to the query

Ranking relevant documents higher than non-relevant documents

- proximity = similarity of vectors
- proximity ≈ inverse of distance

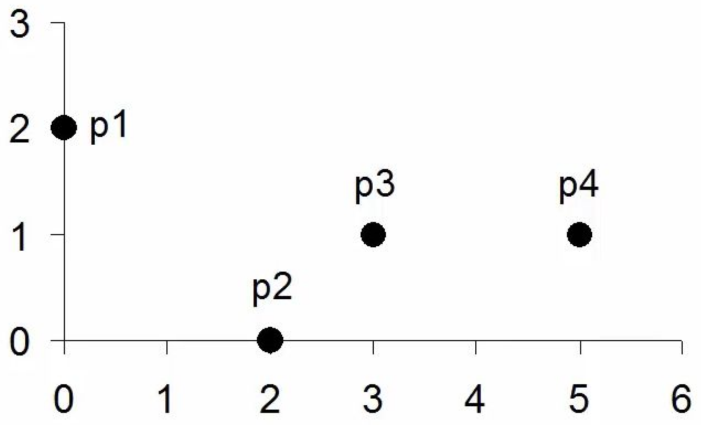
Euclidean Distance

Two-dimensional Example



$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}.$$

Euclidean Distance Example

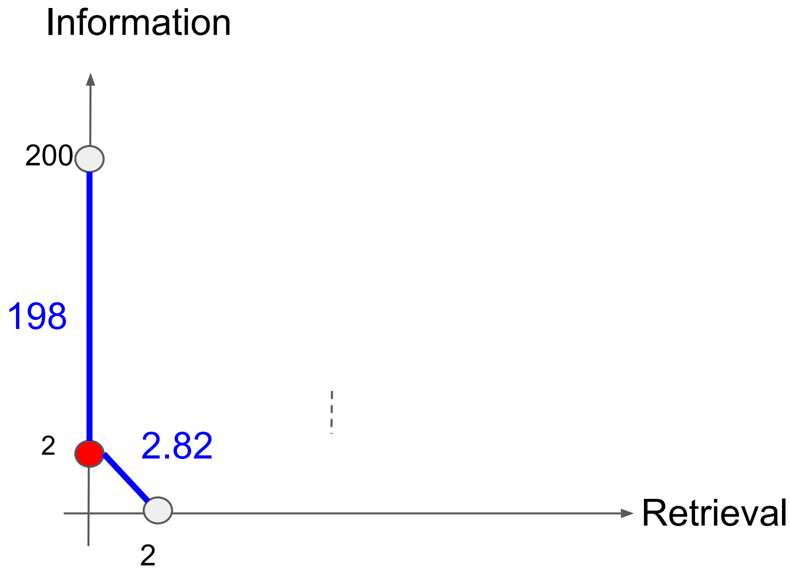


point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

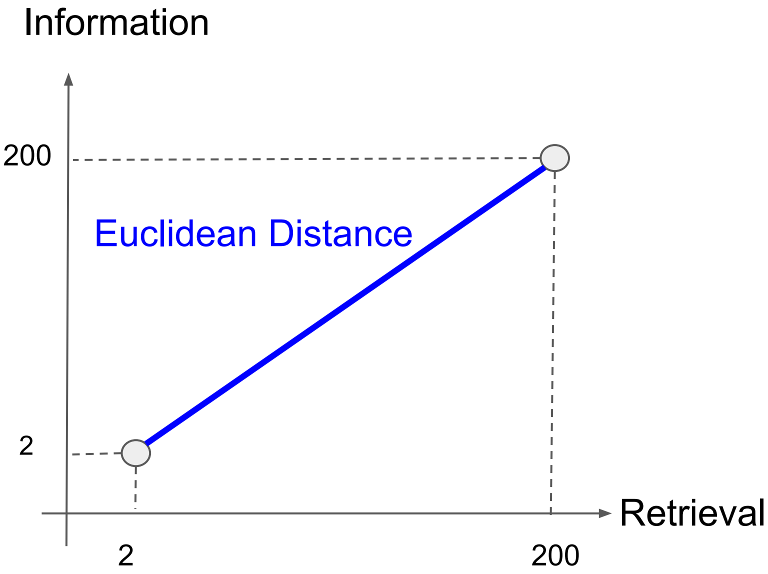
Distance Matrix

Euclidean Distance



- Consider Query “Information Information”
  - D1: “Retrieval Retrieval”
  - D2: “Information … Information” (200 times)
    - Concatenate a document to itself

