Boolean and Vector space retrieval model

19 Nov, 2024

Boolean Model

- one of the simplest IR models, using binary logic to determine whether a document matches a query.

How it works:

Document Representation: Each document is represented as a set of keywords or terms.

Query Formulation:

- Queries are typically expressed as Boolean expressions using operators like AND, OR, and NOT.
- For example, a query could be (AI AND learning) OR (robotics AND NOT neural).

Matching Process:

- A document either satisfies the query (relevant) or does not (irrelevant).
- This model does not provide a ranked list of results; instead, it returns a binary answer (yes or no).

...

- Useful for scenarios where precision and explicit criteria are important, like in legal or medical databases.

- The Boolean model uses **set theory** and **Boolean algebra** to determine whether documents meet query conditions.

...

1. Representing Documents

- **Incidence Matrix**:
  Let $T = \{t_1, t_2, \ldots, t_m\}$ represent the terms in the vocabulary, and $D = \{d_1, d_2, \ldots, d_n\}$ represent the documents.
  The **term-document incidence matrix** is a binary matrix M where:

$$M_{ij} = \begin{cases} 1 & \text{if term } t_i \text{ occurs in document } d_j, \\ 0 & \text{otherwise.} \end{cases}$$

  Example:

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

  Each row represents a term, and each column represents a document.

2. Boolean Query Operations

- **AND (∩)**: Intersection of documents containing all query terms.

$$Q_{\text{AND}} = t_1 \cap t_2 = M_{t_1} \wedge M_{t_2}$$

  Example:
  If $t_1 = [1, 1, 0, 0]$ and $t_2 = [0, 1, 1, 0]$,

$$Q_{\text{AND}} = [1, 1, 0, 0] \wedge [0, 1, 1, 0] = [0, 1, 0, 0].$$

  Document 2 matches.

- **OR (∪)**: Union of documents containing at least one query term.

$$Q_{\text{OR}} = t_1 \cup t_2 = M_{t_1} \vee M_{t_2}$$

  Example:

$$Q_{\text{OR}} = [1, 1, 0, 0] \vee [0, 1, 1, 0] = [1, 1, 1, 0].$$

  Documents 1, 2, and 3 match.

Problem Description:

Goal: Determine which plays contain **Brutus AND Caesar AND NOT Calpurnia.**

Approach:

Manual Linear Scan:

- Start at the beginning of the text.
    - For each play:
    - Check if it contains the words Brutus and Caesar.
    - Exclude the play if it contains Calpurnia.

Computerized Linear Scan:

Use a computer to automate the search.

- Perform a linear scan through the text to identify matching plays.

Use of **grep** Command:

- Grep is a Unix command used for searching text using patterns.

In this context:

Grepping through the text is equivalent to filtering plays based on query criteria.

Modern computers can perform such tasks quickly and efficiently.

…

Limitations:

1. Linear scanning becomes inefficient as the document collection size grows significantly.

2. To allow more flexible matching operations. For example, it is impractical to perform the query Romans NEAR countrymen with grep, where NEAR might be defined as “within 5 words” or “within the same sentence”.

3. Advanced retrieval methods (e.g., inverted indexes) are preferred for large-scale data.

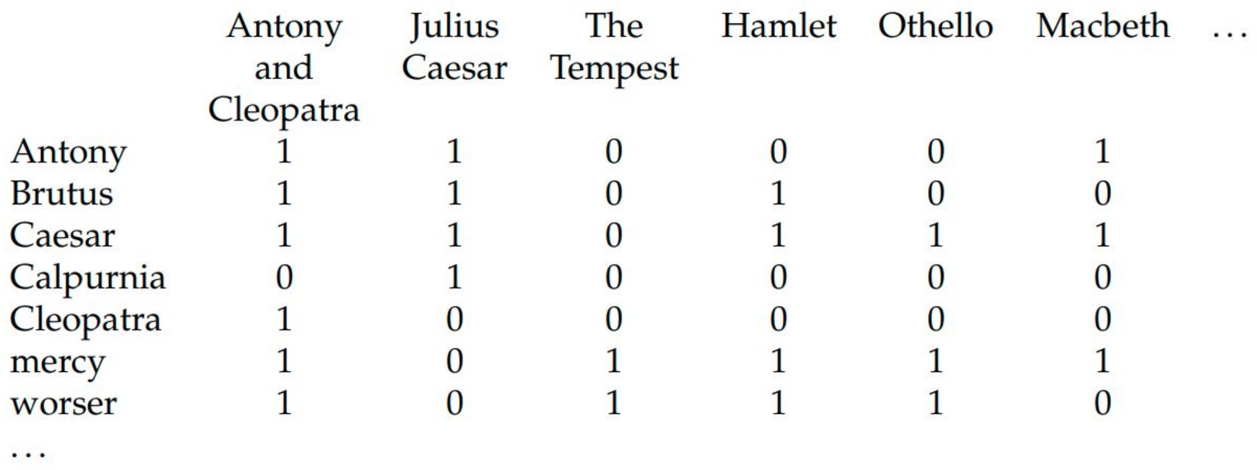
Avoiding Linear Scans: Term-Document Incidence Matrix

Goal: Efficiently retrieve documents without repeatedly scanning the entire text collection for each query.

1. Indexing Documents with a Term-Document Incidence Matrix:

Structure:

- Rows represent terms (words) used in the documents.
- Columns represent documents (e.g., Shakespeare’s plays).
- Each cell (t,d) is 1 if the term t appears in the document d, otherwise 0.



	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	
…							

► **Figure 1.1** 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.

To answer the query Brutus AND Caesar AND NOT Calpurnia,

- we take the vectors for Brutus, Caesar and Calpurnia, complement the last, and then do a bitwise AND

  110100 AND 110111 AND 101111 = 100100

Assignments

1. Draw the inverted index that would be built for the following document collection.

Doc 1 new home sales top forecasts
Doc 2 home sales rise in july
Doc 3 increase in home sales in july
Doc 4 july new home sales rise

2. Consider these documents:

   Doc 1 breakthrough drug for schizophrenia
   Doc 2 new schizophrenia drug
   Doc 3 new approach for treatment of schizophrenia
   Doc 4 new hopes for schizophrenia patients

   a. Draw the term-document incidence matrix for this document collection.
   b. Draw the inverted index representation for this collection

Processing Boolean queries

How do we process a query using an inverted index and the basic Boolean retrieval model?

Consider processing the simple conjunctive query:

QUERIES

Brutus AND Calpurnia   // SIMPLE CONJUNCTIVE Queries

1. Locate Brutus in the Dictionary
2. Retrieve its postings
3. Locate Calpurnia in the Dictionary
4. Retrieve its postings
5. Intersect the two postings lists,

Limitations of Basic Boolean Retrieval

- Strict Boolean logic often leads to:
- High precision, low recall: Using AND retrieves very specific results but may miss relevant documents.
- Low precision, high recall: Using OR retrieves many irrelevant results alongside relevant ones.
- Binary matching lacks the ability to rank results by relevance.

Extended Boolean Model

To address these limitations, extended Boolean models incorporate features like:

Proximity Operators: Allow users to specify the distance between terms.

Example:

/s for the same sentence.

/p for the same paragraph.

/k for terms within k words of each other.

Wildcards and Phrases:

Exclamation marks (!) represent wildcards, e.g., **disclos! matches disclosure, disclosing,** etc.

Quotation marks specify exact phrases, e.g., "trade secret".

Example: Westlaw Queries

Westlaw, a legal information retrieval system, illustrates the extended Boolean model:

Query 1: "trade secret" /s disclos! /s prevent /s employe!

- Searches for documents where these terms appear in the same sentence.

Query 2: disab! /p access! /s work-site work-place

- Finds documents discussing disability and workplace accessibility in the same paragraph.

Query 3: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

- Matches documents about a host’s liability for intoxicated guests in the same paragraph.

Such queries are precise and often built incrementally to refine results.

…

Strengths of the Boolean Model

Control and Transparency: Users know exactly why documents are retrieved.

Domain-Specific Ranking: For example, legal documents in Westlaw are sorted chronologically.

However, experiments (e.g., Turtle, 1994) suggest that ranked retrieval often outperforms Boolean queries,

- even in professional contexts like legal research.

Advanced retrieval capabilities to address Boolean model limitations.

Spelling Tolerance: Retrieval that accounts for typos or synonyms.

Proximity and Phrase Search: Enhanced indexing for compound phrases or nearby terms.

Term Frequency: Accounting for how often terms appear in a document to improve relevance.

Ranking: Scoring documents to provide ordered results rather than a binary set.

Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF is a numerical statistic widely used in information retrieval and text mining to evaluate the importance of a term in a document relative to a collection or corpus of documents.

- It is particularly effective in ranking documents in response to a query.

Key Characteristics of TF-IDF:

Term Frequency (TF): Measures how often a term appears in a document.

- The higher the frequency of a term in a document, the higher its weight for that document.

