# Information Retrieval and Web Search

## IR models: Vector Space Model

Instructor: Rada Mihalcea

[Note: Some slides in this set were adapted from an IR course taught by Ray Mooney at UT Austin (who in turn adapted them from Joydeep Ghosh), and from an IR course taught by Chris Manning at Stanford)

## Problems with Boolean Search

- Boolean queries often result in either too few (=0) or too many (1000s) results.
    - Query 1: "*standard user dlink 650*" → 200,000 hits
    - Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
    - Hard to tune precision vs. recall:
        - AND operator tends to produce high precision but low recall.
        - OR operator gives low precision but high recall.
        - Difficult/impossible to find satisfactory middle ground

## Problems with Boolean Search

- Good for expert users
  - With precise understanding of their needs and the collection
  - With good understanding of Boolean operators
- Good for applications:
  - Applications can easily consume 1000s of results
- Not good for the majority of users
  - Most users incapable of writing Boolean queries (or they are, but they think it’s too much work)
  - Most users don’t want to go through 1000s of results
    - This is particularly true of Web search

## Ranked Retrieval Models

- Rather than a set of documents satisfying a query expression, in **ranked retrieval**, the system returns an ordering over the (top) documents in the collection for a query
- **Free text queries**: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, these are two separate choices. In practice, ranked retrieval has normally been associated with free text queries and vice versa

## Size of Result Sets in Ranked Retrieval

- Ranked retrieval can return the same number of documents as in unranked retrieval
  BUT
- When a system produces a ranked result set, large result sets are not an issue
  - We just show the top *k* ( ≈ 10) results
  - User gets to see the most relevant documents, and can choose how many to consult
  - Premise: the ranking algorithm works

## Scoring as the Basis of Ranked Retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well the document and the query "match"
- In other words: Given a query, we need a way of assigning a score to a query/document pair

## Take 1: Jaccard coefficient

- Jaccard: A commonly used measure of overlap of two sets *A* and *B*
- Jaccard$(A,B) = |A \cap B| \,/\, |A \cup B|$
- 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.

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

## Exercise: Jaccard Coefficient

Assume the following query:

- Query: *march of dimes*

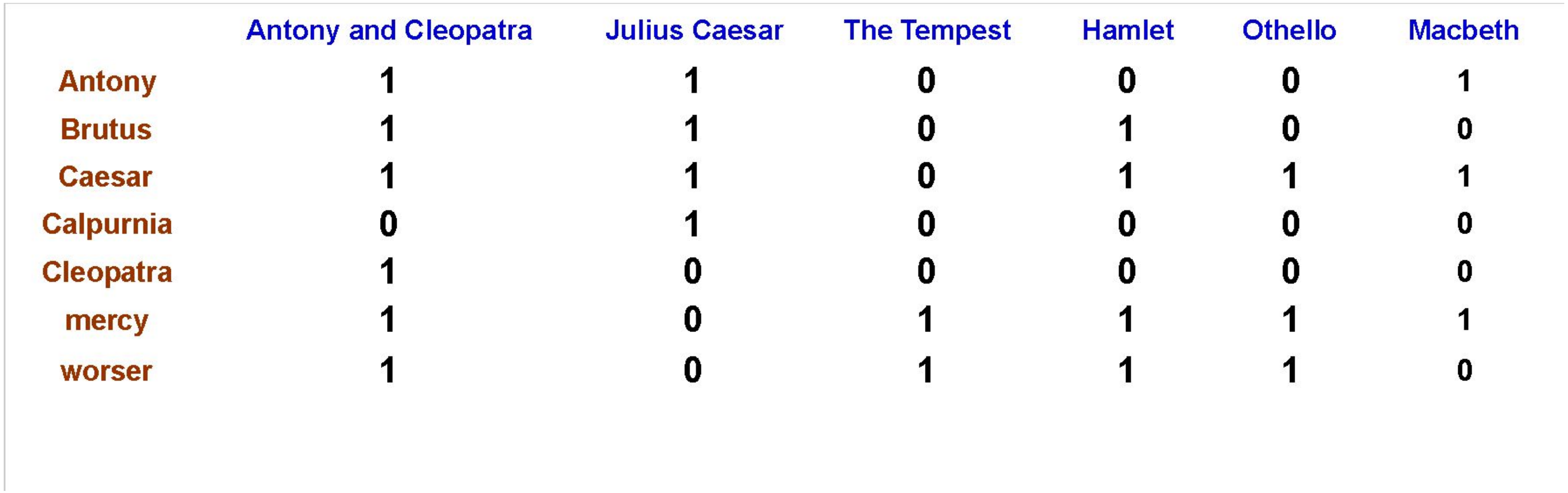
And the following two documents:

- Document 1: *caesar died in march*
- Document 2: *the long march*

## Issues with Jaccard for Scoring

- It does not consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard does not consider this information
- We need a more sophisticated way of normalizing for length

## Recall (from Boolean Retrieval): 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 |

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

## Term-document Count Matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector: a column below

| | 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 |

## Vector-Space Model

- *t* distinct terms remain after preprocessing
  - Unique terms that form the VOCABULARY
- These "orthogonal" terms form a vector space.

  Dimension = *t* = |vocabulary|
  - 2 terms -> bi-dimensional; …; n-terms -> n-dimensional
- Each term *i*, in a document (or query) *j*, is given a real-valued weight, $w_{ij.}$
- Both documents and queries are expressed as t-dimensional vectors:

  $d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$

## Vector-Space Model

Query as vector:

- We regard query as short document
- We return the documents ranked by the closeness of their vectors to the query, also represented as a vector.

- Vector-space model was originally developed and implemented in the SMART system (Salton, c. 1980) and standardly used by TREC participants and web IR systems

