IS418
Lecture 4

Ranking **models**

Dr.Ebtsam AbdelHakam

Indexes

Storing document information for faster queries

Example “Collection”

S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.

S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Four sentences from the Wikipedia entry for *tropical fish*

Simple Inverted Index

Term	Postings	Term	Postings
and	1	only	2
aquarium	3	pigmented	4
are	3 4	popular	3
around	1	refer	2
as	2	referred	2
both	1	requiring	2
bright	3	salt	1 4
coloration	3 4	saltwater	2
derives	4	species	1
due	3	term	2
environments	1	the	1 2
fish	1 2 3 4	their	3
fishkeepers	2	this	4
found	1	those	2
fresh	2	to	2 3
freshwater	1 4	tropical	1 2 3
from	4	typically	4
generally	4	use	2
in	1 4	water	1 2 4
include	1	while	4
including	1	with	2
iridescence	4	world	1
marine	2		
often	2 3		

Inverted Index with counts

- supports better ranking algorithms

and	1:1			
aquarium	3:1			
are	3:1	4:1		
around	1:1			
as	2:1			
both	1:1			
bright	3:1			
coloration	3:1	4:1		
derives	4:1			
due	3:1			
environments	1:1			
fish	1:2	2:3	3:2	4:2
fishkeepers	2:1			
found	1:1			
fresh	2:1			
freshwater	1:1	4:1		
from	4:1			
generally	4:1			
in	1:1	4:1		
include	1:1			
including	1:1			
iridescence	4:1			
marine	2:1			
often	2:1	3:1		

only	2:1		
pigmented	4:1		
popular	3:1		
refer	2:1		
referred	2:1		
requiring	2:1		
salt	1:1	4:1	
saltwater	2:1		
species	1:1		
term	2:1		
the	1:1	2:1	
their	3:1		
this	4:1		
those	2:1		
to	2:2	3:1	
tropical	1:2	2:2	3:1
typically	4:1		
use	2:1		
water	1:1	2:1	4:1
while	4:1		
with	2:1		
world	1:1		

Inverted Index with positions

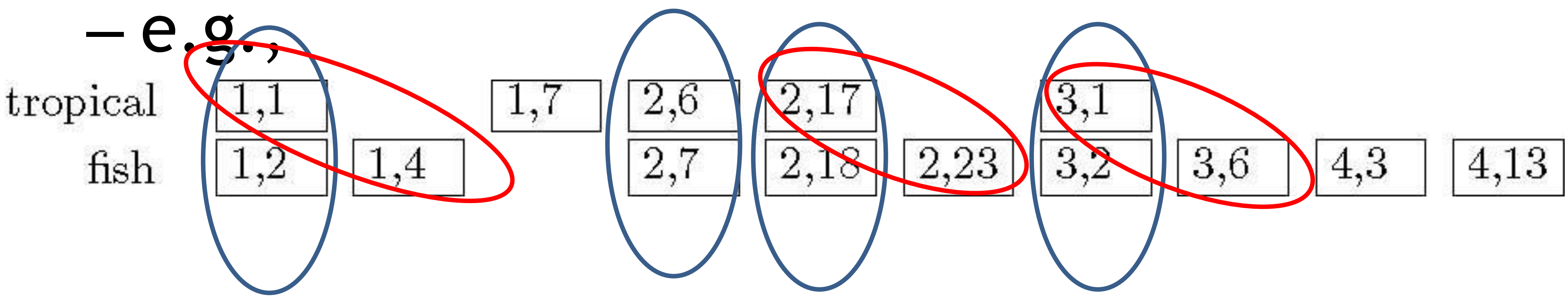
- supports proximity matches

Term	Postings
and	1,15
aquarium	3,5
are	3,3 · 4,14
around	1,9
as	2,21
both	1,13
bright	3,11
coloration	3,12 · 4,5
derives	4,7
due	3,7
environments	1,8
fish	1,2 · 1,4 · 2,7 · 2,18 · 2,23 · 3,2 · 3,6 · 4,3 · 4,13
fishkeepers	2,1
found	1,5
fresh	2,13
freshwater	1,14 · 4,2
from	4,8
generally	4,15
in	1,6 · 4,1
include	1,3
including	1,12
iridescence	4,9