Inverse Document Frequency (IDF):

- Measures how important a term is across all documents.
- Common terms (like "the", "is", "in") that appear in many documents are given a lower weight, while rare terms are assigned a higher weight.

Ranking: The result of TF-IDF is a score for each term in a document, which allows for the ranking of documents based on their relevance to a query.

- Higher TF-IDF scores indicate that a term is more significant in a document in relation to the corpus.

1. Components of TF-IDF

TF-IDF is the product of two measures: **Term Frequency (TF)** and **Inverse Document Frequency (IDF)**.

a) Term Frequency (TF)

TF measures how often a term t appears in a document d, normalized by the total number of terms in that document to avoid bias toward longer documents.

$$\mathrm{TF}(t, d) = \frac{f_{t,d}}{\sum_k f_{k,d}}$$

Where:

- $f_{t,d}$: Number of times term t appears in document d.
- $\sum_k f_{k,d}$: Total number of terms in document d.

b) Inverse Document Frequency (IDF)

IDF evaluates how important a term is by considering how common or rare it is across all documents in the collection.

$$\mathrm{IDF}(t, D) = \log\left(\frac{N}{1 + n_t}\right)$$

Where:

- N: Total number of documents in the corpus.
- n_t: Number of documents containing the term t.
- Adding 1 to n_t avoids division by zero for terms not present in any document.

…

Calculating TF-IDF

The TF-IDF score for a term t in a document d is:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Notes:

High TF: Terms that occur frequently in a document are more important.

High IDF: Terms that appear in fewer documents are more distinctive.

For example:

Common words **like "and" or "the"** appear in most documents, resulting in **low IDF scores**.

Specific terms like **"Shakespeare" or "Boolean"** have higher **IDF scores** if they occur in only a few documents.

Example

Suppose we have 3 documents:

d1 : "Information retrieval is important."

d2 : "Information retrieval uses TF-IDF."

d3 : "TF-IDF ranks documents."

Vocabulary:

- {information, retrieval, is, important, uses, TF-IDF, ranks, documents}

Step 1: Term Frequencies (TF)

For d_1:

- $\text{TF}(\text{information}, d_1) = \frac{1}{4} = 0.25$
- $\text{TF}(\text{retrieval}, d_1) = \frac{1}{4} = 0.25$
- $\text{TF}(\text{is}, d_1) = \frac{1}{4} = 0.25$
- $\text{TF}(\text{important}, d_1) = \frac{1}{4} = 0.25$

Step 2: Inverse Document Frequency (IDF)

For "information":

$$\text{IDF}(\text{information}, D) = \log\left(\frac{3}{1+2}\right) = \log(1) = 0$$

$\text{IDF} = 0$ means "information" is common across all documents.

For "important":

$$\text{IDF}(\text{important}, D) = \log\left(\frac{3}{1+1}\right) = \log(1.5) = 0.176$$

Step 3: Compute TF-IDF

For d_1, the score for "important" is:

$$\text{TF-IDF}(\text{important}, d_1) = 0.25 \times 0.176 = 0.044$$

Applications

Document Ranking: Ranks documents based on query relevance in search engines.

Text Clustering: Groups similar documents.

Topic Extraction: Identifies key terms in text corpora.

…

6. Advantages

- Balances term frequency with rarity, reducing bias from common terms.
- Simple to compute and effective for many text-based tasks.

7. Limitations

- Assumes term independence; ignores semantic relationships.
- May struggle with polysemy (same word, different meanings) and synonymy (different words, same meaning).

Relation Between TF-IDF and Boolean Model:

While Boolean models retrieve documents that match a query in a binary fashion, TF-IDF scores documents based on their relevance, offering a finer-grained ranking.

- However, both methods rely on the concept of indexing terms in documents, and their results can overlap if the TF-IDF thresholding is set to mimic the Boolean approach.

Combination of Both Models:

- Many modern search engines and retrieval systems combine Boolean operators (AND, OR, NOT) with TF-IDF scoring to take advantage of both precision and ranking.
- For example, a query might first filter out documents using Boolean operators (AND, OR, NOT)
- then rank the results based on TF-IDF to ensure the documents returned are most relevant according to term importance.

Vector space model

In this model, documents and queries are assumed to be the part of t-dimensional vector space, where t is the number of index terms (words, stems, phrases, etc.).

For eg: in 3 document , there 10 unique terms, then the vector is represented by a 10 dimensional vector.

- a first approach to finding relevant documents with respect to a given query.
- The main score functions are based on: Term-Frequency (tf) and Inverse-Document-Frequency(idf).

Overview

Document-Term Matrix:

- To create the vector representation of a collection of documents, you first construct a Document-Term Matrix (DTM) or Term-Document Matrix (TDM).
- Rows in this matrix represent documents, and columns represent terms (words or phrases).
- Each cell contains a numerical value representing a term’s frequency or importance within a document.

Term Frequency-Inverse Document Frequency (TF-IDF):

- Once you have the DTM, you often apply a TF-IDF transformation to the raw term frequencies.
- TF-IDF is a measure that reflects the importance of a term within a document relative to its importance across all documents in the corpus.
- It helps in highlighting important terms while ignoring the common terms.

…

Vectorization:

- After TF-IDF or a similar transformation, each document is represented as a vector in the high-dimensional space.
- The length of these vectors is typically the same (equal to the number of unique terms in the corpus),
  - but the values in each dimension vary depending on the term’s importance within the document.

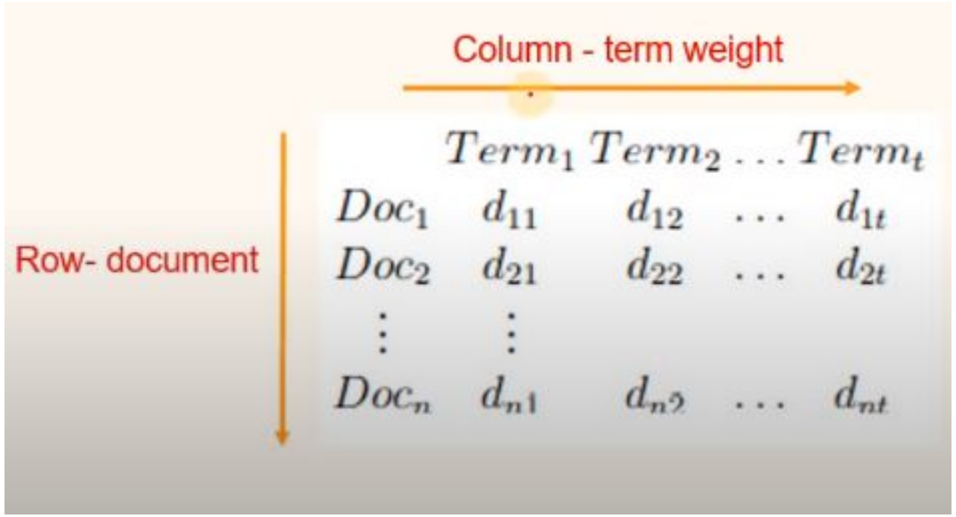
Cosine Similarity:

- To compare documents or perform text retrieval, you can use cosine similarity as a metric to measure the similarity between two document vectors.
- Cosine similarity calculates the cosine of the angle between two vectors and ranges from -1 (entirely dissimilar) to 1 (completely similar).
- A higher cosine similarity indicates a more significant similarity between documents.

