

WEB INTELLIGENCE

Lecture 3: Text Processing I

Russa Biswas

September 16, 2025

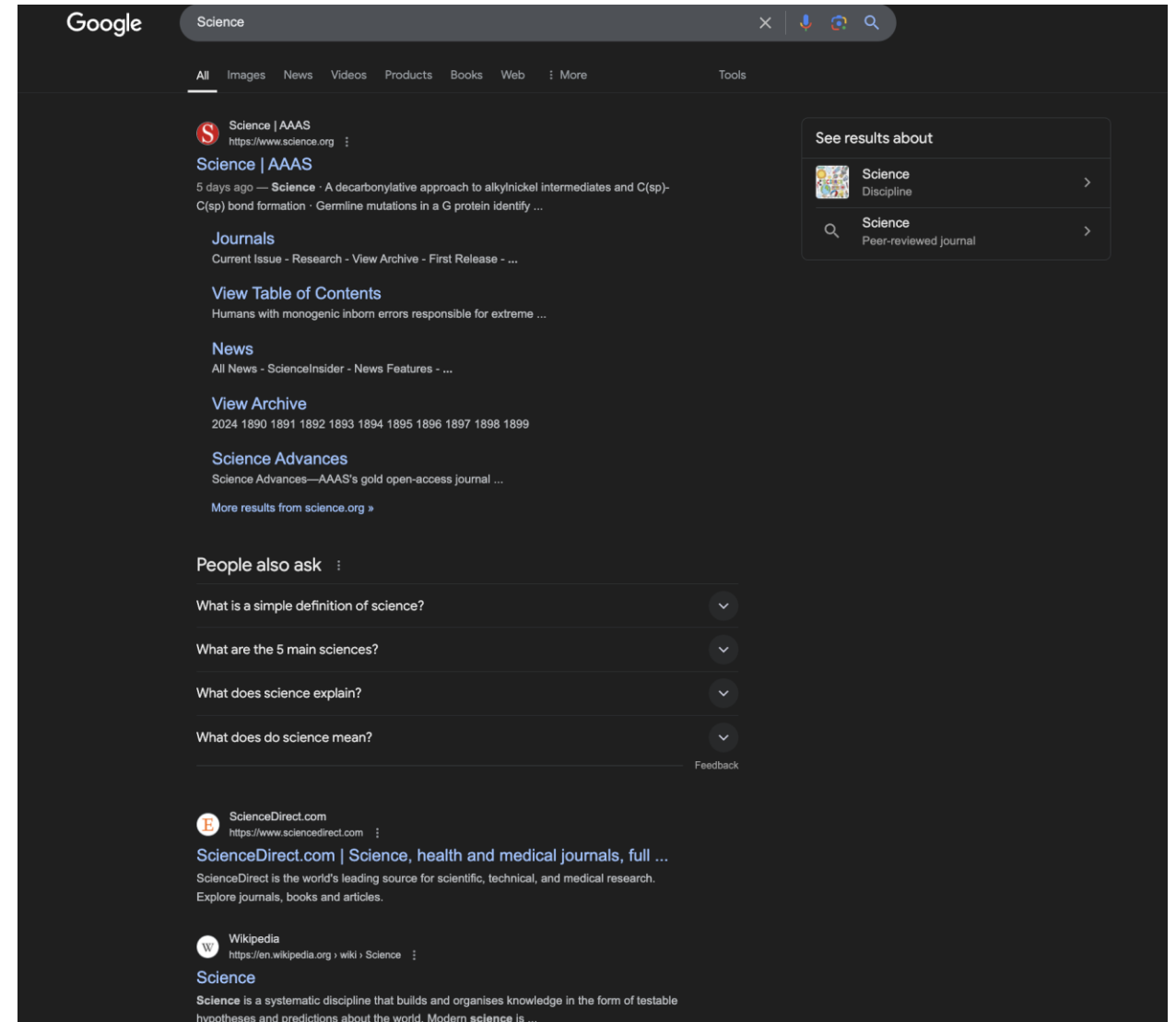


AALBORG UNIVERSITY



What is Text Similarity?

- Word Similarity
- String Similarity
- Query and Document similarity
- Document Similarity



String Similarity

Hamming distance: between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different.