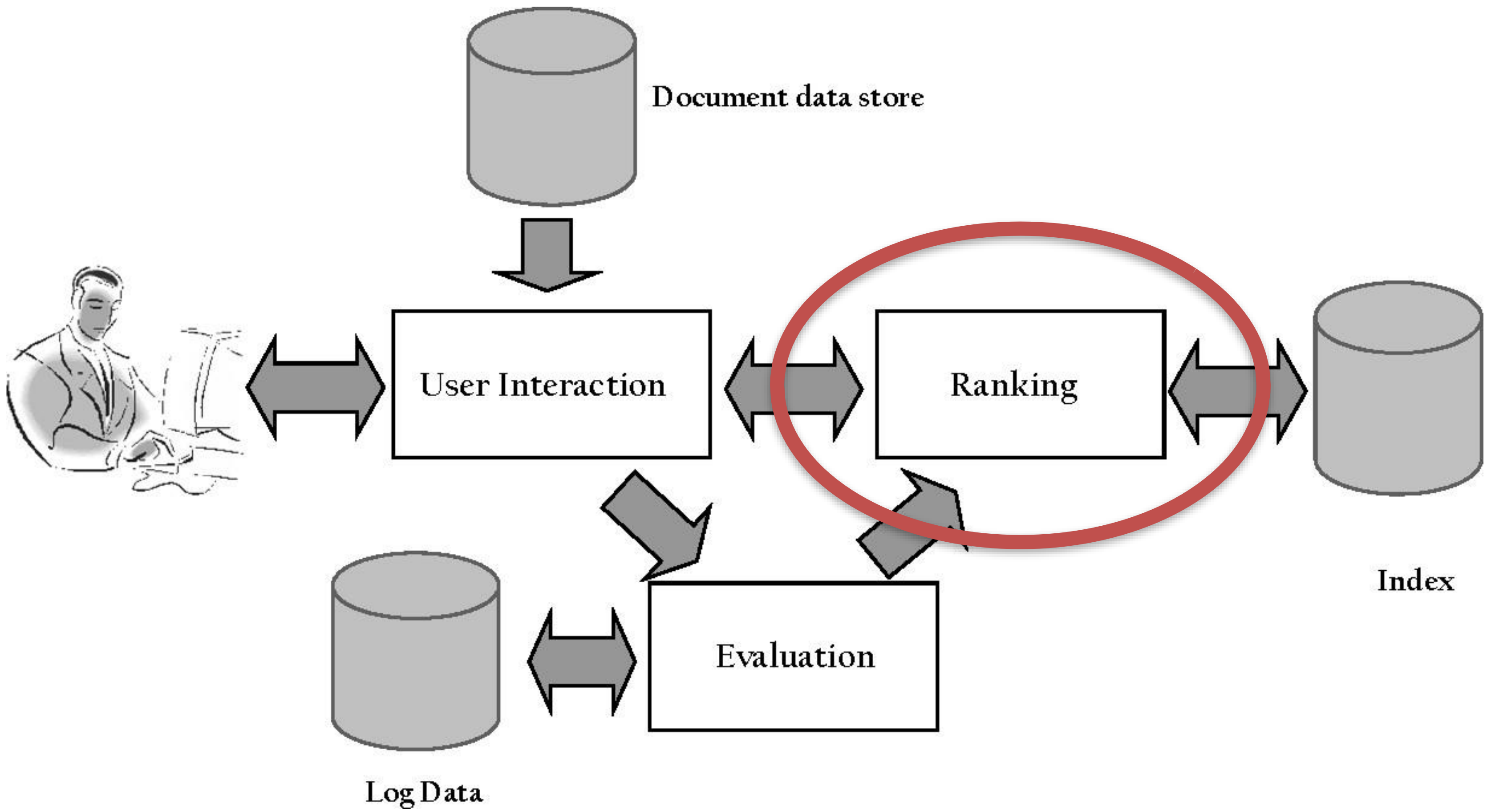
Term	Postings
marine	2,22
often	2,2 · 3,10
only	2,10
pigmented	4,16
popular	3,4
refer	2,9
referred	2,19
requiring	2,12
salt	1,16 · 4,11
saltwater	2,16
species	1,18
term	2,5
the	1,10 · 2,4
their	3,9
this	4,4
those	2,11
to	2,8 · 2,20 · 3,8
tropical	1,1 · 1,7 · 2,6 · 2,17 · 3,1
typically	4,6
use	2,3
water	1,17 · 2,14 · 4,12
while	4,10
with	2,15
world	1,11