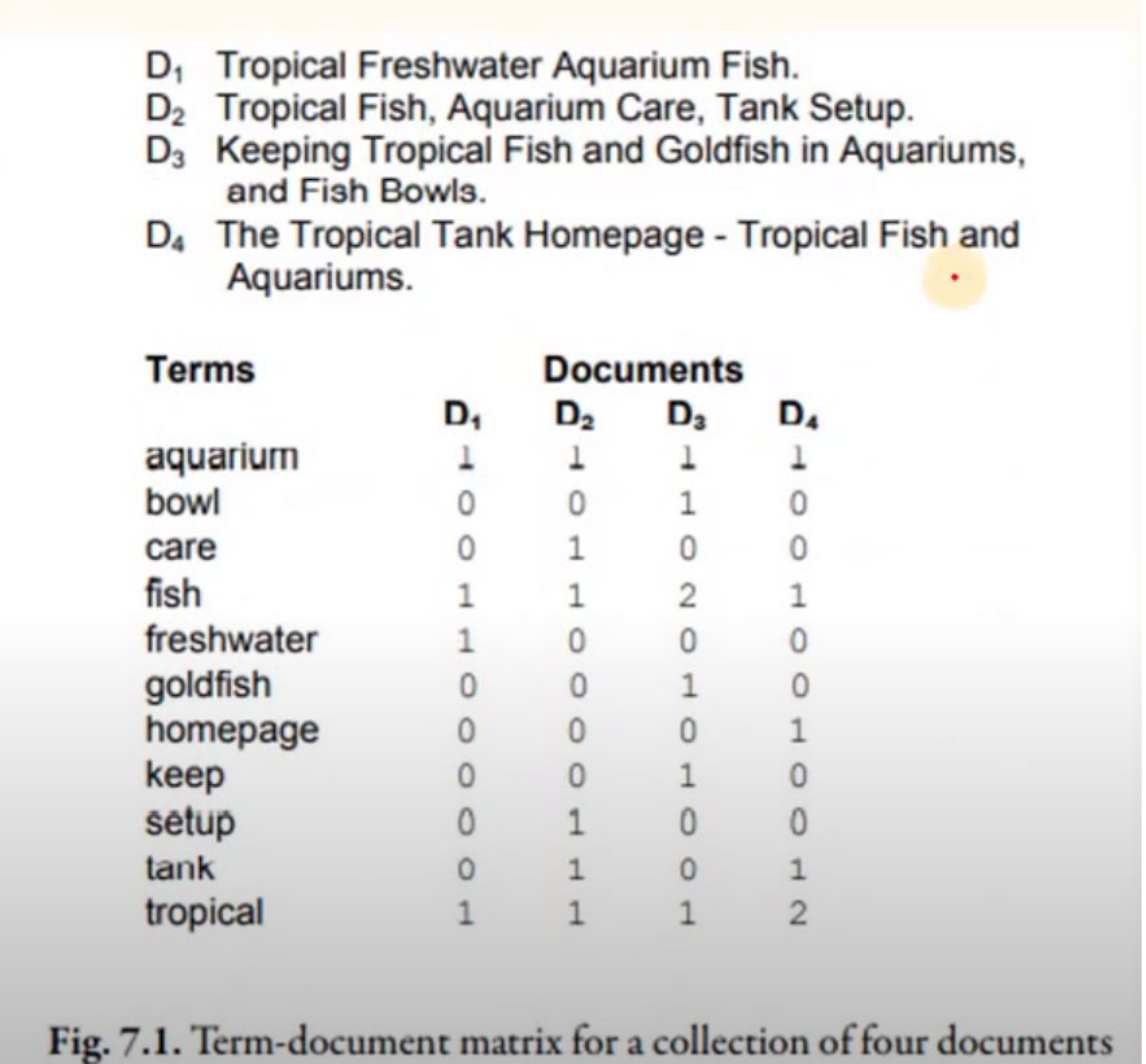
…

A document collection containing n documents can be represented as a matrix of term weights,

- The term weights are simply the count
  Of the terms in the document
- Stopwords are not indexed in this, example
- The words have been stemmed.



$$\begin{array}{ccccc} & Term_1 & Term_2 & \ldots & Term_t \\ Doc_1 & d_{11} & d_{12} & \ldots & d_{1t} \\ Doc_2 & d_{21} & d_{22} & \ldots & d_{2t} \\ \vdots & \vdots & & & \\ Doc_n & d_{n1} & d_{n2} & \ldots & d_{nt} \end{array}$$



D_1 Tropical Freshwater Aquarium Fish.
D_2 Tropical Fish, Aquarium Care, Tank Setup.
D_3 Keeping Tropical Fish and Goldfish in Aquariums, and Fish Bowls.
D_4 The Tropical Tank Homepage - Tropical Fish and Aquariums.

Terms	Documents			
	D_1	D_2	D_3	D_4
aquarium	1	1	1	1
bowl	0	0	1	0
care	0	1	0	0
fish	1	1	2	1
freshwater	1	0	0	0
goldfish	0	0	1	0
homepage	0	0	0	1
keep	0	0	1	0
setup	0	1	0	0
tank	0	1	0	1
tropical	1	1	1	2

Fig. 7.1. Term-document matrix for a collection of four documents

Document D3, for example, is represented by the

vector(1,1,0,2,0,1,0,1,0,0,1).

Query: “tropical fish”, vector representation of query be

vector(0,0,0,1,0,0,0,0,0,0,1).

For eg:

Given this representation, documents could be ranked by computing the distance between the points representing the documents and the query.

- A similarity measure is used (rather than distance or dissimilarity measure)

- The document with the highest score are the most similar to the query

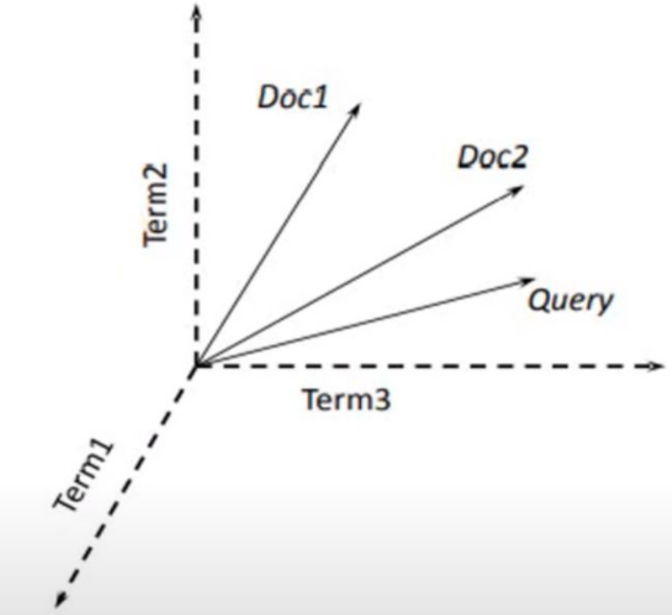


Fig. 7.2. Vector representation of documents and queries

Cosine similarity

The cosine correlation measures the cosine of the angles between the query and the document vectors.

- So, when the vectors are normalized so that all documents and queries are represented by vector of equal length, the cosine of the angle between two identical vectors will be 1 (the angle is zero), and for two vectors that do not share any non-zero terms, the cosine will be 0. The cosine measure is defined as,

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

Example:

consider two documents,

D1 = (0.5, 0.8, 0.3) and

D2 = (0.9, 0.4, 0.2) indexed by three terms.

Here, the numbers represent **term weights**.

Query Q = (1.5, 1.0, 0) indexed by the same terms, the cosine measure for the documents are:

…

We can see that, the document D2 is more relevant to the Query Q:

- As the sore is greater for Cosine(D2,Q), than Cosine(D1,Q)

$$
\begin{aligned}
Cosine(D_1, Q) &= \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}} \\
&= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87
\end{aligned}
$$

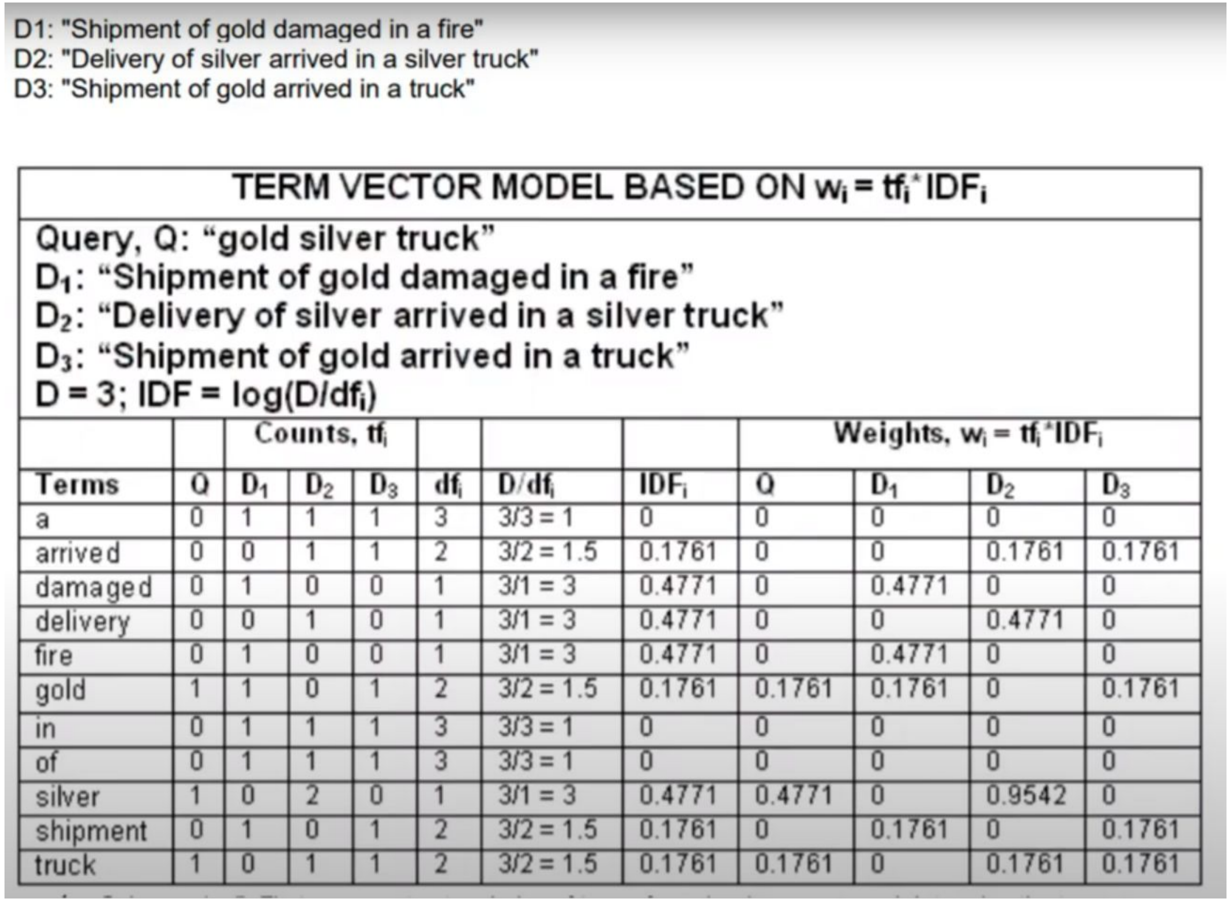
$$
\begin{aligned}
Cosine(D_2, Q) &= \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}} \\
&= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97
\end{aligned}
$$

...