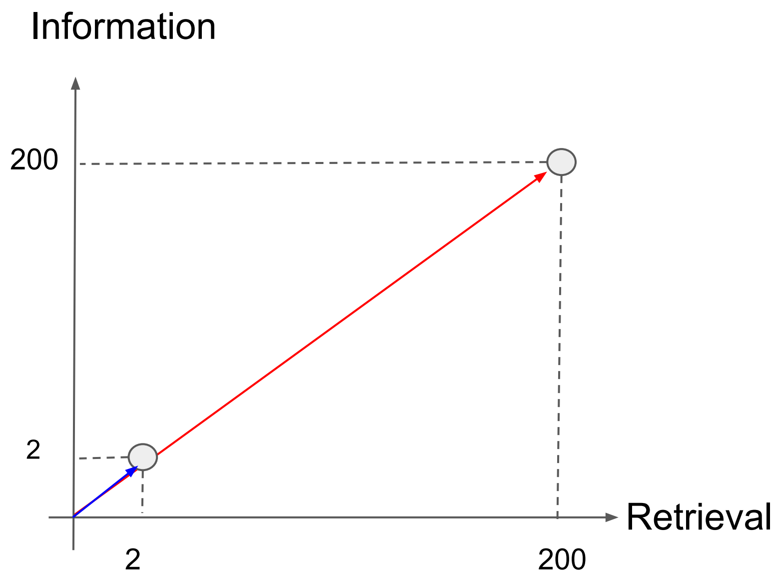
Euclidean Distance



Euclidean Distance is large for vectors of different lengths
Take a document d and append it to itself. Call this document d'

- “Semantically” d and d' have the same content
- The Euclidean distance between d and d’can be quite large

Cosine Similarity

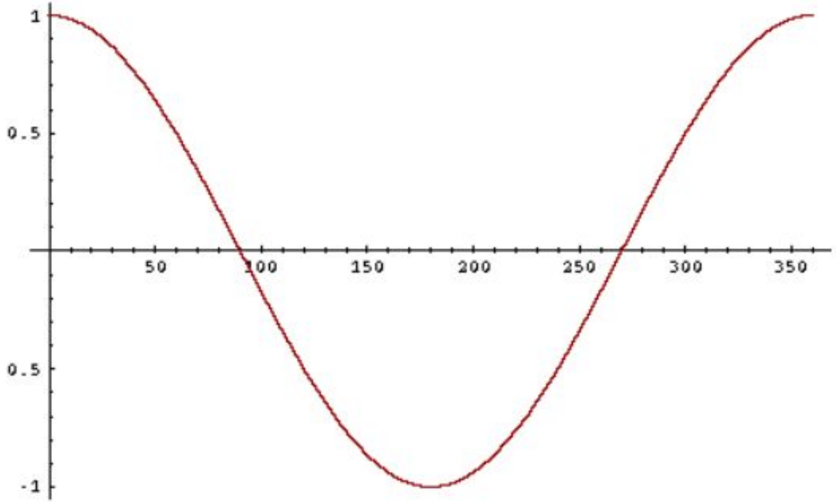


The angle between the two documents is 0
Rank the documents by angle between term weight vectors
Cosine correlation or Cosine Similarity

Cosine Similarity

The following two notions are equivalent

- Rank documents in decreasing order of the angle between query and document
- Rank documents in increasing order of cosine(query,document)



How do we compute the cosine?