Proximity Matches

- Matching phrases or words within a window
  - e.g., "`tropical fish`", or "find tropical within 5 words of fish"
- Word positions in inverted lists make these types of query features efficient
  - e.g.,



Query Process



Retrieval Model Overview

- Simple models
  - Boolean retrieval
  - Vector Space model
- Probabilistic Models
  - BM25
  - Language models
- Combining evidence
  - Inference networks
  - Learning to Rank

Boolean Retrieval model

- The **Boolean Model** is one of the simplest and earliest retrieval models.
- It relies on Boolean logic (AND, OR, NOT) to determine whether a document is relevant or not.

Key Features:

1. Uses **exact matching** (documents either match or don't).
2. Queries are expressed as Boolean expressions.
3. Simple and efficient for **small** datasets.
4. Does not **rank** documents by relevance.

Boolean Retrieval Model

- **Queries**: Users express queries as a ***Boolean expression***
  - AND, OR, NOT
  - Can be arbitrarily nested
- Ex. query: ***Qatar*** *AND* ***University*** *AND NOT* ***Street***

- **Documents**: Views each document as a ***"bag" of words***

- Return only documents that satisfy the Boolean query.

Exercise

Build a **Term-Document Incidence Matrix**

- Which term appears in which document
- Rows are terms
- Columns are documents

Given example collection:

d_1: He likes to wink, he likes to drink

d_2: He likes to drink, and drink, and drink

d_3: The thing he likes to drink is ink

d_4: The ink he likes to drink is pink

d_5: He likes to wink, and drink pink ink

	d_1	d_2	d_3	d_4	d_5
he	1	1	1	1	1
likes	1	1	1	1	1
to	1	1	1	1	1
wink	1	0	0	0	1
drink	1	1	1	1	1
and	0	1	0	0	1
the	0	0	1	1	0
thing	0	0	1	0	0
ink	0	0	1	1	1
is	0	0	1	1	0
pink	0	0	0	1	1

Term-Document Incidence Matrix

documents

terms	d_1	d_2	d_3	d_4	d_5
he	1	1	1	1	1
likes	1	1	1	1	1
to	1	1	1	1	1
wink	1	0	0	0	1
drink	1	1	1	1	1
and	0	1	0	0	1
the	0	0	1	1	0
thing	0	0	1	0	0
ink	0	0	1	1	1
is	0	0	1	1	0
pink	0	0	0	1	1

1 if ***document*** contains ***term***, 0 otherwise

Term-Document Incidence Matrix

	d_1	d_2	d_3	d_4	d_5
he	1	1	1	1	1
likes	1	1	1	1	1
to	1	1	1	1	1
wink	1	0	0	0	1
drink	1	1	1	1	1
and	0	1	0	0	1
the	0	0	1	1	0
thing	0	0	1	0	0
ink	0	0	1	1	1
is	0	0	1	1	0
pink	0	0	0	1	1