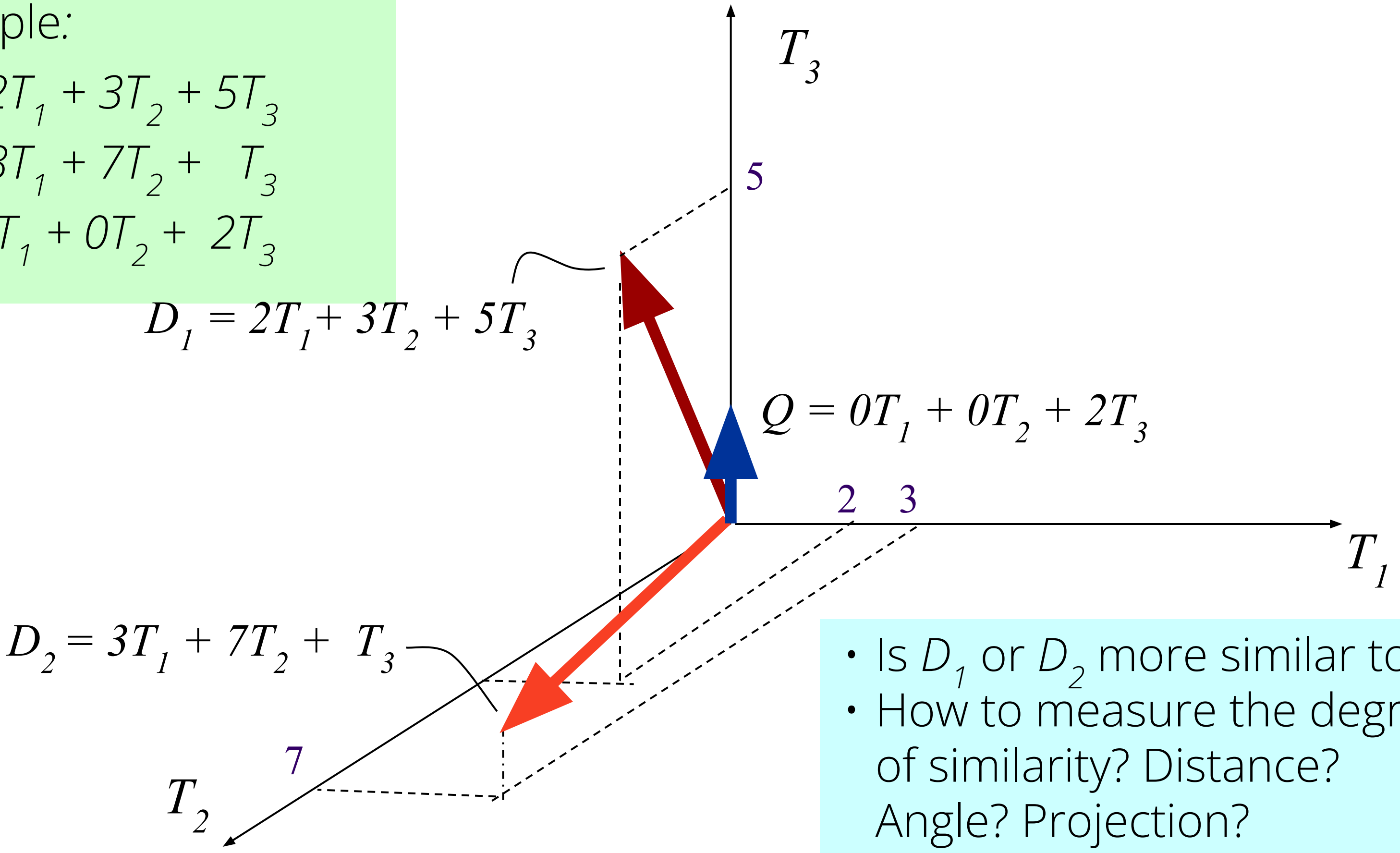
## Graphic Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$



- Is $D_1$ or $D_2$ more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

## Document Collection Representation

- A collection of *n* documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document.

$$\left(\begin{array}{ccccc}
 & T_1 & T_2 & \cdots & T_t \\
D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\
D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\
\vdots & \vdots & \vdots & & \vdots \\
\vdots & \vdots & \vdots & & \vdots \\
D_n & w_{1n} & w_{2n} & \cdots & w_{tn}
\end{array}\right)$$

## Term Frequency tf

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- More frequent terms in a document are more important, i.e. more indicative of the topic.
- May want to normalize *term frequency* (*tf*) :

$$tf_{t,d} = f_{t,d} \;/\; max\{f_{t,d}\}$$

- 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.

## Document Frequency

- Rare terms are more informative than frequent terms
  - Recall stopwords: very high-frequency words (e.g., function words: “a”, “the”, “in”, “to”; pronouns: “I”, “he”, “she”, “it”).
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*

  → We want a high weight for rare terms like *arachnocentric*.
- We will use document frequency (df) to capture this.

## Idf Weight

- $\mathrm{df}_t$ is the document frequency of *t*: the number of documents that contain *t*
  - $\mathrm{df}_t$ is an inverse measure of the informativeness of *t*
- N is the total number of documents in the collection
  - $\mathrm{df}_t \leq N$
- We define the idf (inverse document frequency) of *t* by

$$\mathrm{idf}_t = \log_{10}\,(N/\mathrm{df}_t)$$

  - We use log ($N/\mathrm{df}_t$) instead of $N/\mathrm{df}_t$ to “dampen” the effect of idf.

## idf example, suppose *N* = 1 million

| term | $\text{df}_t$ | $\text{idf}_t$ |
|---|---:|---|
| calpurnia | 1 | |
| animal | 100 | |
| sunday | 1,000 | |
| fly | 10,000 | |
| under | 100,000 | |
| the | 1,000,000 | |

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

There is one idf value for each term *t* in a collection.

What is $\text{idf}_{\text{the}}$ ?

## Collection vs. Document Frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example:

| Word | Collection frequency | Document frequency |
|---|---|---|
| *insurance* | 10440 | 3997 |
| *try* | 9800 | 8760 |

- Which word is a better search term (and should get a higher weight)?

## tf-idf weighting

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

$$\mathrm{w}_{t,d} = \mathrm{tf}_{t,d} \times \log_{10}(N / \mathrm{df}_t)$$

- Best known weighting scheme in information retrieval
  - Theoretically proven to work well
  - Note: the "-" often used in "tf-idf" notation 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

## Computing tf-idf: An Example

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and

document frequencies of these terms are:

A(50), B(2500), C(100)

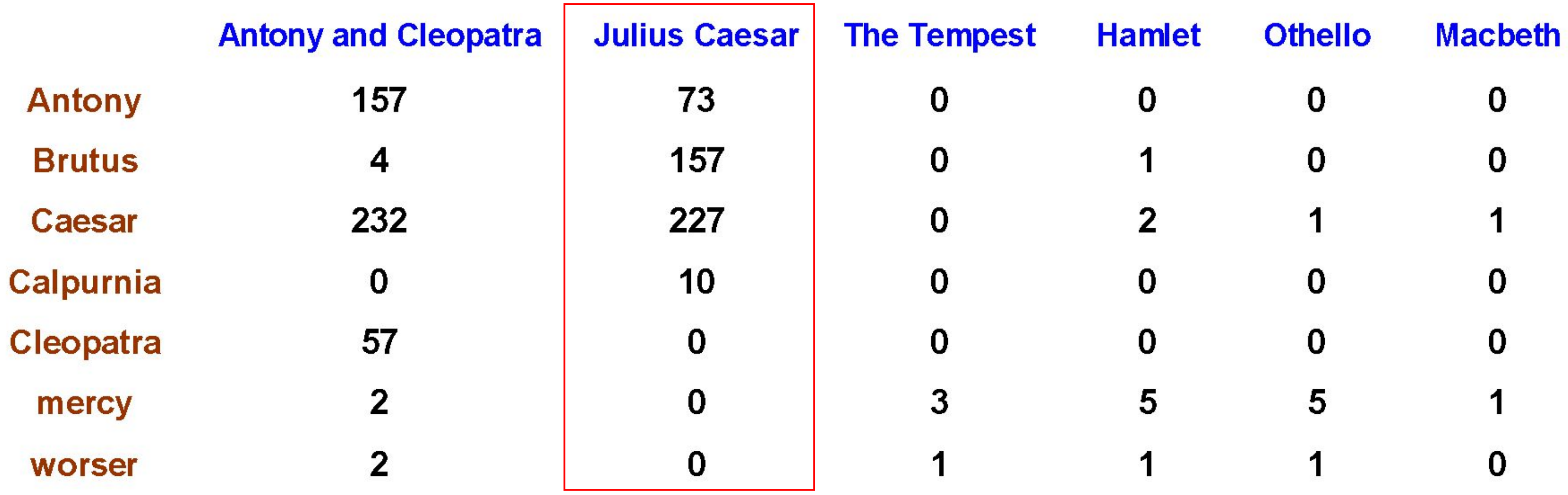
Then (assuming $\log_{10}$)