The above formula computes the cosine of the angle described by the two normalized vectors : if the vectors are close, the angle is small and the relevance is high.

It can be shown the cosine similarity is the same of the Euclidean distance under the assumption of vector normalization.

Another Example



D1: "Shipment of gold damaged in a fire"
D2: "Delivery of silver arrived in a silver truck"
D3: "Shipment of gold arrived in a truck"

TERM VECTOR MODEL BASED ON $w_i = tf_i * IDF_i$

Query, Q: "gold silver truck"
D_1: "Shipment of gold damaged in a fire"
D_2: "Delivery of silver arrived in a silver truck"
D_3: "Shipment of gold arrived in a truck"
D = 3; IDF = log(D/df_i)

		Counts, tf_i						Weights, $w_i = tf_i * IDF_i$			
Terms	Q	D_1	D_2	D_3	df_i	D/df_i	IDF_i	Q	D_1	D_2	D_3
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

Example in python

```
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Sample documents

documents = [
    "The quick brown fox jumps over the lazy dog.",
    "A brown dog chased the fox.",
    "The dog is lazy."
]

# Sample query

query = "brown dog"

# Step 1: Tokenize and preprocess the text
nltk.download('punkt')
from nltk.tokenize import word_tokenize
tokenized_documents = [word_tokenize(doc.lower()) for
doc in documents]
tokenized_query = word_tokenize(query.lower())
```

```
# Step 2: Calculate TF-IDF
# Convert tokenized documents to text
preprocessed_documents = [' '.join(doc) for doc in
tokenized_documents]
preprocessed_query = ' '.join(tokenized_query)

# Create a TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix =
tfidf_vectorizer.fit_transform(preprocessed_documents)

# Transform the query into a TF-IDF vector

query_vector = tfidf_vectorizer.transform([preprocessed_query])

# Step 3: Calculate cosine similarity

cosine_similarities = cosine_similarity(query_vector, tfidf_matrix)

# Step 4: Rank documents by similarity

results = [(documents[i], cosine_similarities[0][i]) for i in
range(len(documents))]
results.sort(key=lambda x: x[1], reverse=True)

# Print the ranked documents

for doc, similarity in results:
    print(f"Similarity: {similarity:.2f}\n{doc}\n")
```

…

Cosine similarity has several advantages when applied to text data:

- Scale Invariance: Cosine similarity is scale-invariant, meaning it’s not affected by the magnitude of the vectors. This makes it suitable for documents of different lengths.

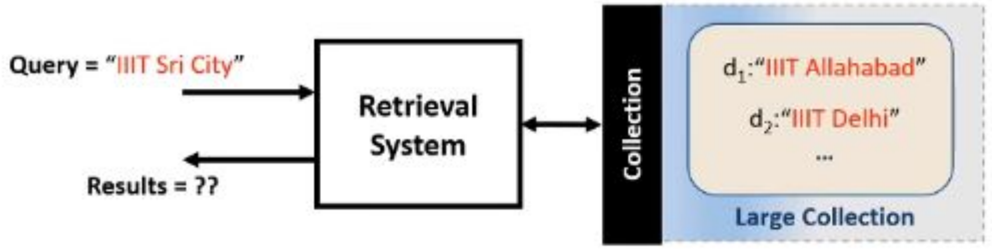
- Angle Measure: It focuses on the direction of vectors rather than their absolute values, which is crucial for text similarity, where document length can vary.

- Efficiency: Calculating cosine similarity is computationally efficient, making it suitable for large-scale text datasets.

Simple Retrieval Problem

- A **collection** with 5 **documents** having the following contents
  - d1: IIIT ALLAHABAD
  - d2: IIIT DELHI
  - d3: IIIT GUWAHATI
  - d4: IIIT KANCHIPURAM
  - d5: IIIT SRI CITY
- **Query** is
  - IIIT SRI CITY
- Which **document** will you match and why?

The Problem – How to Rank?

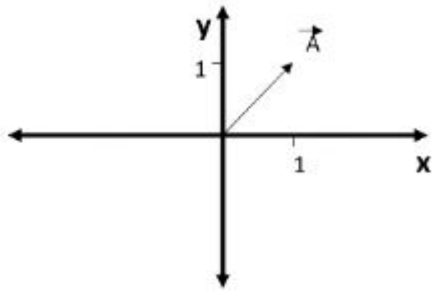


Compare each document with the query and assign a score of relevance?

This is slow and is there any other way to rank?

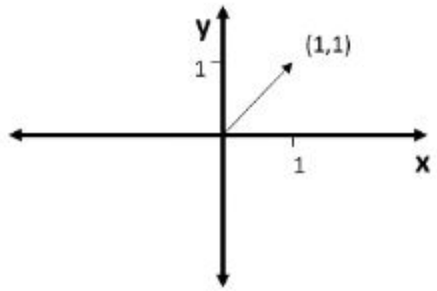
Vectors

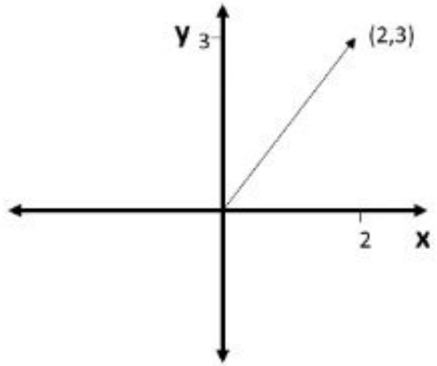
- Geometric entity which has magnitude and direction



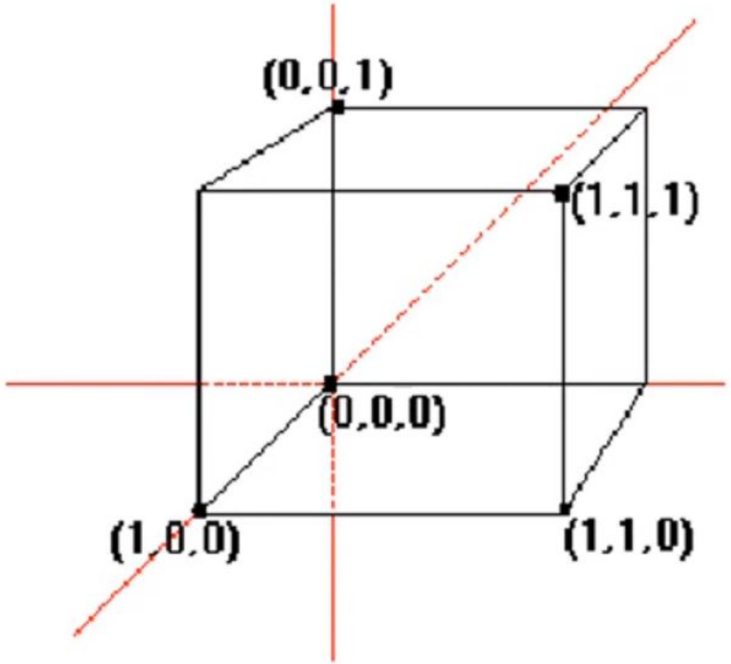
- If (x,y) is our vector of interest, this figure shows $\vec{A}$ vector = (1,1).

How is (2,3) Different?



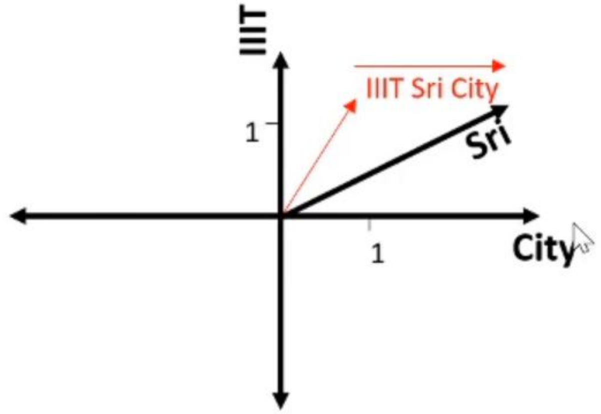


What is (1,1,1) ?