Query: ***wink*** *AND* ***drink*** *AND NOT* ***ink***

Apply on rows: **10001** *AND* **11111** *AND* !(**00111**) = **10000**

Bigger Collections …

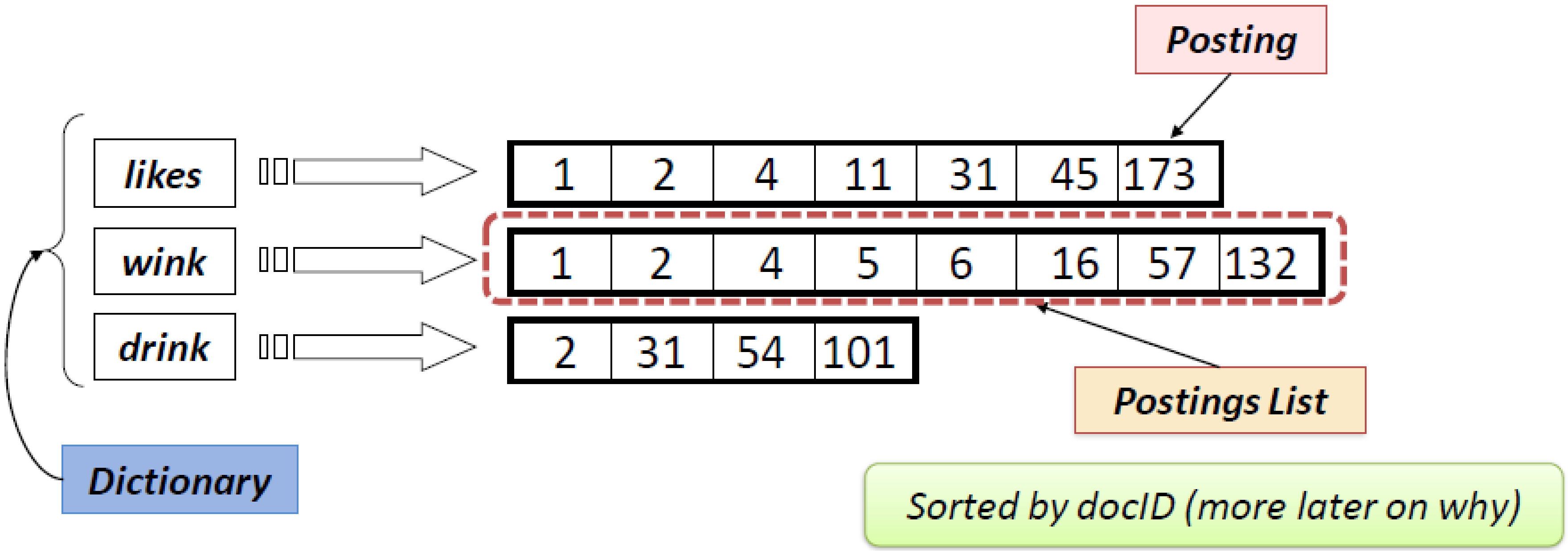
- Consider N = 1 million documents, each with about 1000 words
- Say there are M = 500K *distinct* terms among these.
- 500K x 1M matrix has half-a-trillion 0's and 1's.

- But it has no more than one billion 1's. **?**
  - matrix is extremely sparse.

What's a better representation?