A: tf = 3/3; idf = log(10000/50) = 2.3; tf-idf = 2.3

B: tf = 2/3; idf = log(10000/2500) = 0.60; tf-idf = 0.40

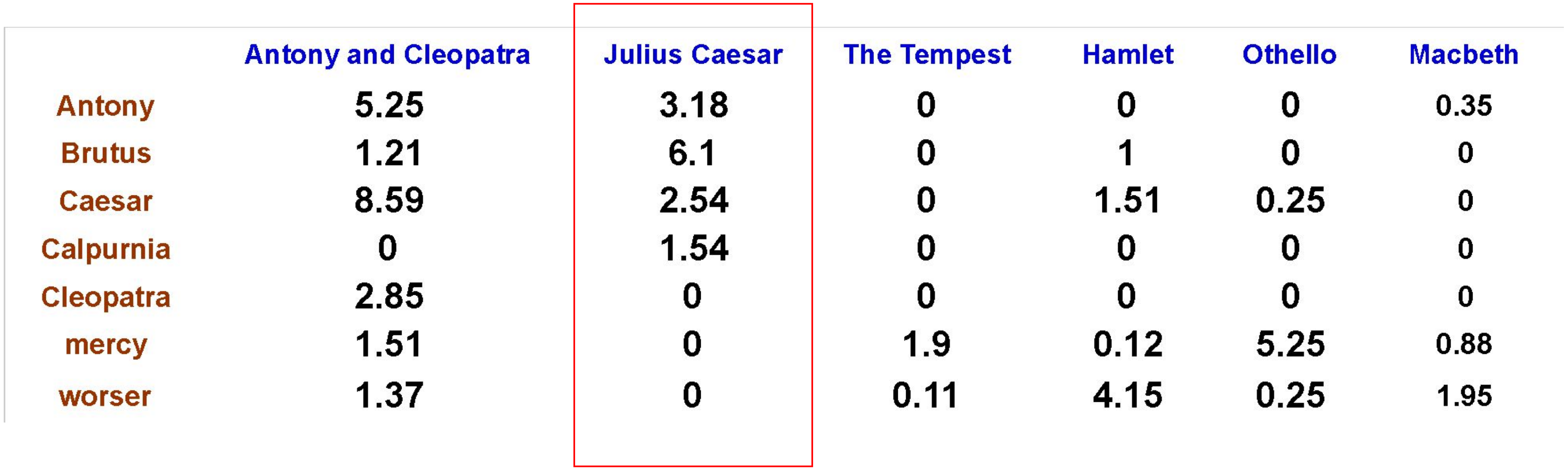
C: tf = 1/3; idf = log(10000/100) = 2.0; tf-idf = 0.66

## Back to the Term-document Count Matrix



| | 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 |

## Binary (Boolean) → Count → Term Weight



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

## Documents and Queries 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: tens of millions of dimensions when you apply this to a web search engine
  - These are very sparse vectors - most entries are zero

- Query is also typically treated as a document and also tf-idf weighted.

## Similarity Measure

- We now have vectors for all documents in the collection, a vector for the query, how to compute similarity?
- A similarity measure is a function that computes the degree of similarity between two vectors.
- Using a similarity measure between the query and each document:
  - It is possible to rank the retrieved documents in the order of presumed relevance.
  - It is possible to enforce a certain threshold so that the size of the retrieved set can be controlled.

## First Cut: Euclidean Distance

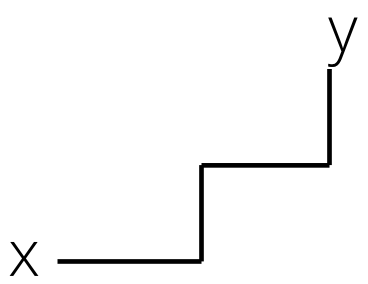
- Distance between vectors X and Y is the length of the vector |X – Y|.
  - Euclidean distance

$$EuclideanDist(X,Y) = \sqrt{\sum(x_i - y_i)^2}$$

- **Exercise**: Determine the Euclidean distance between the vectors (0, 3, 2, 1, 10) and (2, 7, 1, 0, 0)

## Second Cut: Manhattan Distance

- Or "city block" measure
  - Based on the idea that generally in American cities you cannot follow a direct line between two points.



- Uses the formula:

$$ManhDist(X,Y) = \sum_{i=1}^{n} | x_i - y_i |$$

- **Exercise**: Determine the Manhattan distance between the vectors (0, 3, 2, 1, 10) and (2, 7, 1, 0, 0)

## Why are These Measures Not a Great Idea?

- Documents with no terms in common are not penalized by the measure
    - E.g., documents (1,0) and (0,1) have the same Manhattan distance measure as documents (1,0) and (3,0)
- We still haven’t dealt with the issue of length normalization
    - Long documents would be more similar to each other by virtue of length, not topic

## Third Cut: Inner Product

- Similarity between vectors for the document $\boldsymbol{d}_i$ and query $\boldsymbol{q}$ can be computed as the vector inner product:

$$\text{sim}(\boldsymbol{d}_j,\boldsymbol{q}) = \boldsymbol{d}_j \bullet \boldsymbol{q} = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

where $w_{ij}$ is the weight of term *i* in document *j* and $w_{iq}$ is the weight of term *i* in the query q