Length Normalization

Document lengths vary in many collections

Long documents have an unfair advantage

- They use a lot of terms
  - So they get more matches than short documents
- They use the same terms repeatedly
  - So they have much higher term frequencies

Two strategies

- Adjust term frequencies for document length
- Divide the documents into equal “passages”

Length Normalization in Cosine Similarity

A vector can be (length-) normalized by dividing each of its components by its length – for this, we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

– Multiply each weight by itself
– Add all the resulting values
– Take the square root of that sum

only changes its magnitude, not its direction

Dividing a vector by its L_2 norm makes it a unit (length) vector

- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have identical vectors after length-normalization
- Long and short documents now have comparable weights

Cosine Similarity (Query, Document)

$$\cos(\vec{q}, \vec{d}) = \mathrm{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the tf-idf weight of term i in the query.

d_i is the tf-idf weight of term i in the document.

$|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.

This is the cosine similarity of $\vec{q}$ and $\vec{d}$ . . . . . . or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Cosine for Normalized Vectors

For normalized vectors, the cosine is equivalent to the dot product or scalar product

$$\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$$

(if $\vec{q}$ and $\vec{d}$ are length-normalized)

Vector Space Model

1. Represent the query as a weighted tf-idf vector

2. Represent each document as a weighted tf-idf vector

3. Compute the cosine similarity score for the query vector and each document vector

4. Rank documents with respect to the query by score

5. Return the top K (e.g., K = 10) to the user

Different Weighting Queries VS. Documents

Different weightings for queries and documents

Using notation: ddd.qqq

Example: lnc.ltn

Document:

- l ogarithmic tf
- n o df weighting
- c osine normalization

Query:

- l ogarithmic tf
- t – means idf
- n o normalization

SMART

System for the Mechanical Analysis and Retrieval of Text

Example lnc.lt**c**

Query: “good news”
Document: “good news bad news”
In the table, log tf is the tf weight based on log-frequency weighting,
q is the query vector, q’ is the length-normalized q, d is the document vector, and d’ is the length-normalized d. Assume N=10,000,000

		Query					Document			
terms	df	idf	tf	log tf	q	q’	tf	log tf	d	d’
bad	10000	3	0	0	0	0	1	1	1	0.52
good	100000	2	1	1	2	0.45	1	1	1	0.52
news	1000	4	1	1	4	0.89	2	1.3	1.3	0.68

Example: lnc.ltn

Document:
- l ogarithmic tf
- n o df weighting
- c osine normalization

Query:
- l ogarithmic tf
- t – means idf
- n o normalization

The cosine similarity between q and d is the dot product of q’ and d’,
which is: 0x0.52 + 0.45x0.52 + 0.89x0.68 = 0.84

Example lnc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Doc length $= \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$

Score = 0 + 0 + 1.04 + 2.04

Example: lnc.ltn

Document:

- l ogarithmic tf
- n o df weighting
- c osine normalization

Query:

- l ogarithmic tf
- t – means idf
- n o normalization

Let’s Code Assignment 3!

TF-IDF

- Implement a function for TF (doc) → return dictionary
  - You will update the main index for each term
  - Term: (Doc:TF)
- Implement a function for IDF(docList) → return dictionary
  - Term: IDF
- Implement a function that takes in query and return TF-IDF score

Vector Space Model

- Use your previous code to get TF*IDF score for each term in documents
- Get vector representation for each document
  - Index time: each doc is a vector of n-dimension, n=#terms
- Get vector for each query
- Use numpy Vectors and for cosine similarity
  - from numpy import dot
  - from numpy.linalg import norm
  - cos_sim = dot(a, b)/(norm(a)*norm(b))
- For assignment follow lnc.ltc


WHAT HAVE
YOU LEARNED?

Summary

Today we learned about:

✓ TF-IDF Model

✓ VSM Rank Model

✓ How to use TF-IDF and VSM in Python

Next Session

Building Test Collections for IR

In the next class, we will talk about building test collections

On September 29th Dr. Michael Ekstrand
https://md.ekstrandom.net/

TREC 2021 Fair Ranking Track

https://fair-trec.github.io/2021/

To do:

- Reading: Chapters 6 of Manning's book (No need for 6.4.4)
- Submit Assignment 2