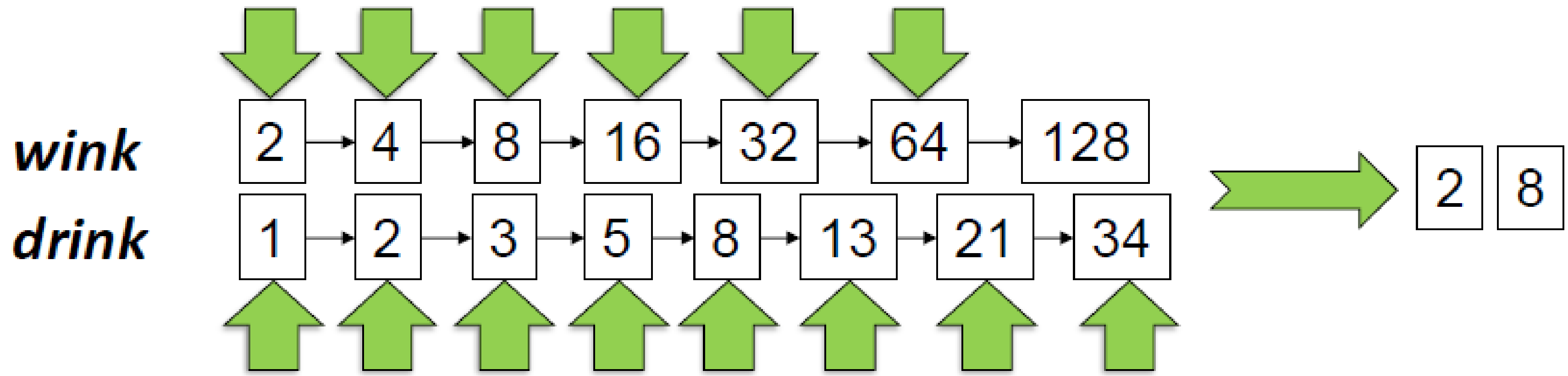
Inverted Index

- For each term *t*, we must store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number



Query Processing: AND

- Consider processing the query: ***wink*** *AND* ***dink***
  1. Locate ***likes*** in the Dictionary, Retrieve its postings
  2. Locate ***wink*** in the Dictionary, Retrieve its postings
  3. "Merge" the two postings lists



- Complexity ?

Complexity: If the list lengths are *x* and *y*, the merge takes **O(*x+y*)** operations.

Disadvantages of

Boolean Retrieval

- Boolean retrieval gives a “Boolean score” to each document!
- For query: ***qatar*** *AND* ***university***
  - If none or one of them only appeared in ***d*** ➔ ***d*** has a score of 0.
  - If both appeared in document ***d*** ➔ ***d*** has a score of 1.
- What if:

	d_1	d_2
qatar	2	17
university	3	13

term frequency

d_1 better than d_2?

(Unranked) Boolean Retrieval

- Every document that matches the query gets a score of 1

Boolean Model Limitations

1. Results are binary (relevant or not) with **no ranking**.
2. Does not handle **partial** matches well.
3. Users must **precisely** define queries.