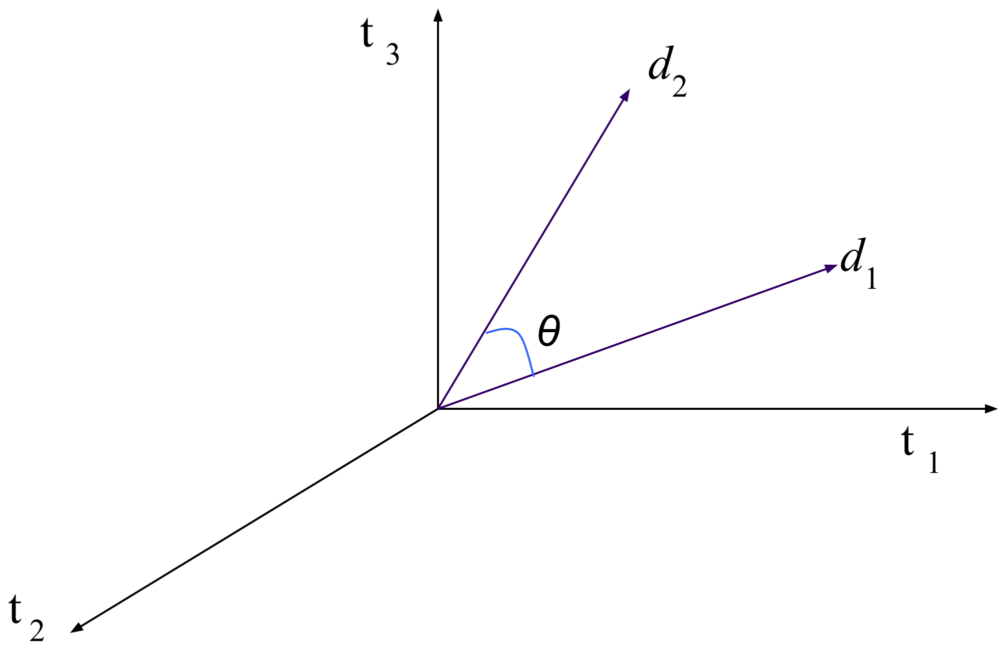
- For binary vectors, the inner product is the number of matched query terms in the document (size of intersection).
- For weighted term vectors, it is the sum of the products of the weights of the matched terms.

## Properties of Inner Product

- Favors long documents with a large number of unique terms.
  - Again, the issue of normalization
- Measures how many terms matched but not how many terms are *not* matched.

## Cosine Similarity

- Distance between vectors $d_1$ and $d_2$ *captured* by the cosine of the angle $x$ between them.
- Note – this is *similarity*, not distance



## Cosine Similarity

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right| \left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

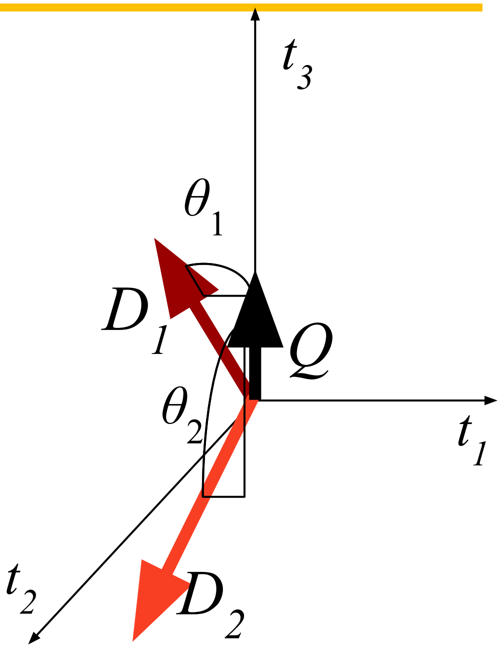
- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors
- So the cosine measure is also known as the *normalized inner product*

$$\text{Length } \left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}^2}$$

## Cosine Similarity

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\text{CosSim}(\boldsymbol{d}_j, \boldsymbol{q}) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim($D_1$ , $Q$) = 10 / $\sqrt{(4+9+25)(0+0+4)}$ = 0.81

$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim($D_2$ , $Q$) = 2 / $\sqrt{(9+49+1)(0+0+4)}$ = 0.13

$Q = 0T_1 + 0T_2 + 2T_3$

## Exercise: Cosine Similarity

- *Exercise: Rank the following by decreasing cosine similarity:*
  - (P1) Two documents that have only frequent words ***(the, a, an, of)*** in common.
  - (P2) Two documents that have no words in common.
  - (P3) Two documents that have many rare words in common ***(wingspan, tailfin).***

## Side Note: Cosine Similarity can be Used for Authorship Attribution

- Documents: *Sense and Sensibility*, *Pride and Prejudice*, *Wuthering Heights*

| term | SaS | PaP | WH |
|---|---|---|---|
| affection | 115 | 58 | 20 |
| jealous | 10 | 7 | 11 |
| gossip | 2 | 0 | 6 |
| wuthering | 0 | 0 | 38 |

| 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(SaS,PaP) ≈ 0.789 × 0.832 + 0.515 × 0.555 + 0.335 × 0.0 + 0.0 × 0.0 ≈ 0.94
- cos(SaS,WH) ≈ 0.79
- cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

## Exercise

Consider the following four documents:

- **Doc1**: apple orange banana peach
- **Doc2**: orange orange apple apple
- **Doc3**: banana tangerine peach
- **Doc4**: peach peach apple banana

And the query:

- **Query**: apple peach tangerine

Assume no pre-processing (i.e., no normalization, no case-folding, no stopword-removal, no stemming), and a tfidf weighting scheme (tf not normalized, log based 10 for idf).

1. Write the vector representations for each of the four documents and for the query.

2. Determine the cosine similarities between each document and the query, and rank the documents.

## Comments on Vector Space Models

- Simple, mathematically based approach.
- Considers both local (*tf*) and global (*idf*) word occurrence frequencies.
- Provides partial matching and ranked results.
- Tends to work quite well in practice despite obvious weaknesses.
- Allows efficient implementation for large document collections.

## Problems with Vector Space Model

- Missing semantic information (e.g., word sense).
- Missing syntactic information (e.g., phrase structure, word order, proximity information).
- Assumption of term independence
- Lacks the control of a Boolean model (e.g., *requiring* a term to appear in a document).
  - Given a two-term query “A B”, may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.

## Evaluation of IR Models
## Precision & Recall

## Standard Evaluation Measures

Start with a CONTINGENCY table

| | retrieved | not retrieved | |
|---|---|---|---|
| relevant | $w$ | $x$ | $n_1 = w + x$ |
| not relevant | $y$ | $z$ | |
| | $n_2 = w + y$ | | |

$N = \text{total documents} = w + x + y + z$

# Precision and Recall

From all the documents that are relevant out there, how many did the IR system retrieve?

Recall: $\frac{w}{w+x}$

From all the documents that are retrieved by the IR system how many are relevant?

Precision: $\frac{w}{w+y}$

## Exercise

- Assume a collection of 10 documents: D1, D2, D3, ... , D10
- Assume a query Q with relevant documents D3, D5, D7
- Assume a system that returns the following documents: D2, D7, D3, D10
- What is the precision and recall of this system?

## Exercise

- Assume two systems A and B, A has a recall of 23%, B has a recall of 29%. Which system is better?
  - A
  - B
  - Cannot tell

- What is the maximum recall that a system can get?
  - How?

## Precision and Recall for a Set of Queries

- For each query, determine the retrieved documents and the relevant documents

- Calculate
  - Macro-average: average the P/R calculated for the individual queries
  - Micro-average: sum all/relevant documents for individual queries, and calculate P/R only once

## Exercise

- Assume a collection of 10 documents: D1, D2, D3, ... , D10 and a system that returns five ranked documents
- Assume a query Q1:
  - relevant documents D3, D5, D7
  - system returns documents D5, D1, D7
- Assume a query Q2
  - relevant documents D6, D7
  - system returns documents D1, D2, D3, D4, D6
- Assume a query Q3
  - relevant documents D1, D6, D8, D9
  - system returns D6, D8, D2, D3
- What is the macro-average precision and recall of this system? How about the micro-average precision and recall?