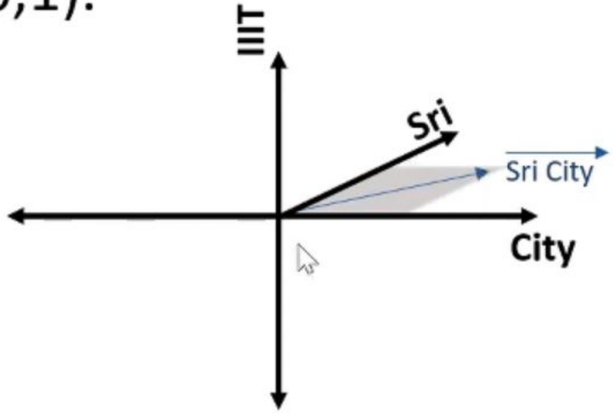
Sentences are Vectors

- "IIIT Sri City" is a 3-dimensional vector



Sentences are Vectors

- On this 3D space, “Sri City” vector will lie on the x (City) and z (Sri) plane. If (x,y,z) denotes the vector, “Sri City” is (1,0,1).



More Linear Algebra…

- So, we learned to represent English phrases on the vector space.
- We need something more!

Revisiting Matrices

Natural Language Phrases as Vectors

Let query q = "IIIT Sri City".

Let document, d_1 = "IIIT Sri City" and d_2 = "IIIT Delhi".

	IIIT	Sri	City	Delhi
q	1	1	1	0
d_1	1	1	1	0
d_2	1	0	0	1

$q = (1,1,1,0)$, $d_1 = (1,1,1,0)$ and $d_2 = (1,0,0,1)$

Quiz

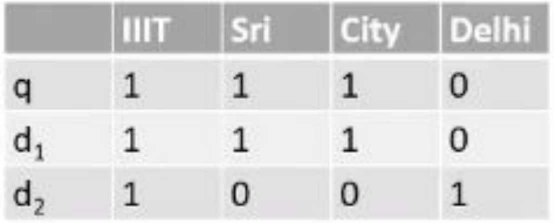
- Considering the following vectors:

	IIIT	Sri	City	Delhi
q	1	1	1	0
d_1	1	1	1	0
d_2	1	0	0	1

- What is the Natural Language (NL) equivalent of (0,1,1,0) ?
- What is the NL equivalent of (1,0,0,1) ?
- What is the vector for Delhi?
- What is the NL equivalent of q?

Quiz

- Considering the following vectors:



	IIIT	Sri	City	Delhi
q	1	1	1	0
d_1	1	1	1	0
d_2	1	0	0	1

- What is the Natural Language (NL) equivalent of (0,1,1,0) ? **Sri City**
- What is the NL equivalent of (1,0,0,1) ? **IIIT Delhi**
- What is the vector for Delhi? **0001**
- What is the NL equivalent of q? **IIIT Sri City**

Which of the Following are Sets?

- {1, 2, 3, 4, 5, 6, 5, 7, 8, 9, 10, 11, 12, 13}
- {A, B, C, D, E, F, G, H, I, I, J, K, L, M, N, O}
- {apple, banana, orange, apple, banana, orange}

Bag

- {1, 2, 3, 4, 5, 6, 5, 7, 8, 9, 10, 11, 12, 13}
- {A, B, C, D, E, F, G, H, I, I, J, K, L, M, N, O}
- {apple, banana, orange, apple, banana, orange}

Set of Words Representation

- “IIIT Sri City” → {IIIT, Sri, City}
- “IIIT Sri City, Sri City” → {IIIT, Sri, City}

	IIIT	Sri	City
q	1	1	1

Leads to same term-document matrix

Set Similarity

- Similarity between two sets is easy

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Jaccard Similarity

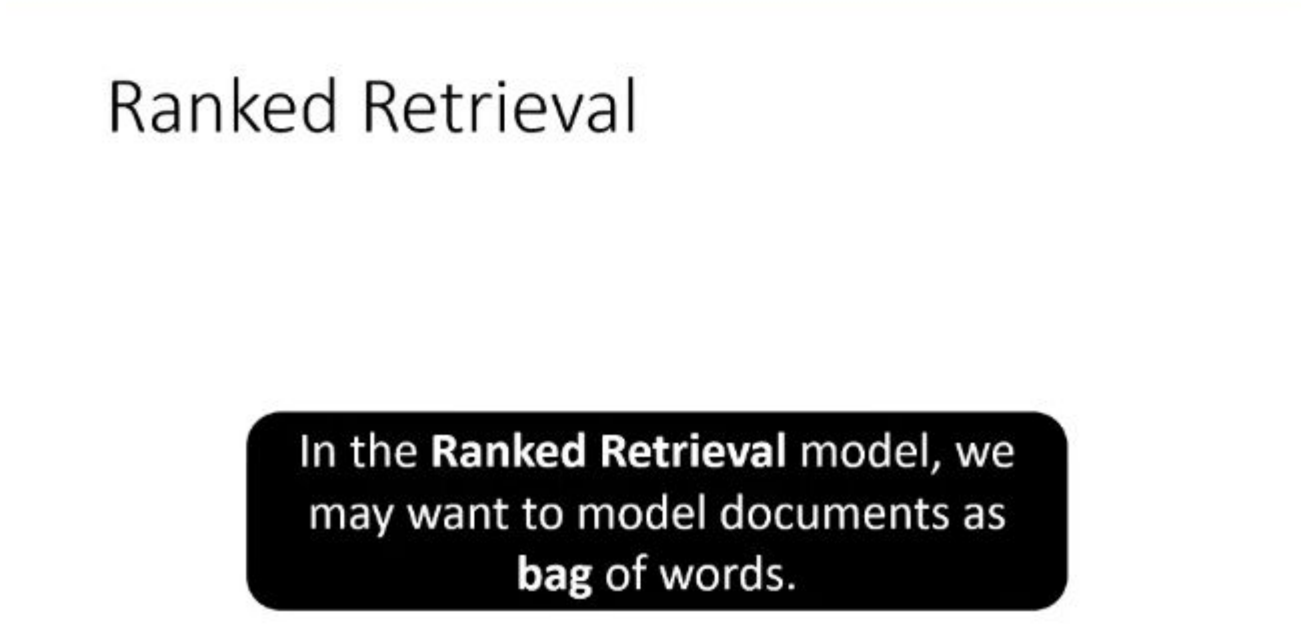
Quiz

- What is the Jaccard similarity between
  - {1,2,3} and {4,5,6} = 0
  - {1,2,3} and {1,2,4} = $\frac{|\{1,2\}|}{|\{1,2,3,4\}|}$ = 0.5
  - {1,2,3} and {1,2,3} = 1

Quiz

- What is the Jaccard similarity between
  - “IIIT is Great” and “IIITD is Great”?
- Same as Jaccard Similarity between {IIIT, is, Great} and {IIITD, is, Great}. Equals 0.5.

This sets and idea for ranked retrieval model



Ranked Retrieval

In the **Ranked Retrieval** model, we may want to model documents as **bag** of words.

Bag of Words Representation

- "IIIT Sri City" → {IIIT, Sri, City}
- "IIIT Sri City, Sri City" → [IIIT, Sri, Sri, City, City]

IIIT Sri City

	IIIT	Sri	City
q	1	1	1

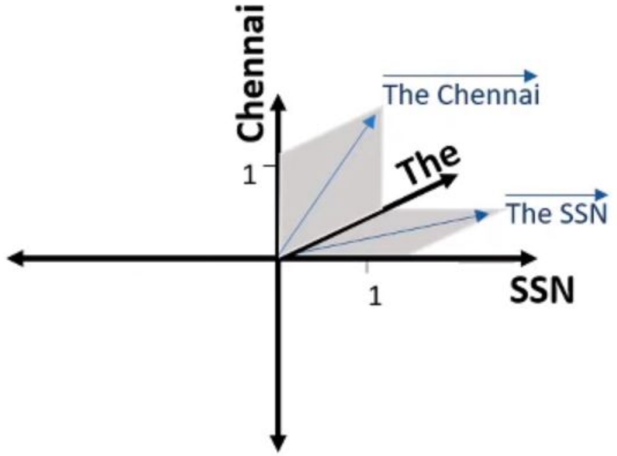
IIIT Sri City, Sri City

	IIIT	Sri	City
q	1	2	2

Leads to different term-document matrix

Comparing Sentences

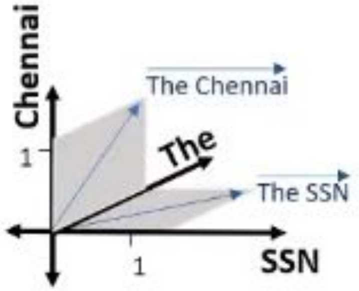
- We can compare sentences using the angle between vectors


Chennai
$\overrightarrow{\text{The Chennai}}$
The
$\overrightarrow{\text{The SSN}}$
1
1
SSN

All are 90 degree apart.