Binary Strings:

- Strings: 11001 and 10110
- Hamming Distance: 4
- **Text Strings:**
- Strings: Base and Case
- Hamming Distance: 1 (differ at position 1)

It is useful for **detecting and correcting errors in data transmission over networks, comparing DNA sequences, or identifying similarities between cryptographic keys**

String Similarity

Levenshtein distance measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another.

Strings: kitten and sitting

kitten → sitten (substitute k with s)

sitten → sittin (substitute e with i)

sittin → sitting (insert g at the end)

Levenshtein Distance: 3

Strings: flaw and lawn

flaw → law (delete f)

law → lawn (insert n at the end)

Levenshtein Distance: 2

It is useful for tasks such as **spell-checking, fuzzy search, DNA sequence analysis, plagiarism detection, and data cleaning** where **minor typos, variations, or errors**

String Similarity

Jaccard Similarity is a statistic to determine the similarity or the overlap between the two sets

$$\text{Jaccard Similarity: } \frac{A \cap B}{A \cup B}$$

where A and B are two sets, $A \cap B$ is the intersection of two sets (common items), and $A \cup B$ is the union of two sets (all unique elements)

Example:

String 1: "Cats are wonderful pets"

Set A: {Cats, are, wonderful, pets}

String 2: "Dogs are wonderful companions"

Set B: {Dogs, are, wonderful, companions}

$A \cap B = \{\text{are, wonderful}\}$

$A \cup B = \{\text{Cats, are, wonderful, pets, Dogs, companions}\}$

$$\text{Jaccard Similarity: } \frac{A \cap B}{A \cup B} = 2/6 = 0.33$$

Jaccard Distance: dissimilarity between the two strings

$$\begin{aligned} \text{Jaccard Distance} &= 1 - \frac{A \cap B}{A \cup B} \\ &= 1 - 0.33 = 0.67 \end{aligned}$$

Interpretation:

A Jaccard Distance of 0.6667 means the two sets are moderately dissimilar, with some overlap in the words "are" and "wonderful."

It is used in **comparing text documents, user preferences in recommendation systems, network analysis**, etc.

Scoring with Jaccard Coefficient

Jaccard (A,B) = ?

$$\text{Jaccard (A,B)} = \frac{A \cap B}{A \cup B}$$

Jaccard (A,A) = ?

Jaccard (A,A) = 1

Jaccard (A,B) = 0 (when?)

Jaccard (A,B) = 0, when $A \cap B = 0$

A and B can be of different sizes

Always assigns a number between 0 and 1

Query = “march”

S1 = “Memories in March is a good movie”

S2 = “Flowers start blooming in March”

Jaccard(Query, S1) = ?

Jaccard(Query, S2) = ?

Jaccard(Query, S1) = 1/7

Jaccard(Query, S2) = 1/5

Limitations of Jaccard Similarity

- Term frequency (no. of occurrences of a term) is not considered
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information
 - For e.g., adjectives such as good, beautiful, etc.
- Order of the terms not considered

Text Similarity from Unstructured Data

Unstructured data in 1620

- Which plays of Shakespeare contain the words ***Brutus*** ***AND Caesar*** but ***NOT Calpurnia***?
- One could `grep` all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia***?
- Why is that not the answer?
 - Slow (for large corpora)
 - ***NOT Calpurnia*** is non-trivial
 - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
 - Ranked retrieval (best documents to return)
 - Later lectures

Ranked Retrieval Model

- In a **ranked retrieval model**, the system returns an ordering over the (top) documents in the collection for a query
 - More relevant results are **ranked higher** than less relevant ones
 - Queries are in form of **one or more words** in natural language text (human languages) and not query languages
 - Large result set is not an issue in ranked retrieval systems, usually top $k \approx 10$ results are returned
 - The output documents (results) are ranked according to how relevant they are to a query, i.e., how well document and query “match”
 - A score – say in $[0, 1]$ – is assigned to each document

For e.g., query term is “Aalborg”

- Score =0, if “Aalborg” does not occur in the document
- More frequent the occurrence of “Aalborg” in the document, higher the score.