## Term Weighting Approaches

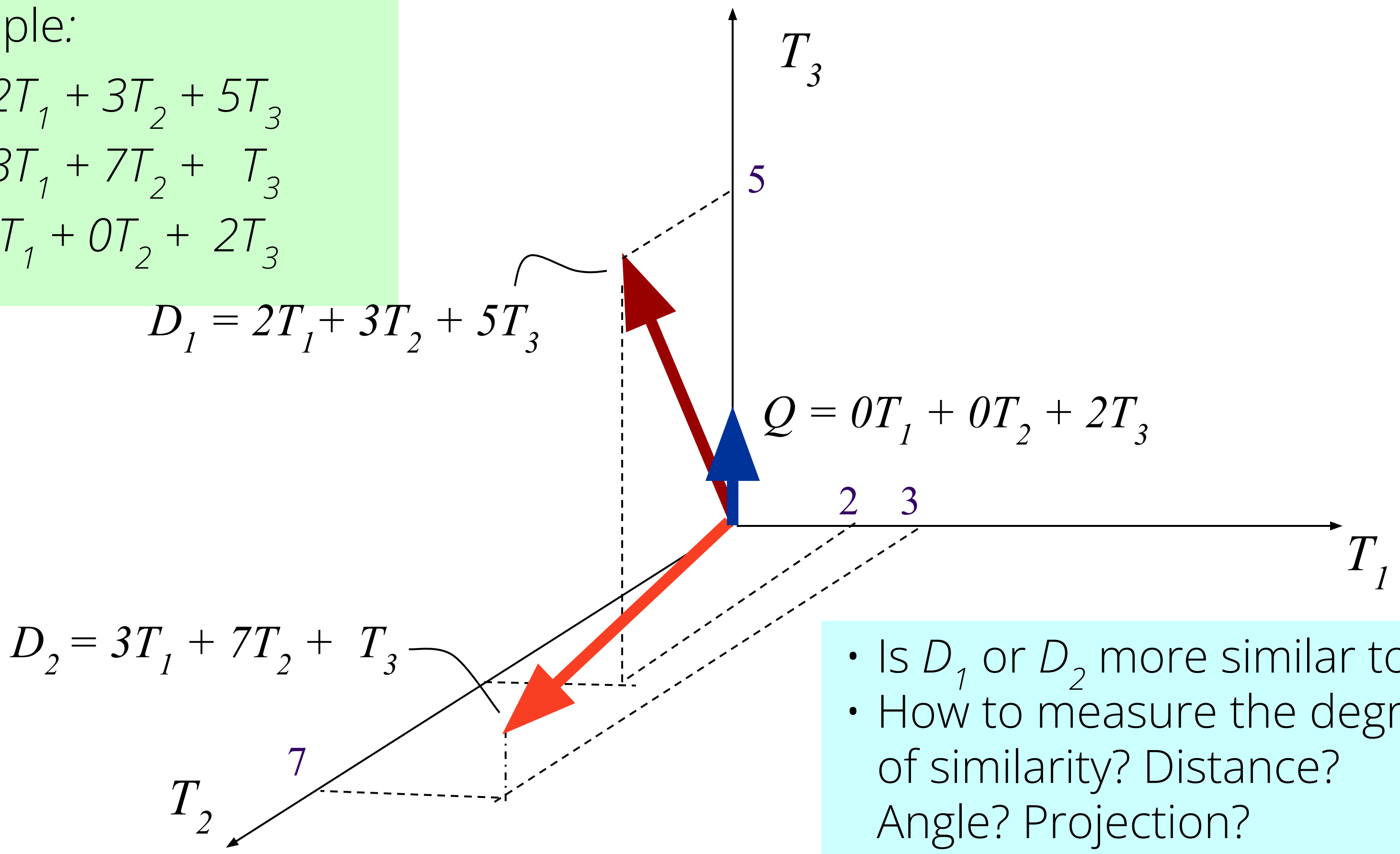
# Graphic Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$



- Is $D_1$ or $D_2$ more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

## Document Collection Representation

- A collection of *n* documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document.

$$\begin{pmatrix} & T_1 & T_2 & \cdots & T_t \\ D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\ D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \cdots & w_{tn} \end{pmatrix}$$

## Term Weighting Experiments

- Salton & Buckley: Experiments with term weighting approaches
- Six collections, 1,800 term weighting approaches
  - 287 combinations found to be distinct
- Comparative evaluations using:
  - Ranking: the lower the better
  - Best weighting scheme has a rank of 1
- Average search precision:
  - Average of precisions for recall points of 0.25, 0.50, 0.75
- Average across queries:
  - Macro-average: average of the average search precisions

## IR Test Collections

- CACM (articles from 'Communications of the ACM journal): 3204 D(ocuments), 64 Q(ueries)
- CISI (articles about information sciences): 1460 D, 116 Q
- CRAN (abstracts from aeronautics articles): 1398 D, 225 Q
- INSPEC (articles in computer engineering): 12684 D, 84 Q
- MED (medical articles): 1033 D, 30 Q
- NPL (articles about electrical engineering): 11429 D, 100 Q

## Components Tested

Term Frequency

- b: 1
- t: tf
- n: normalized tf

Inverse Document Frequency

- x: 1
- f: log (N/n)
- p: log ( (N-n)/n)

Cosine Similarity Normalization with Document Length

- x: 1
- c: Length $\left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}^2}$

Note: these components can be chosen separately for the document and for the query