Mathematical Notation

- We represent vectors as follows:
  - Vector = (dimension1, dimension2, dimension3, …)
    - First, define the dimensions
    - Next, put “1” if the word is present in the sentence, else “0”
- Example



Chennai
$\overrightarrow{\text{The Chennai}}$
The
$\overrightarrow{\text{The SSN}}$
1
1
SSN

Vector = (dimension1, d2, d3, …)

In our case,
vector = (The, SSN, Chennai)

So, $\overrightarrow{\text{The Chennai}} = (1,0,1)$

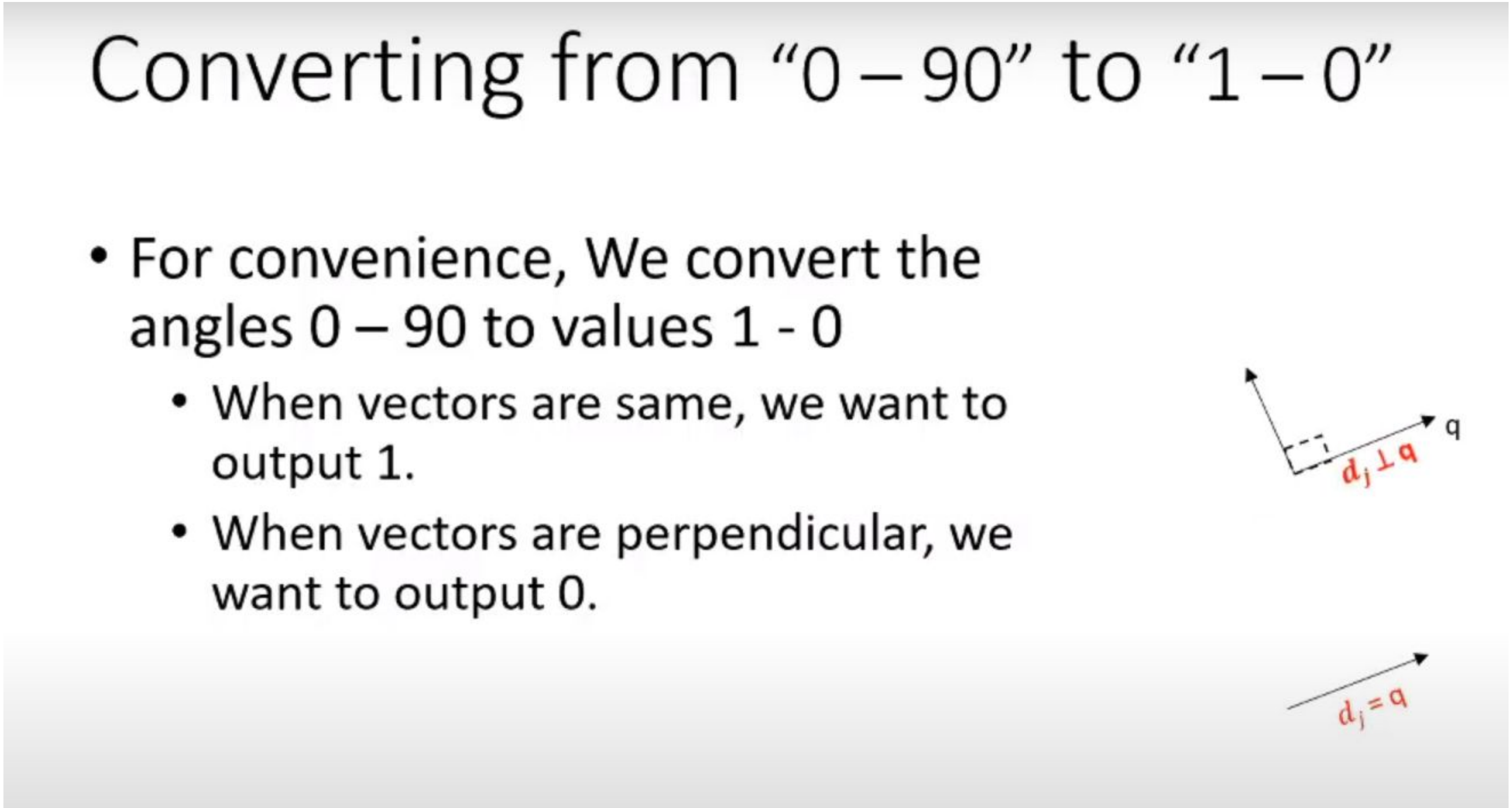
$\overrightarrow{\text{The SSN}} = (1,1,0)$

Similarity Score

- D1 = “Chennai”
- D2 = “Chennai”

- Quiz
  - On a scale of 0 – 1, how similar are D1 and D2?
    - 0 → Dissimilar
    - 1 → Identical

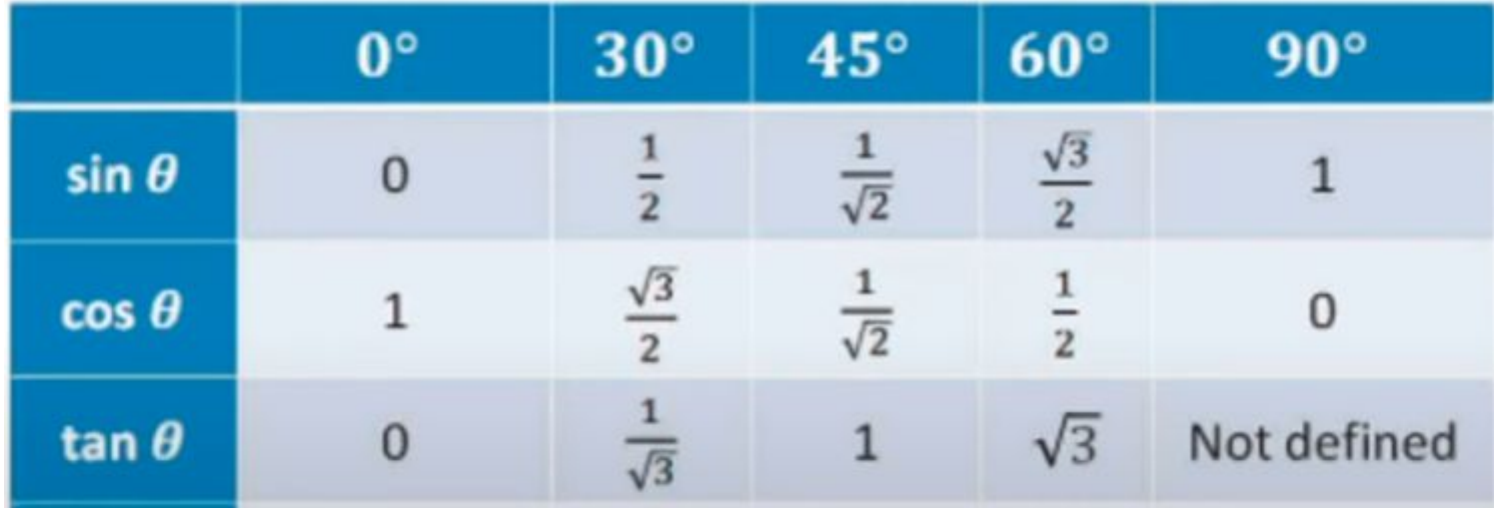
How to convert 0 to 90 -> 1 to 0



Converting from "0 – 90" to "1 – 0"

- For convenience, We convert the angles 0 – 90 to values 1 - 0
  - When vectors are same, we want to output 1.
  - When vectors are perpendicular, we want to output 0.

0-90 to 1-0: How?



	$0°$	$30°$	$45°$	$60°$	$90°$
$\sin\theta$	0	$\frac{1}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{\sqrt{3}}{2}$	1
$\cos\theta$	1	$\frac{\sqrt{3}}{2}$	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	0
$\tan\theta$	0	$\frac{1}{\sqrt{3}}$	1	$\sqrt{3}$	Not defined

Back to Trigonometry

- If x and y are non-unit vectors, what is the cosine of angle between them (cos Θ)?