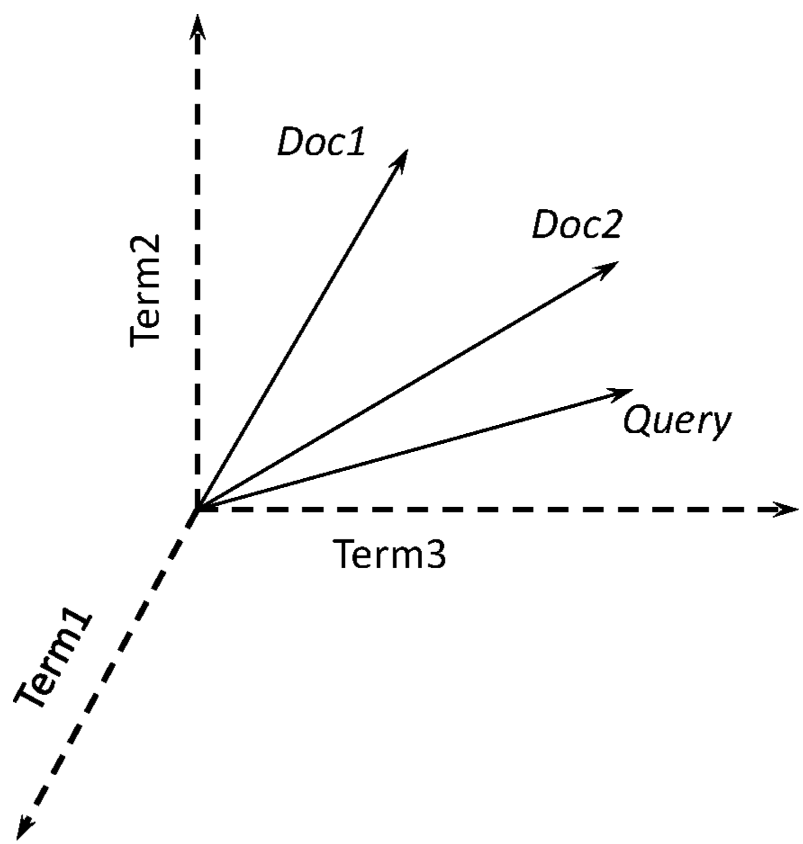
Vector Space Model

1. **Representing Documents and Queries as Vectors**
   Each document and query is represented as a vector in an **n-dimensional space**, where **n** is the number of unique terms in the corpus.
   - Each vector consists of **weights** assigned to terms, typically using **TF-IDF (Term Frequency-Inverse Document Frequency)**.

2. **Computing Cosine Similarity**
   The similarity between a document **d** and a query **q** is calculated using the **cosine similarity formula**.

Vector Space Model

- 3-d pictures useful, but can be misleading for high-dimensional space



Vector Space Model

- Documents ranked by **distance between points representing query and documents**
  - *Cosine Similarity* measure.

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

Vector Space Model

- The **Vector Space Model** represents documents and queries as vectors in a high-dimensional space. It measures relevance using **cosine similarity** between the query and document vectors.
- Documents and query represented by a vector of term weights
- Collection represented by a matrix of term weights

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \qquad Q = (q_1, q_2, \ldots, q_t)$$

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

Vector Space Model

Binary → Count → Weight Matrix

	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 $\in R^{|V|}$

Cosine Similarity

– Consider two documents D_1, D_2 and a query Q

- D_1 = (0.5, 0.8, 0.3), D_2 = (0.9, 0.4, 0.2), Q = (1.5,1.0, 0)

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

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

Term weighting Scheme

- Term weighting is a procedure that takes place during the text indexing process in order to assess the value of each term to the document.
- Term weighting is the assignment of numerical values to terms that represent their importance in a document in order to improve retrieval effectiveness

– Consider two documents D_1, D_2 and a query Q

- $D_1 = (0.5, 0.8, 0.3)$, $D_2 = (0.9, 0.4, 0.2)$, $Q = (1.5, 1.0, 0)$

Weights

Dataset (3 Documents and 1 Query)

Document ID	Content
D1	"Machine learning is great"
D2	"Deep learning improves AI"
D3	"AI and machine learning"

Query: `"machine learning"`

Step 1: Build the Term-Document Matrix

Term	D1	D2	D3	Query
Machine	1	0	1	1
Learning	1	1	1	1
Deep	0	1	0	0
AI	0	1	1	0
Great	1	0	0	0
Improves	0	1	0	0
And	0	0	1	0

Step 2: Compute Cosine Similarity

We calculate the **cosine similarity** between the query vector and each document vector.

Example for Document D1:

$$\cos(\theta) = \frac{(1 \times 1) + (1 \times 1)}{\sqrt{(1^2 + 1^2)} \times \sqrt{(1^2 + 1^2)}}$$

$$= \frac{1 + 1}{\sqrt{2} \times \sqrt{2}} = \frac{2}{2} = 1.0$$

Similarly, compute for D2 and D3.

Step 3: Rank Documents

Document	Cosine Similarity Score
D1	**1.0**
D3	**0.71**
D2	**0.5**

Thus, **D1 is the most relevant document.**

TFIDF Term Weights scheme

The weight of a term that occurs in a document is simply proportional to the term frequency.