## Sample Weighting Schemes

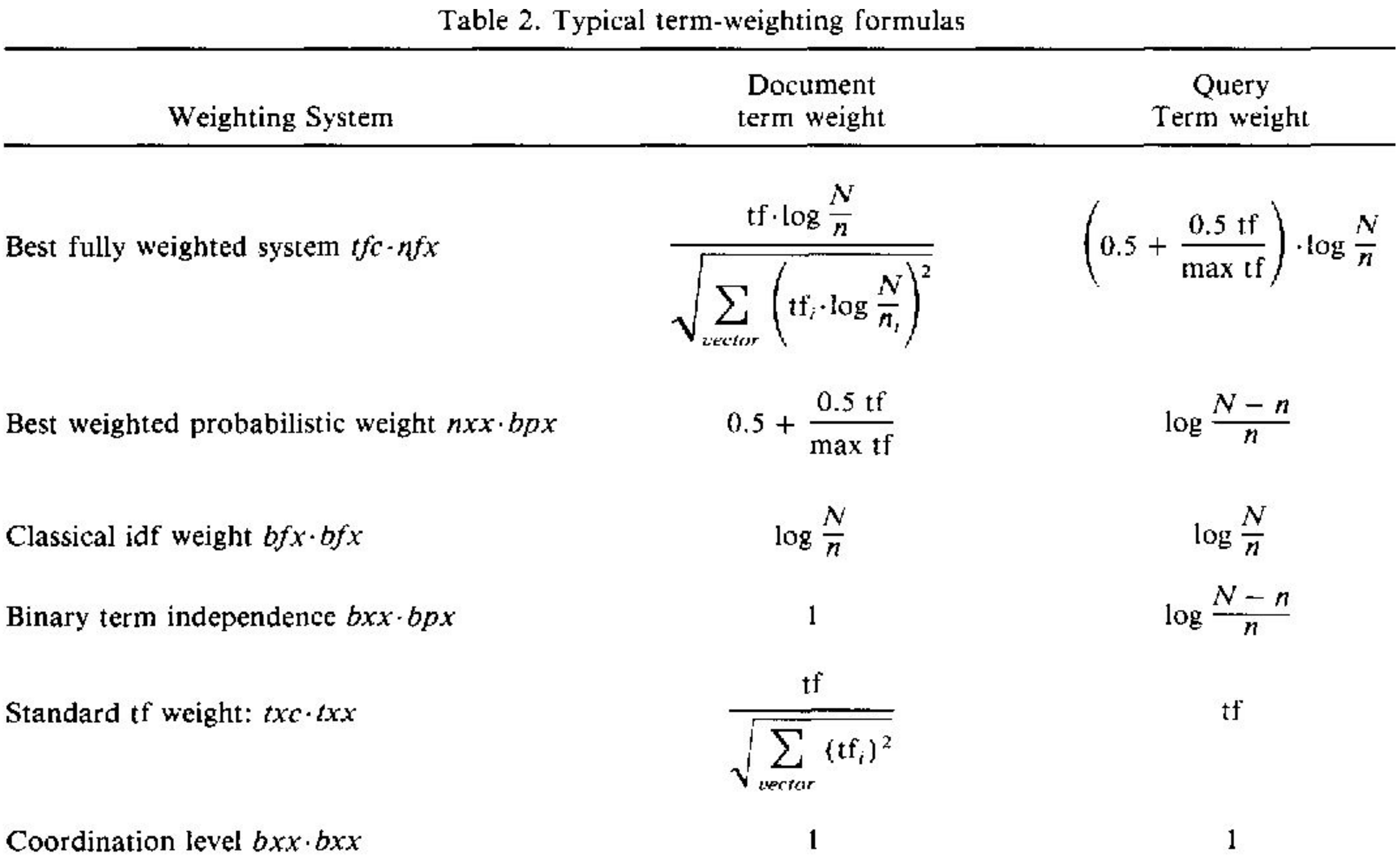
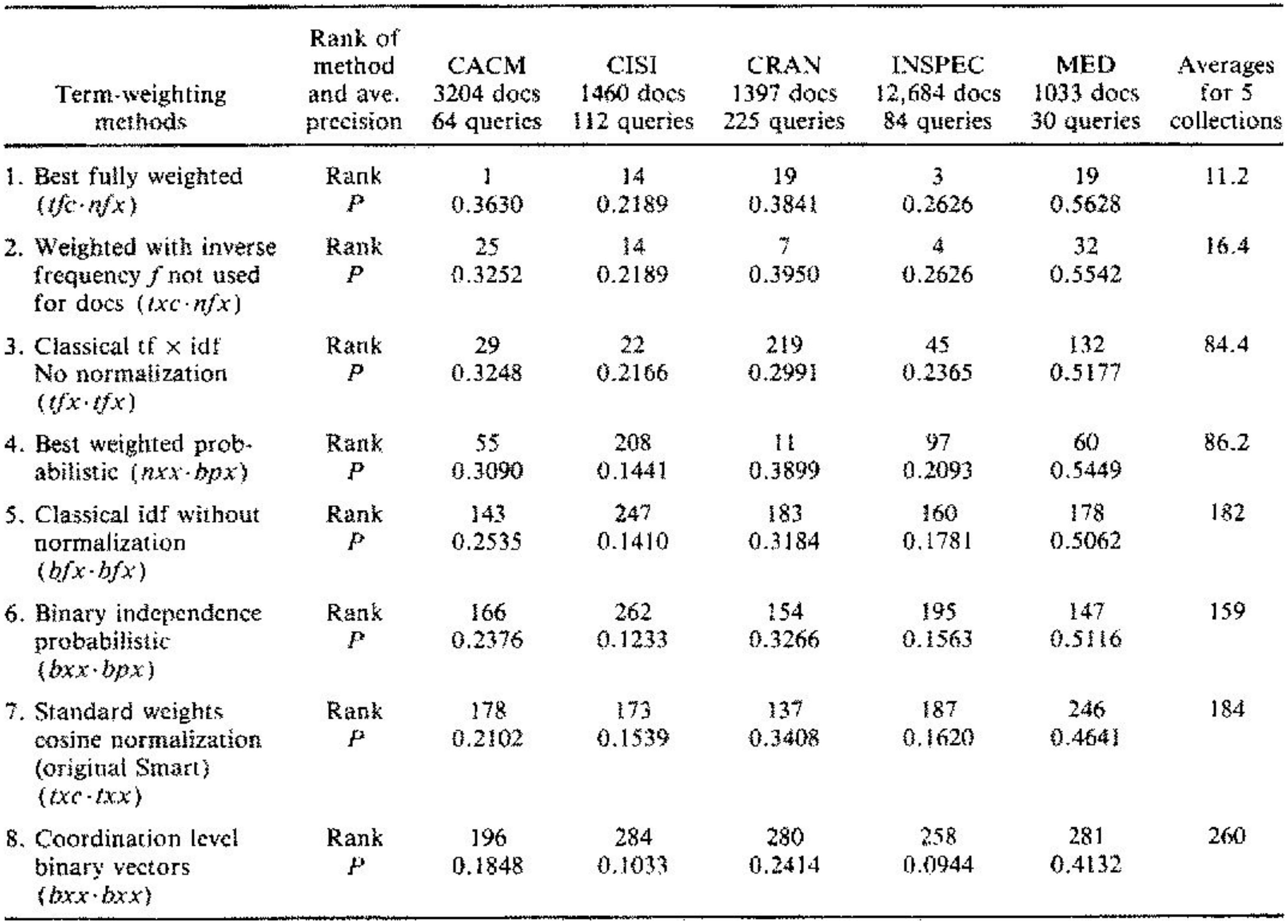


Table 2. Typical term-weighting formulas

| Weighting System | Document term weight | Query Term weight |
|---|---|---|
| Best fully weighted system *tfc·nfx* | $\dfrac{\mathrm{tf}\cdot\log\frac{N}{n}}{\sqrt{\sum_{vector}\left(\mathrm{tf}_i\cdot\log\frac{N}{n_i}\right)^2}}$ | $\left(0.5+\dfrac{0.5\ \mathrm{tf}}{\max\ \mathrm{tf}}\right)\cdot\log\dfrac{N}{n}$ |
| Best weighted probabilistic weight *nxx·bpx* | $0.5+\dfrac{0.5\ \mathrm{tf}}{\max\ \mathrm{tf}}$ | $\log\dfrac{N-n}{n}$ |
| Classical idf weight *bfx·bfx* | $\log\dfrac{N}{n}$ | $\log\dfrac{N}{n}$ |
| Binary term independence *bxx·bpx* | 1 | $\log\dfrac{N-n}{n}$ |
| Standard tf weight: *txc·txx* | $\dfrac{\mathrm{tf}}{\sqrt{\sum_{vector}(\mathrm{tf}_i)^2}}$ | tf |
| Coordination level *bxx·bxx* | 1 | 1 |