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \; \|\mathbf{b}\| \cos(\theta)$$

Matching Documents to Queries

- Document as a vector of term-weights

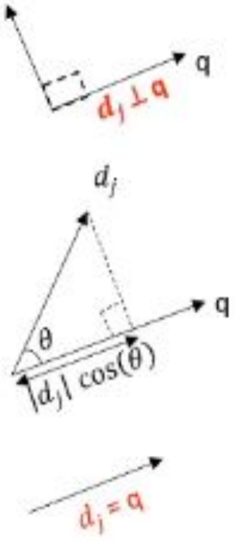
$$d_j = (w_{1j}, w_{2j}, \dots w_{nj})$$

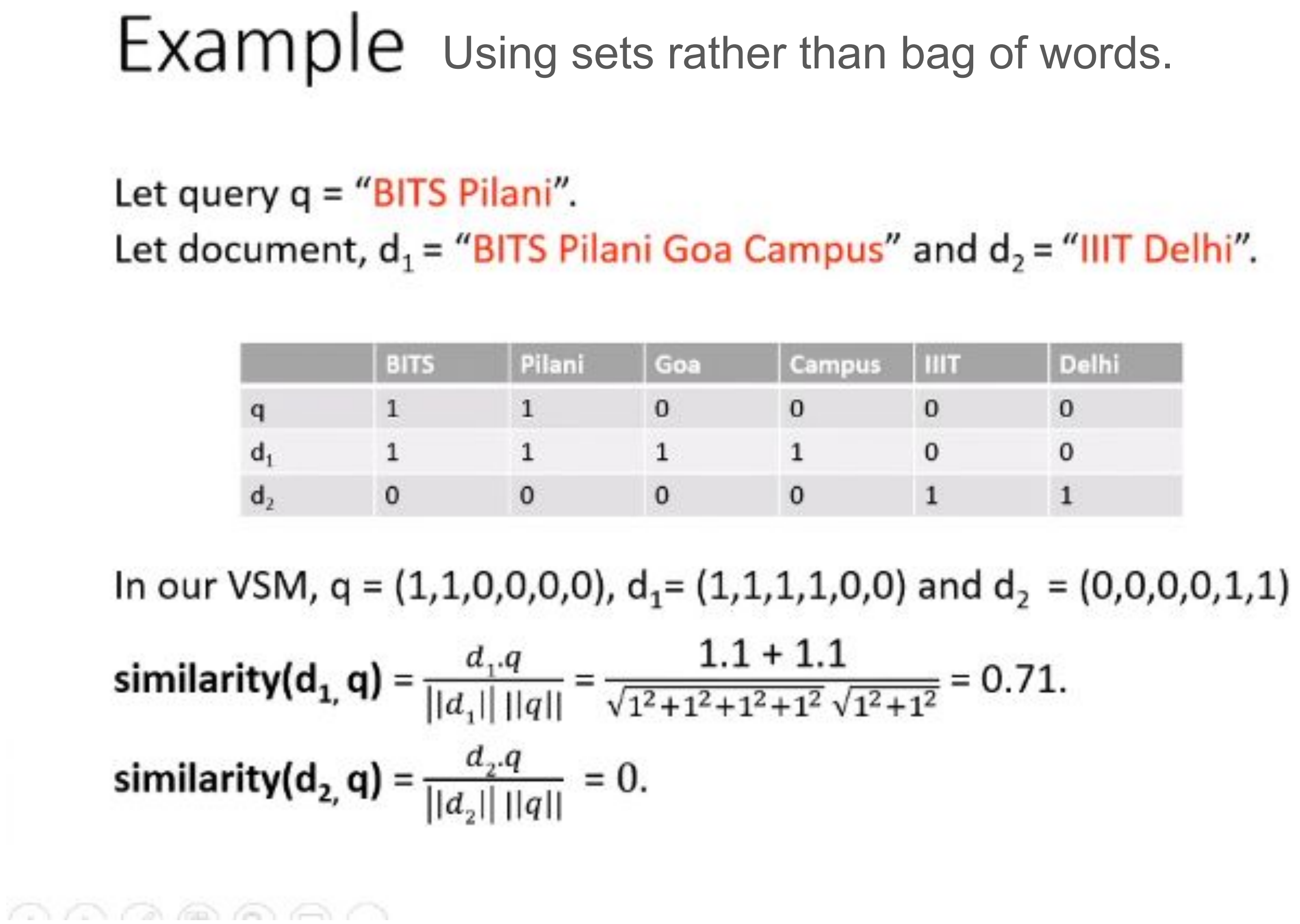
- Query as a vector of term-weights

$$q = (w_1, w_2, \dots wm)$$

- Similarity between these vectors can be represented as

$$Cosine\ Similarity = \cos(\theta) = \frac{d_j . q}{||d_j||\ ||q||}$$





Example

Using sets rather than bag of words.

Let query q = “BITS Pilani”.

Let document, d_1 = “BITS Pilani Goa Campus” and d_2 = “IIIT Delhi”.

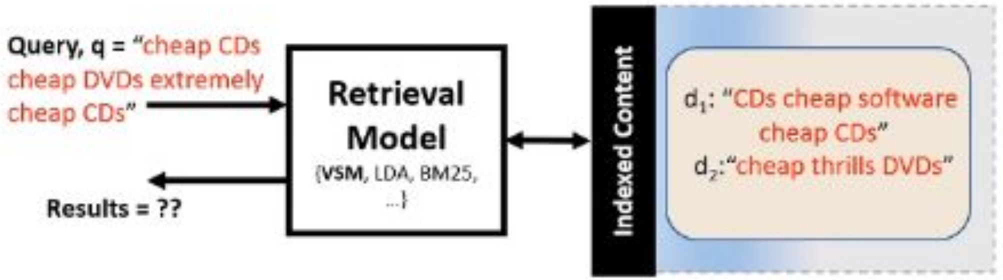
	BITS	Pilani	Goa	Campus	IIIT	Delhi
q	1	1	0	0	0	0
d_1	1	1	1	1	0	0
d_2	0	0	0	0	1	1

In our VSM, q = (1,1,0,0,0,0), d_1= (1,1,1,1,0,0) and d_2 = (0,0,0,0,1,1)

$$\textbf{similarity}(\mathbf{d_1, q}) = \frac{d_1.q}{||d_1||\ ||q||} = \frac{1.1 + 1.1}{\sqrt{1^2+1^2+1^2+1^2}\ \sqrt{1^2+1^2}} = 0.71.$$

$$\textbf{similarity}(\mathbf{d_2, q}) = \frac{d_2.q}{||d_2||\ ||q||} = 0.$$

Using the Bag of Words Model

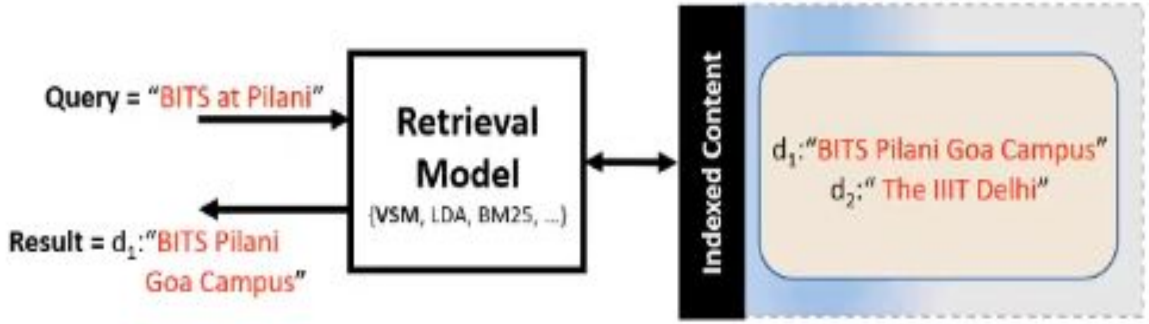

Query, q = "cheap CDs cheap DVDs extremely cheap CDs"
Retrieval Model
{VSM, LDA, BM25, ...}
Results = ??
Indexed Content
d1: "CDs cheap software cheap CDs"
d2:"cheap thrills DVDs"

	cheap	CDs	DVDs	extremely	software	thrills
q	3	2	1	1	0	0
d_1	2	2	0	0	1	0
d_2	1	0	1	0	0	1

$\text{sim}(q,d_1) = 0.86$

$\text{sim}(q,d_2) = 0.59$

Not Every Word is Important!



Let us add Term Weights

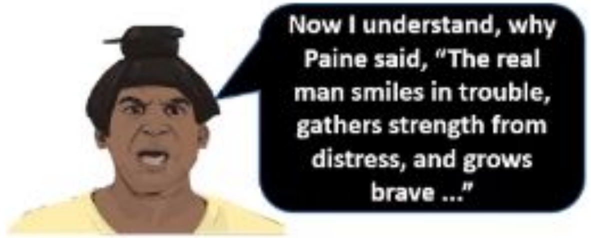
	BITS	the (* 0)	Pilani	Goa	Campus	IIIT	Delhi
q	1	1 * 0 = 0	1	0	0	0	0
d_1	1	0 * 0 = 0	1	1	1	0	0
d_2	0	1 * 0 = 0	0	0	0	1	1

← $sim(q,d_1) = 0.71$

← $sim(q,d_2) = 0$

Term Weighting with Inverse Document Frequency

What should be the term weight for each of the terms in this document?


Now I understand, why Paine said, "The real man smiles in trouble, gathers strength from distress, and grows brave ..."

Indexed Content

d$_1$: *"An inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely."*

d$_2$: *"A high weight in tf–idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms"*

Inverse Document Frequency

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

where N = |D| = Total no. of documents.