- (TF) Term frequency weight measures importance in document:

$$\mathrm{tf}(t,d) = f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$$

- Where $\sum_{t' \in d} f_{t',d}$

is the total frequency of all terms in a document d

= Doc. Length

TFIDF Term Weights scheme

- (IDF) Inverse document frequency measures importance in collection:

$$idf(t, D) = \log \frac{N}{n_t}$$

- Where **N** is the total number of documents in corpus D   N = |D|
- $\mathbf{n}_t$ is number of documents where term t appear

TFIDF Term Weights scheme

– TFIDF:

Then tf–idf is calculated as

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D)$$

Example of tf–idf [edit]

Suppose that we have term count tables of a corpus consisting of only two documents, as listed on the right.

The calculation of tf–idf for the term "this" is performed as follows:

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

In its raw frequency form, tf is just the frequency of the "this" for each document. In each document, the word "this" appears once; but as the document 2 has more words, its relative frequency is smaller.

$$\mathrm{tf}(''\textsf{this}'', d_1) = \frac{1}{5} = 0.2$$
$$\mathrm{tf}(''\textsf{this}'', d_2) = \frac{1}{7} \approx 0.14$$

An idf is constant per corpus, and **accounts** for the ratio of documents that include the word "this". In this case, we have a corpus of two documents and all of them include the word "this".

$$\mathrm{idf}(''\textsf{this}'', D) = \log\left(\frac{2}{2}\right) = 0$$

So tf–idf is zero for the word "this", which implies that the word is not very informative as it appears in all documents.

$$\mathrm{tfidf}(''\textsf{this}'', d_1, D) = 0.2 \times 0 = 0$$
$$\mathrm{tfidf}(''\textsf{this}'', d_2, D) = 0.14 \times 0 = 0$$

The word "example" is more interesting - it occurs three times, but only in the second document:

$$\mathrm{tf}(''\textsf{example}'', d_1) = \frac{0}{5} = 0$$
$$\mathrm{tf}(''\textsf{example}'', d_2) = \frac{3}{7} \approx 0.429$$
$$\mathrm{idf}(''\textsf{example}'', D) = \log\left(\frac{2}{1}\right) = 0.301$$

Finally,

$$\mathrm{tfidf}(''\text{example}'', d_1, D) = \mathrm{tf}(''\text{example}'', d_1) \times \mathrm{idf}(''\text{example}'', D) = 0 \times 0.301 = 0$$
$$\mathrm{tfidf}(''\text{example}'', d_2, D) = \mathrm{tf}(''\text{example}'', d_2) \times \mathrm{idf}(''\text{example}'', D) = 0.429 \times 0.301 \approx 0.129$$

(using the base 10 logarithm).

Let's take an example to get a clearer understanding.
Sentence 1 : The car is driven on the road.
Sentence 2: The truck is driven on the highway.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

Vector Space Model

- Advantages
  - Simple computational framework for ranking
  - Any similarity measure or term weighting scheme could be used
  - **Supports Ranked Retrieval** – Unlike Boolean models, it ranks documents based on similarity.
  - **Handles Partial Matching** – Can retrieve relevant documents even if they don't contain all query terms.
- Disadvantages
  - Assumption of term independence
  - **High Dimensionality** – Large vocabularies result in high-dimensional vectors.
  - **Computationally Expensive** – Calculating cosine similarity for large datasets is costly

Review: Ranking

- **Ranking** is the process of selecting *which documents* to show the user, and *in what order*
- Rankers are generally developed with a certain **retrieval model** in mind. The retrieval model provides base-line assumptions about what relevance means:

➡ **Boolean Retrieval** models assume a document is entirely relevant or non- relevant, and compose queries using set operations (AND, OR, NOT, XOR, NOR, XNOR).

➡ **Vector Space Models** treat a document or a query as a vector of weights for each vocabulary word, and find document vectors that best match the query's vector.