Binary Term-Document Incidence Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t , and is 0 otherwise

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Julius Caesar: 1111000

Binary Term-Document Incidence Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t , and is 0 otherwise

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Query Brutus AND Caesar AND NOT Calpurnia

= Brutus 110100, Caesar 110111, NOT(Calpurnia) 101111 (bitwise AND)

= 110100 AND 110111 AND 101111 = **100100**

Count Vector

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Consider the number of occurrences of a term in a document:

Each document is a **count vector** (i.e., each column) $\in \mathbb{N}^{|V|}$

Bag of Words Model

Vector representation doesn't consider the ordering of words in a document

Let's consider a small dataset of three sentences:

1. **Sentence 1:** "I like programming."
2. **Sentence 2:** "Programming is fun."
3. **Sentence 3:** "I like to code."

Vocabulary: ["I", "like", "programming", "is", "fun", "to", "code"]

Sentence 1: "I like programming." **Vector:** [1, 1, 1, 0, 0, 0, 0]

Sentence 2: "Programming is fun." **Vector:** [0, 0, 1, 1, 1, 0, 0]

Sentence 3: "I like to code." **Vector:** [1, 1, 0, 0, 0, 1, 1]

Represents text as a collection of word counts, enabling the analysis and processing of text data while ignoring the order of words

Bag of Words

Sentence 1: John runs faster than Mary.

Sentence 2: Mary runs faster than John.

Generate the vectors

Vocabulary: ["John", "runs", "faster", "than", "Mary"]

- **Sentence 1:** "John runs faster than Mary."

- Vector: [1, 1, 1, 1, 1]

- **Sentence 2:** "Mary runs faster than John."

- Vector: [1, 1, 1, 1, 1]

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** .

Raw term frequency is not what we want:

- A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
- But not 10 times more relevant

Relevance does not increase proportionally with term frequency.

$$tf(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **WARNING:** In a lot of IR literature, “frequency” is used to mean “count”
 - Thus *term frequency* in IR literature is used to mean *number of occurrences* in a doc
 - Not divided by document length (which would actually make it a frequency)

Log-Frequency Weighting

The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d}, & \text{if } \text{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$\text{tf}_{t,d}$	0	1	2	10	1000
$w_{t,d}$	0	1	1.3	2	4

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

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

The score is 0 if none of the query terms is present in the document.

Log-Frequency Weighting

- **Sentence 1:** "The dog barks loudly."
- **Sentence 2:** "The cat meows softly."
- Compute Log-Frequency Weighting

Log-Frequency Weighting

Sentence 1: "The dog barks loudly."

Sentence 2: "The cat meows softly."

Compute Log-Frequency Weighting

Sentence 1: "The dog barks loudly." → Vector: [1, 1, 1, 1, 0, 0, 0]

Sentence 2: "The cat meows softly." → Vector: [1, 0, 0, 0, 1, 1, 1]

Document Frequency

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high*, *increase*, *line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
- → For frequent terms, we want high positive weights for words like *high*, *increase*, and *line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

Document Frequency

- df_t is the document frequency of t : the number of documents that contain t
 - df_t is an inverse measure of the informativeness of t
 - $df_t \leq N$
- We define the idf (inverse document frequency) of t by

$$idf_t = \log_{10} (N/df_t)$$

- We use $\log (N/df_t)$ instead of N/df_t to “dampen” the effect of idf.
- It is a measure of the informativeness of the term

Inverse Document Frequency - Impact

- Does idf have an effect on ranking for one-term queries, like
 - iPhone
- idf has no effect on ranking one term queries
 - idf affects the ranking of documents for queries with at least two terms
 - For the query **capricious person**, idf weighting makes occurrences of **capricious** count for much more in the final document ranking than occurrences of **person**.

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

N = 1,000,000

Inverse Document Frequency - Impact

- Does idf have an effect on ranking for one-term queries, like
 - iPhone
- idf has no effect on ranking one term queries
 - idf affects the ranking of documents for queries with at least two terms
 - For the query **capricious person**, idf weighting makes occurrences of **capricious** count for much more in the final document ranking than occurrences of **person**.

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

N = 1,000,000

TF-IDF

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = \log(1 + \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

- Best known weighting scheme in information retrieval
 - Note: the “-” in tf-idf is a hyphen, not a minus sign!
 - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection
- Score of a document given a query q is $\text{Score}(q, d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$

TF-IDF

	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 $\in \mathbb{R}^{|V|}$

TF-IDF Variants

$$tf(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

Term frequency		Document frequency	
n (natural)	$tf_{t,d}$	n (no)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - df_t}{df_t}\}$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$		

Vector Similarity in Information Retrieval – Use Case

- **Threshold**

- For query q , retrieve all documents with similarity above a threshold, e.g., $\text{similarity} > 0.50$.

