# Ranking Models

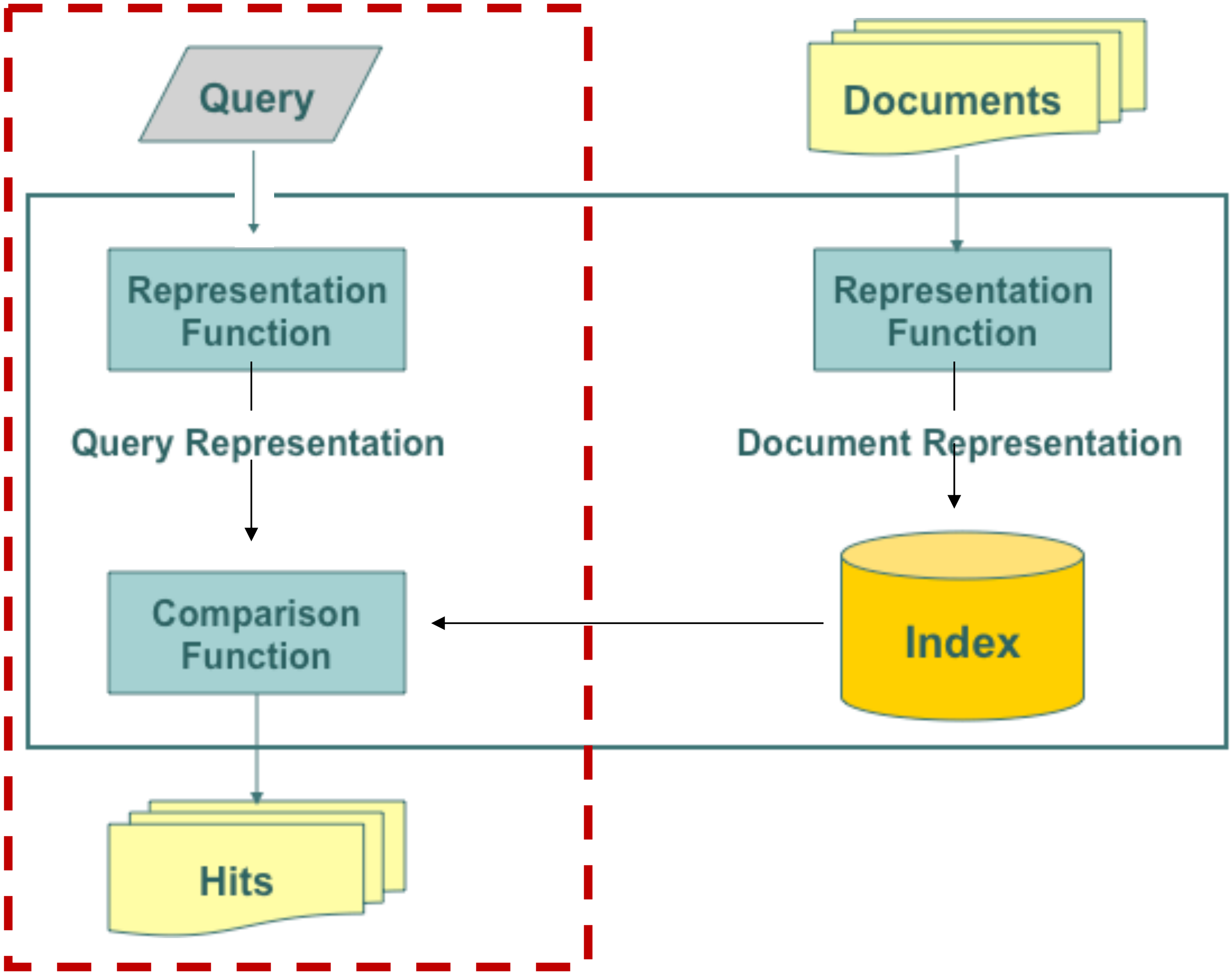
Basics

by

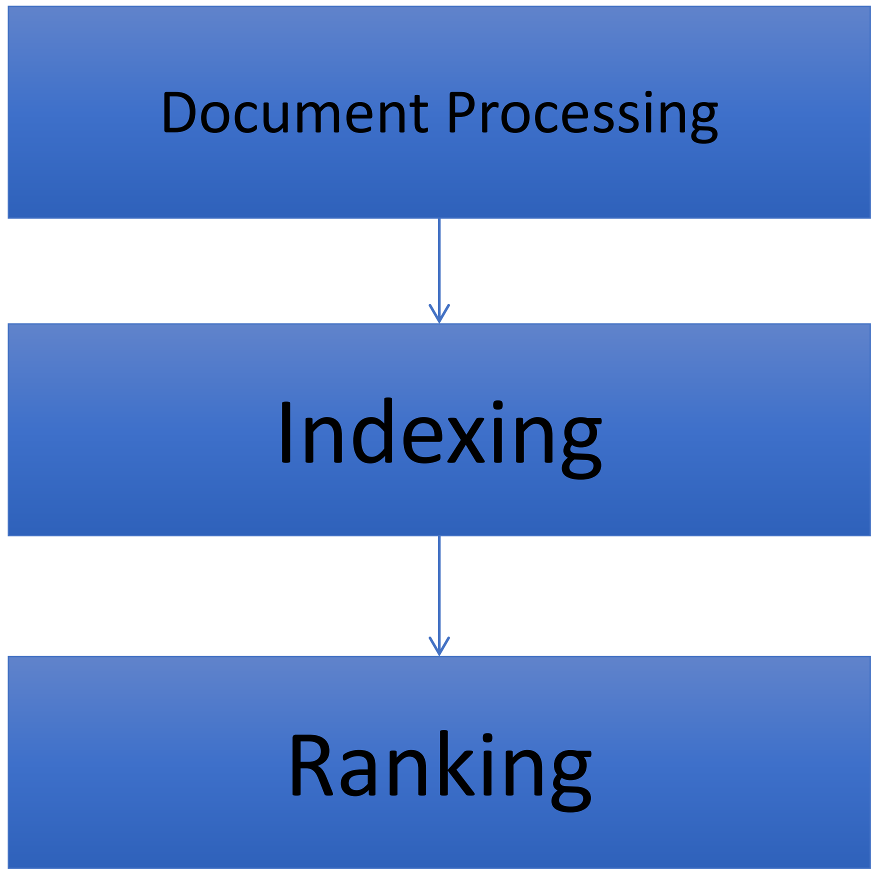
Giovanni Stilo, PhD.

giovanni.stilo@univaq.it

# Retrieval Phase



# Workflow



# Boolean Model

# Boolean Model

- Simple model based on set theory
- First model used in “classic” IR systems
- Queries and documents specified as boolean expressions :
  - precise semantics
  - *E.g.*, $q = k_a \wedge (k_b \vee \neg k_c)$
  - *(apple* $\wedge$ *(computer* $\vee$ $\neg$red)
- Terms can be present or absent. Thus, $w_{ij} \in \{0,1\}$

# Example

- $q = a \wedge (b \vee (\neg c)) =$

$(a \wedge b \wedge c) \vee (a \wedge b \wedge (\neg c)) \vee (a \wedge (\neg b) \wedge (\neg c))$ *(DNF form)*