# Performance Results



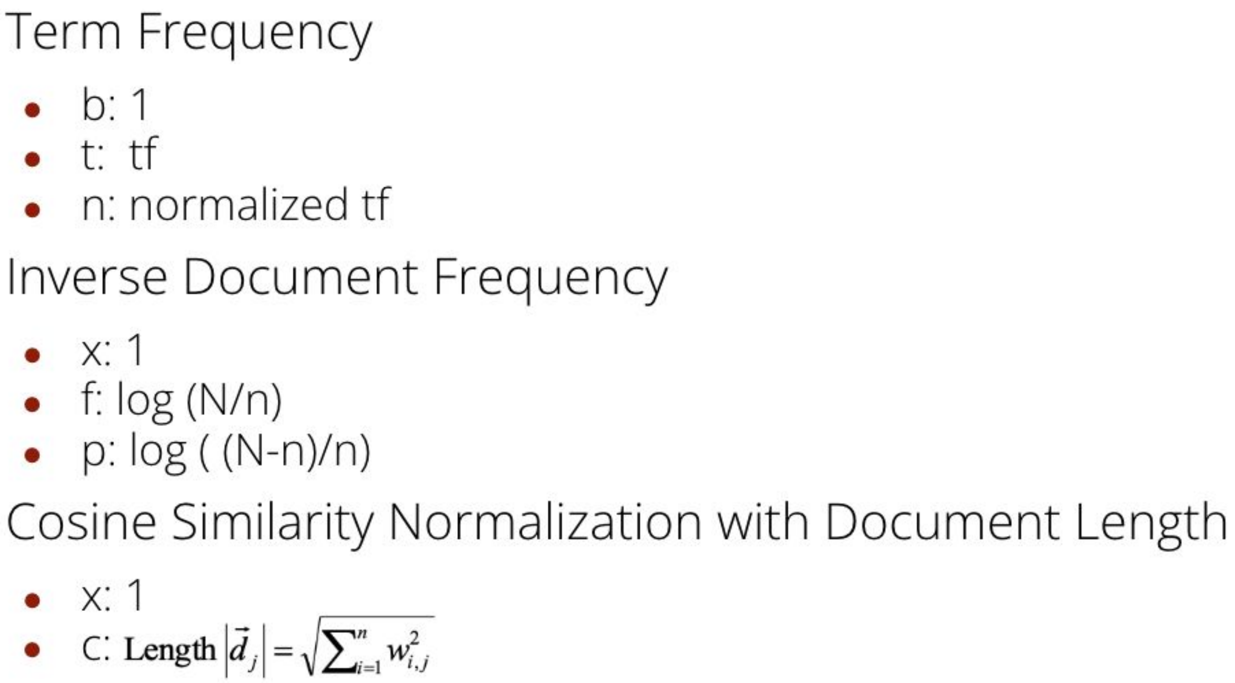
| Term-weighting methods | Rank of method and ave. precision | CACM 3204 docs 64 queries | CISI 1460 docs 112 queries | CRAN 1397 docs 225 queries | INSPEC 12,684 docs 84 queries | MED 1033 docs 30 queries | Averages for 5 collections |
|---|---|---|---|---|---|---|---|
| 1. Best fully weighted ($tfc \cdot nfx$) | Rank | 1 | 14 | 19 | 3 | 19 | 11.2 |
| | $P$ | 0.3630 | 0.2189 | 0.3841 | 0.2626 | 0.5628 | |
| 2. Weighted with inverse frequency $f$ not used for docs ($txc \cdot nfx$) | Rank | 25 | 14 | 7 | 4 | 32 | 16.4 |
| | $P$ | 0.3252 | 0.2189 | 0.3950 | 0.2626 | 0.5542 | |
| 3. Classical tf × idf No normalization ($tfx \cdot tfx$) | Rank | 29 | 22 | 219 | 45 | 132 | 84.4 |
| | $P$ | 0.3248 | 0.2166 | 0.2991 | 0.2365 | 0.5177 | |
| 4. Best weighted probabilistic ($nxx \cdot bpx$) | Rank | 55 | 208 | 11 | 97 | 60 | 86.2 |
| | $P$ | 0.3090 | 0.1441 | 0.3899 | 0.2093 | 0.5449 | |
| 5. Classical idf without normalization ($bfx \cdot bfx$) | Rank | 143 | 247 | 183 | 160 | 178 | 182 |
| | $P$ | 0.2535 | 0.1410 | 0.3184 | 0.1781 | 0.5062 | |
| 6. Binary independence probabilistic ($bxx \cdot bpx$) | Rank | 166 | 262 | 154 | 195 | 147 | 159 |
| | $P$ | 0.2376 | 0.1233 | 0.3266 | 0.1563 | 0.5116 | |
| 7. Standard weights cosine normalization (original Smart) ($txc \cdot txx$) | Rank | 178 | 173 | 137 | 187 | 246 | 184 |
| | $P$ | 0.2102 | 0.1539 | 0.3408 | 0.1620 | 0.4641 | |
| 8. Coordination level binary vectors ($bxx \cdot bxx$) | Rank | 196 | 284 | 280 | 258 | 281 | 260 |
| | $P$ | 0.1848 | 0.1033 | 0.2414 | 0.0944 | 0.4132 | |

## Lessons Learned

- Term weighting DOES matter
- Query vector:
  - Term frequency
    - Use n for short queries
    - Use t for longer queries that require better discrimination among terms
  - Document frequency
    - Use f
  - Do not do normalization with query length

## Exercise

- Assume a system that only receives single term queries. What weighting formula should be used for the queries, using the components defined earlier?
  - tfx
  - tfc
  - bfx



Term Frequency

- b: 1
- t: tf
- n: normalized tf

Inverse Document Frequency

- x: 1
- f: log (N/n)
- p: log ( (N-n)/n)

Cosine Similarity Normalization with Document Length

- x: 1
- C: Length $\left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}^2}$

## Lessons Learned

- Document vectors:
  - Term-frequency:
    - For technical vocabulary (e.g., CRAN) use n
    - For more varied vocabulary, use t
    - For short document vectors, use b
  - Document frequency:
    - Inverse document-frequency f is similar to probabilistic term weight p: typically use f
    - For dynamic collections with many changes in the document collection makeup, use x
  - Normalization:
    - Typically use c (in particular when there is high deviation in vector length)
    - For short documents with homogenous length, use x

# Exercise

Consider the following 4 documents:

- **Doc1**: apple orange banana peach
- **Doc2**: orange orange apple apple
- **Doc3**: banana tangerine peach
- **Doc4**: peach peach apple banana

And the query:

- **Query**: apple peach tangerine

Assume no pre-processing (i.e., no normalization, no case-folding, no stopword-removal, no stemming).