$idf(\text{"the"}, \{\mathbf{d_1}, \mathbf{d_2}\}) = \log\frac{2}{2} = 0$

$idf(\text{"batsmen"}, \{\mathbf{d_1}, \mathbf{d_2}\}) = \log\frac{2}{1} = 0.3$

But, do you see any problem? Clue… divide by zero.

Indexed Content

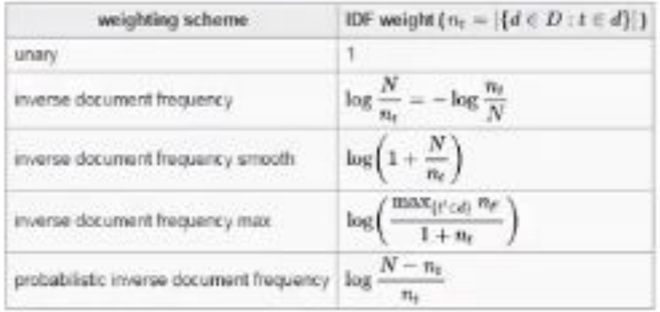
d$_1$: *"An inverse document frequency factor is incorporated which diminishes* ***the*** *weight of terms that occur very frequently in* ***the*** *document set and increases* ***the*** *weight of terms that occur rarely."*

d$_2$: *"Sachin Ramesh Tendulkar is a former Indian international cricketer and a former captain of* ***the*** *Indian national team, regarded as one of* ***the*** *greatest batsmen of all time."*

Variants

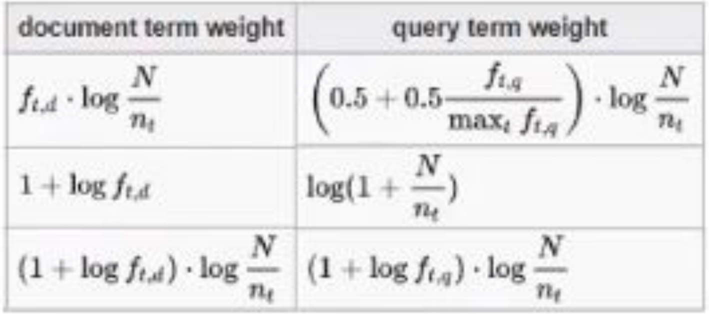
Wikipedia lists the following IDF and TF-IDF variants.

IDF Variants*

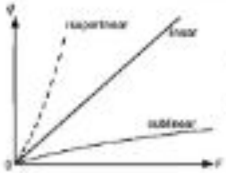


weighting scheme	IDF weight ($n_t = \|\{d \in D : t \in d\}\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

TF-IDF Variants*



document term weight	query term weight
$f_{t,d} \cdot \log \frac{N}{n_t}$	$\left(0.5 + 0.5 \frac{f_{t,q}}{\max_t f_{t,q}}\right) \cdot \log \frac{N}{n_t}$
$1 + \log f_{t,d}$	$\log(1 + \frac{N}{n_t})$
$(1 + \log f_{t,d}) \cdot \log \frac{N}{n_t}$	$(1 + \log f_{t,q}) \cdot \log \frac{N}{n_t}$



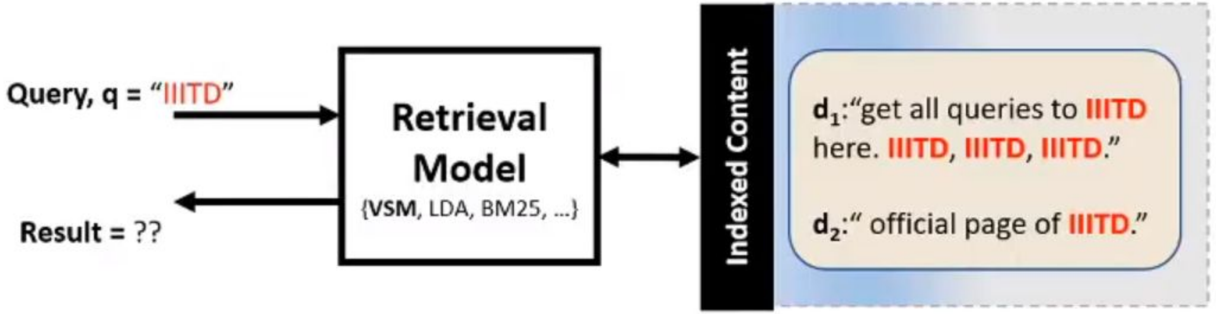
Sub-linear Damping

Add-One Smoothing

Normalization

*See Wikipedia page on TF-IDF (https://en.wikipedia.org/wiki/Tf%E2%80%93idf) for more information.

We are Biased Towards Long Documents


Query, q = "IIITD"
Result = ??
Retrieval Model
{VSM, LDA, BM25, ...}
Indexed Content
d1:"get all queries to IIITD here. IIITD, IIITD, IIITD."
d2:" official page of IIITD."

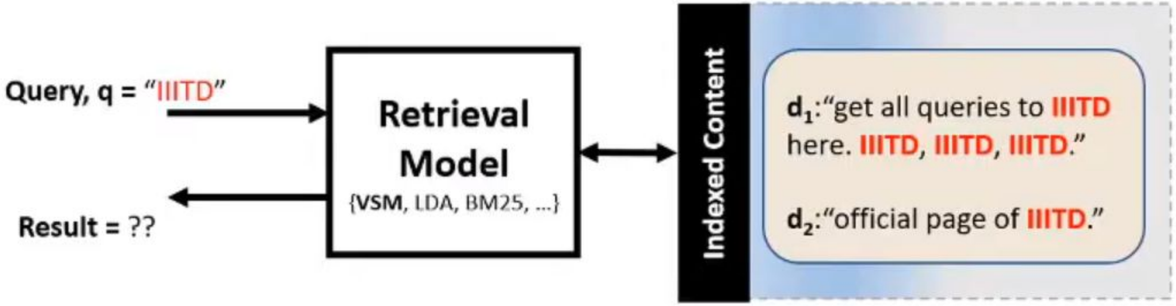
	IIITD	get	all	queries	to	here	official	page	of
q	1	0	0	0	0	0	0	0	0
d_1	4	1	1	1	1	1	0	0	0
d_2	1	0	0	0	0	0	1	1	1

$$\mathrm{sim}(q,d_1) = \frac{d_1.q}{||d_1||\,||q||} = \frac{4.1}{\sqrt{4^2+1^2+1^2+1^2+1^2+1^2}\,\sqrt{1^2}} = 0.87 \quad \mathrm{sim}(q,d_2) = \frac{d_2.q}{||d_2||\,||q||} = \frac{1.1}{\sqrt{1^2+1^2+1^2+1^2}\,\sqrt{1^2}} = 0.5$$

Length Normalization

$$tf_L(t,dj) = tf(t,dj) \, * \log(1 + \frac{Avg.DL}{len(dj)})$$

$$Avg.DL = \frac{9+4}{2} = 6.5$$



d$_1$:"get all queries to **IIITD** here. **IIITD**, **IIITD**, **IIITD**."

d$_2$:"official page of **IIITD**."

$$tf_L(\text{"IIITD"}, d_1) = 4 \, * \log(1 + \frac{6.5}{9}) = 0.94$$

$$tf_L(\text{"IIITD"}, d_2) = 1 \, * \log(1 + \frac{6.5}{4}) = 0.42$$

Challenges and Limitations of a Vector Space Model

1. The Curse of Dimensionality

VSM represents text as high-dimensional vectors, the dimensions can become extremely large as the vocabulary and document corpus grow. This results in several issues:

- Increased Computational Complexity:
    - Operating in high-dimensional spaces requires substantial computational resources for representation and similarity calculations.
- Data Sparsity:
    - In high-dimensional spaces, data can become sparse, meaning many dimensions have zero values. This sparsity can affect the efficiency and accuracy of algorithms.
- Overfitting:
    - In machine learning tasks, high dimensionality can lead to overfitting, where models become too specific to the training data and fail to generalize well to new data.

…

2. Semantic Understanding

VSM, while effective in capturing term frequency and importance, doesn’t inherently capture the semantic meaning of words or phrases.

It treats words as independent entities and doesn’t recognize the relationships between them. This limitation can lead to issues such as:

- Synonymy and Polysemy: VSM may struggle to distinguish between synonyms (words with similar meanings) and polysemous words (words with multiple meanings) because they rely on individual term frequencies.

- Lack of Context: It doesn’t consider the context in which words are used. Two sentences with similar words but different meanings may have similar vector representations.

Advances in NLP

Word Embeddings:

- Word embeddings such as Word2Vec, GloVe, and FastText capture semantic relationships between words by learning dense, low-dimensional vector representations.
- They provide more nuanced representations of word meanings.

Transformer Models:

- Models like BERT, GPT, and their variants have revolutionized NLP by learning contextual embeddings for words and sentences.
- They excel in question answering, text generation, and sentiment analysis.

Thank you for your time and attention 😊