- $v(q_{dnf}) = (1,1,1)\ (1,1,0)\ (1,0,0)$
  - » ***Disjunctive Normal Form***
  - » ***Ex: (apple,computer,red) ∨ (apple, computer) ∨ (apple)***
- $v(q_{cc}) = (1,1,0)$
  - » **Conjunctive Component**

# Similarity/Matching function

$sim(q,d_j) = 1$ if $vec(d_j) = v\,(qcc)_i$

$v(qcc)_i \in v(qdnf)$

$0$ otherwise

*In other terms, only documents whose boolean vector is* ***one of the conjunctive components*** *of the query disjunctive normal form*

# Example

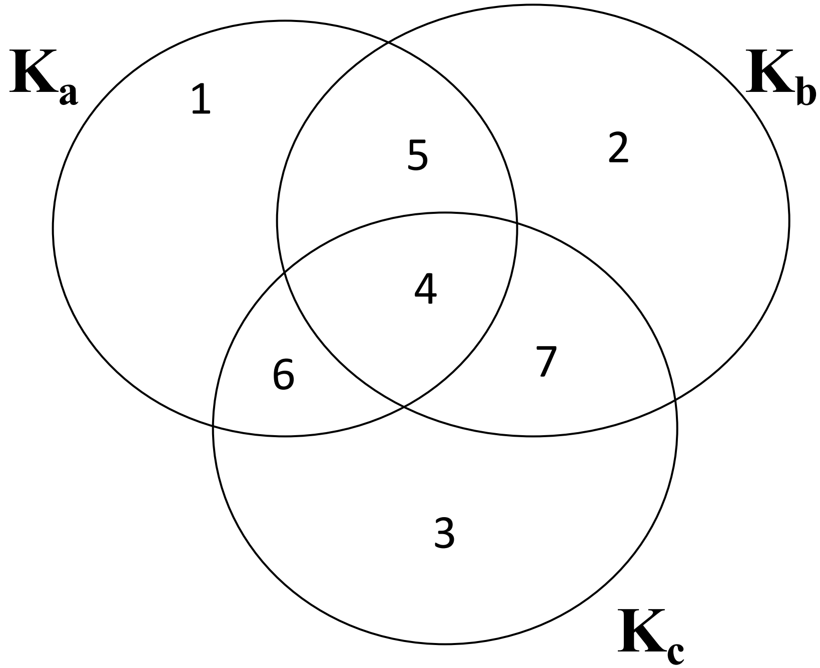
- $q = a \wedge (b \vee (\neg c)) =$

  $(a \wedge b \wedge c) \vee (a \wedge b \wedge (\neg c)) \vee (a \wedge (\neg b) \wedge (\neg c))$ *(DNF form)*