- **Ranking**

- For query q , return the n most similar documents ranked in order of similarity.

Documents and Queries as Vectors

- Documents
 - So we have a $|V|$ -dimensional vector space
 - Terms are axes of the space
 - Documents are points or vectors in this space
 - Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
 - These are very sparse vectors - most entries are zero.
- Queries
 - [Key idea 1](#): Do the same for queries: represent them as vectors in the space
 - [Key idea 2](#): Rank documents according to their proximity to the query in this space
 - proximity = similarity of vectors
 - proximity \approx inverse of distance
 - Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model.
 - Instead: rank more relevant documents higher than less relevant documents

Vector Spaces

Distance between distance between the end points of the two vectors: Euclidean distance

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

String 1: "dog barks"

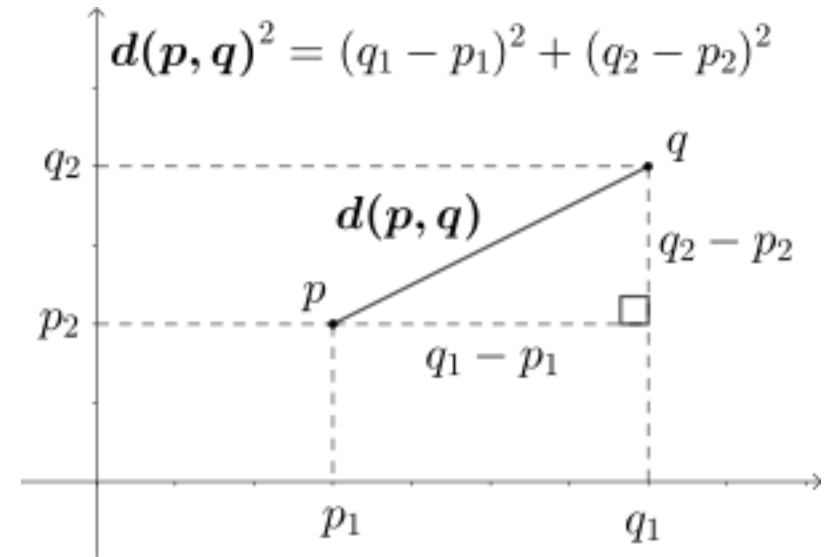
String 2: "cat meows"

Vocabulary = ["dog", "barks", "cat", "meows"]

String 1 ("dog barks") \rightarrow [1, 1, 0, 0]

String 2 ("cat meows") \rightarrow [0, 0, 1, 1]

$$\begin{aligned}\text{Euclidean Distance} &= \sqrt{(1 - 0)^2 + (1 - 0)^2 + (0 - 1)^2 + (0 - 1)^2} \\ &= \sqrt{4} = 2\end{aligned}$$



Vector Spaces

Euclidean distance is **large** for vectors of **different lengths** – Major Drawback

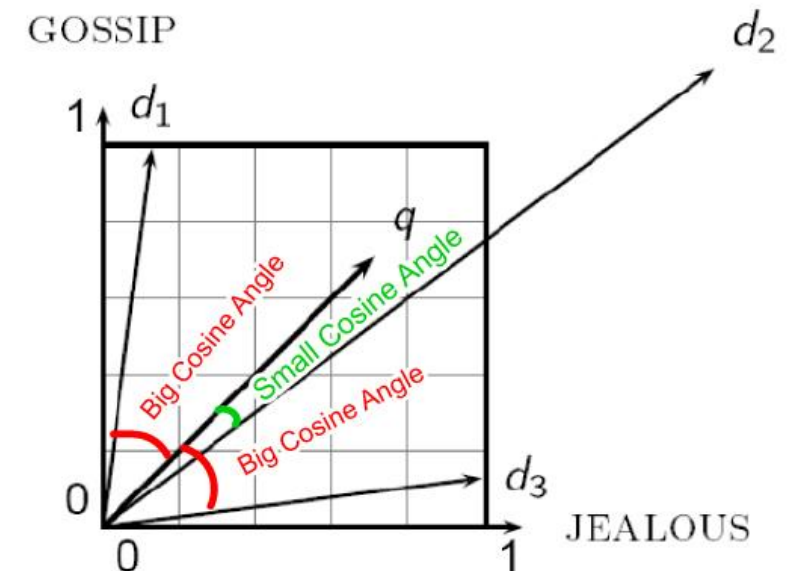
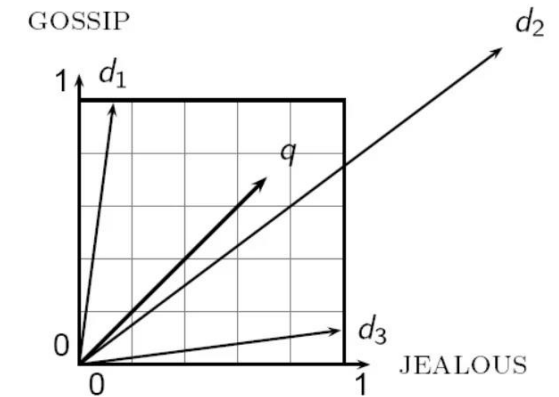
The angle between the two documents is 0, corresponding to maximal similarity