What are the vector representations for the following weighting schemes:

tfc.tfc

bxx.bxx

nxx.bfx

Term Frequency

- b: 1
- t: tf
- n: normalized tf

Inverse Document Frequency

- x: 1
- f: log (N/n)
- p: log ( (N-n)/n)

Cosine Similarity Normalization with Document Length

- x: 1
- C: Length $|\vec{d}_j| = \sqrt{\sum_{i=1}^{n} w_{i,j}^2}$

## Vector Space Model Implementation

# Naïve Implementation

```
Convert all documents in collection D to TF.IDF weighted
    vectors, $\boldsymbol{d}_j$, for keyword vocabulary V.
Convert query to a tf-idf-weighted vector $\boldsymbol{q}$.
For each $\boldsymbol{d}_j$ in D do
    Compute score $s_j = \text{cosSim}(\boldsymbol{d}_j, \boldsymbol{q})$
Sort documents by decreasing score.
Present top ranked documents to the user.
```

Time complexity: $O(|V|\cdot|D|)$ Bad for large V & D !

$|V| = 10{,}000$; $|D| = 100{,}000$; $|V|\cdot|D| = 1{,}000{,}000{,}000$

## Practical Implementation

- Based on  the observation that documents containing none of the query keywords do not affect the final ranking
- Try to identify only those documents that contain at least one query keyword
- Actual implementation of an inverted index

## Step 1: Preprocessing

- Implement the preprocessing functions:
  - For tokenization
  - Other pre-processing
    - Stopword removal
    - Stemming

- Input: Documents that are read one by one from the collection

- Output: Tokens to be added to the index
  - No punctuation, no stop-words, stemmed

## Step 2: Indexing

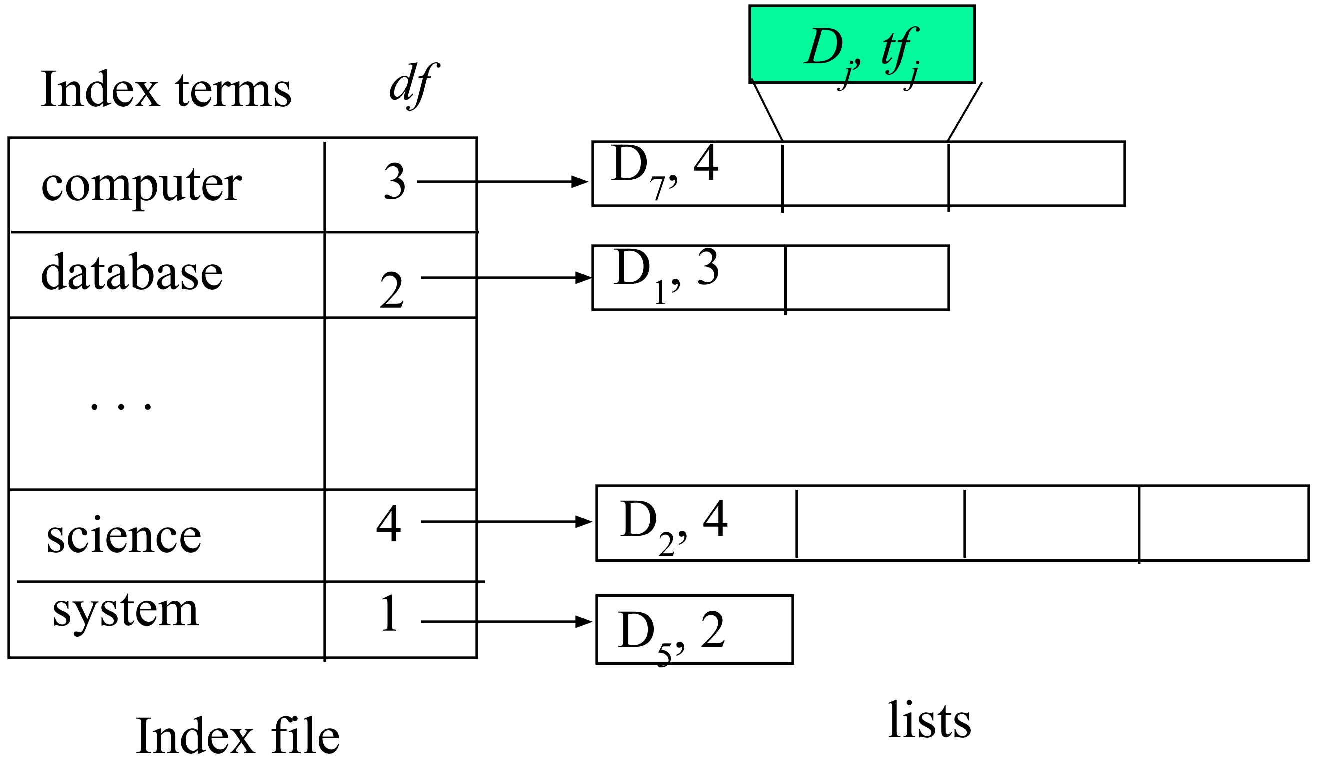
- Build an inverted index, with an entry for each word in the vocabulary

- Input: Tokens obtained from the preprocessing module
- Output: An inverted index for fast access

## Step 2 (continued)

- Many data structures are appropriate for fast access
- We need:
  - One entry for each word in the vocabulary
  - For each such entry:
    - Keep a list of all the documents where it appears together with the corresponding frequency □ TF
  - For each such entry, keep the total number of occurrences in all documents:
    - □ IDF

## Step 2 (continued)



## Step 2 (continued)

- TF and IDF for each token can be computed in one pass
- Cosine similarity also requires document lengths
- Need a second pass to compute document vector lengths
  - Remember that the length of a document vector is the square-root of sum of the squares of the weights of its tokens.
  - Remember the weight of a token is: TF * IDF
  - Therefore, must wait until IDFs are known (and therefore until all documents are indexed) before document lengths can be determined.
- Do a second pass over all documents: keep a list or hashtable with all document id-s, and for each document determine its length.

## Time Complexity of Indexing

- Complexity of creating vector and indexing a document of $n$ tokens is $O(n)$.
- So indexing $|D|$ such documents is $O(|D|\, n)$.
- Computing token IDFs can be done during the same first pass
- Computing vector lengths is also $O(|D|\, n)$.
- Complete process is $O(|D|\, n)$, which is also the complexity of just reading in the corpus.

## Step 3: Retrieval

- Use inverted index (from step 2) to find the limited set of documents that contain at least one of the query words.
- Incrementally compute cosine similarity of each indexed document as query words are processed one by one.
  - If tf is normalized for the query, may need to first read all the query tokens
- To accumulate a total score for each retrieved document, store retrieved documents in a hashtable (or another search data structure), where the document id is the key, and the partial accumulated score is the value.
- <u>Input</u>: Query and Inverted Index (from Step 2)
- <u>Output</u>: Similarity values between query and documents

# Step 4: Ranking

- Sort the search structure including the retrieved documents based on the value of cosine similarity
- Return the documents in descending order of their relevance

- <u>Input</u>: Similarity values between query and documents
- <u>Output</u>: Ranked list of documented in reversed order of their relevance