Rank results based on angle between the document and query

Cosine is a monotonically decreasing function for the interval $[0^\circ, 180^\circ]$

- Rank documents in decreasing order of the angle between query and document
- Rank documents in increasing order of $\cos(\text{query}, \text{document})$

Cosine similarity - Example in 2d



Cosine Similarity

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

Dot product

Unit vectors

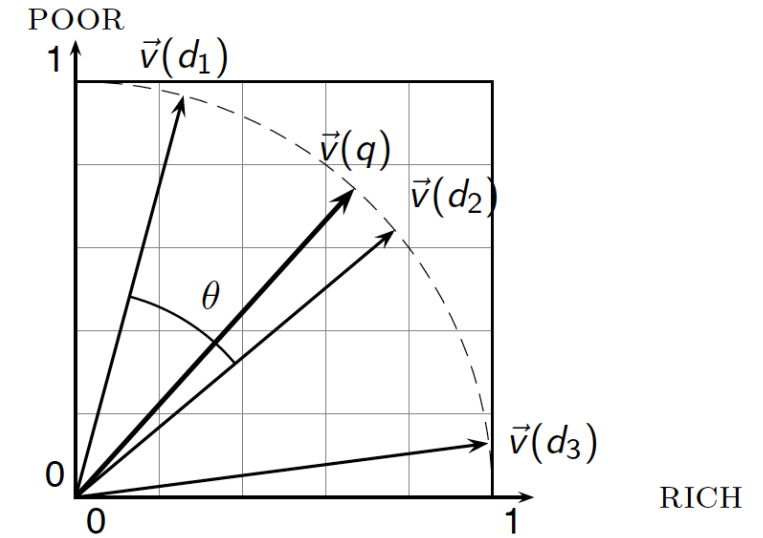
q_i is the tf-idf weight of term i in the query
 d_i is the tf-idf weight of term i in the document

$\cos(q, d)$ is the cosine similarity of q and d or,
equivalently, the cosine of the angle between q and d .

For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos(\vec{q}, \vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for q, d length-normalized



Vector Spaces

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

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

Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)

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 - Example

term	Sense and Sensibility	Pride and Prejudice	Wuthering Heights
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

How similar are the novels?

Note: To simplify this example, we don't do idf weighting.

Cosine Similarity - Example

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

$\cos(\text{SaS}, \text{PaP}) \approx$

$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$

≈ 0.94

$\cos(\text{SaS}, \text{WH}) \approx 0.79$

$\cos(\text{PaP}, \text{WH}) \approx 0.69$

Implication of the result?

Vector Space Ranking - Summary

Represent the query as a weighted tf-idf vector

Represent each document as a weighted tf-idf vector

Many search engines allow for different weightings for queries vs. documents

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

Rank documents with respect to the query by score

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

Literature

- <https://www.cse.iitd.ac.in/~mausam/courses/csl772/autumn2014/lectures/14-ir.pdf>
- <https://courses.cs.washington.edu/courses/cse573/12sp/lectures/17-ir.pdf>
- [Christopher D. Manning](#), [Prabhakar Raghavan](#) and [Hinrich Schütze](#), *Introduction to Information Retrieval*, Cambridge University Press. 2008.