- $v(qdnf) = (1,1,1)\ (1,1,0)\ (1,0,0)$
  - » ***Disjunctive Normal Form***
  - » ***Ex: (apple,computer,red) ∨ (apple, computer) ∨ (apple)***
- $v(qcc) = (1,1,0)$
  - » **Conjunctive Component**

- Similar/Matching documents
  - $md_1$ = [apple apple blue day] => (1,0,0)
  - $md_2$ = [apple computer red ] => (1,1,1)
- Unmatched documents
  - $ud_1$ = [apple red] => (1,0,1)
  - $ud_2$ = [day] => (0,0,0)

# Venn Diagram ($K_i$ = generic keyword)

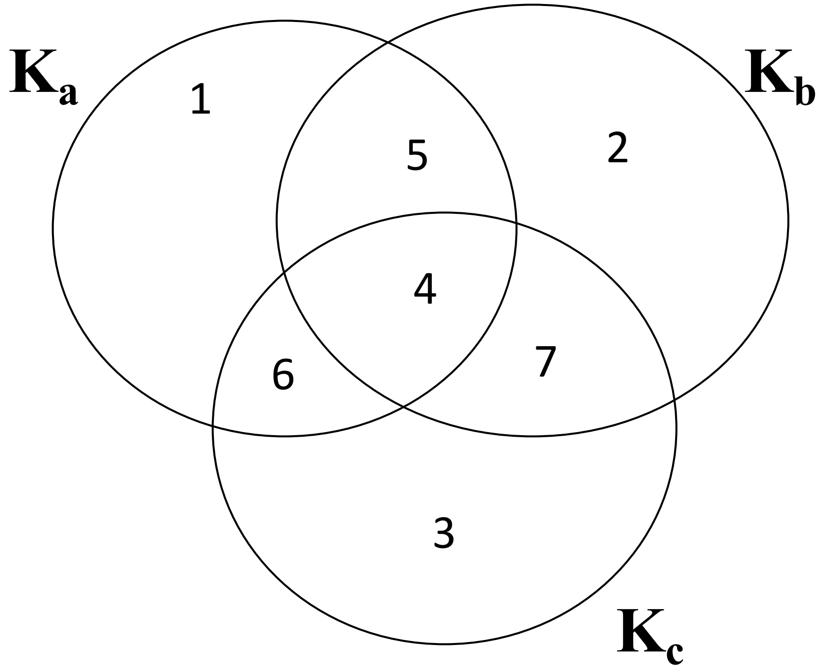



$$q = k_a \wedge (k_b \vee k_c)$$

$$(1 \wedge 1 \wedge 1) \vee (1 \wedge 1 \wedge 0) \vee (1 \wedge 0 \wedge 1)$$

Which one?

# Venn Diagram ($K_i$ = generic keyword)




$$q = k_a \wedge (k_b \vee \neg k_c)$$

$$(1 \wedge 1 \wedge 1) \vee (1 \wedge 1 \wedge 0) \vee (1 \wedge 0 \wedge 0)$$

Which one?

# Drawbacks of the Boolean Model

- ❑ Expressive power of boolean expressions to capture information needs and document semantics is *inadequate*
- ❑ Retrieval based on binary decision criteria (with no partial match) does not reflect our intuitions behind relevance adequately

- As a result
  - ❑ Answer set (results) contains either too few or too many documents in response to a user query
  - ❑ No ranking of documents

# Vector Model

# Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
  - Good for expert users with precise understanding of their needs and the collection (e.g., legal search / medical domain).
  - 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 wade through 1000s of results (e.g., web search).

# Problem 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 skill to come up with a query that produces a manageable number of hits.
- With **a ranked list of documents**, it does not matter how large the retrieved set is.
- User will looks at **first results.**

# Scoring as the basis of ranked retrieval

- We wish to return *in* ***order of relevance*** 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 pertinence score – say in [0, 1] – to each document
- This score measures how well document and query "match".

# Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let’s start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

# Vector Weighting Model

- **Model**: each document is a bag-of-words
- **Representation**:
  a **|V|-dimensional vector** (|V|, the dimension of the vocabulary)

- **Weighting scheme**: coordinate $w_{ij}$ of vector $d_j$ associated to document $d_j$ is the RELEVANCE of WORD $i$ in document $j$
- How do we measure $w_{ij}$ ?

# Bag of words vector

- Vector representation doesn't consider the ordering of words in a document
  - *d1: John is quicker than Mary* and
    *d2: Mary is quicker than John*
    have the same vectors, since we have a coordinate (or coefficient, or weight) $w_i$ for every word *i* of the vocabulary, and coordinates are ordered alphabetically
  - $d1=d2=(w_{John}, w_{is}, w_{Mary}, w_{quicker}, w_{than})$

- This is called (as we said in previous lessons) the bag of words model.
  - In a sense, this is a step back: the **positional index** (see lectures on indexing) was able to distinguish these two documents.

# Binary term-document matrix

documents

| words | 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 |

Any column j is a document vector $d_j$.
Each document is represented by a binary vector $\in \{0,1\}^{|V|}$, $w_{ij}$ is either 0 (word *i* is absent in $d_j$) or 1 (word *i* appears in $d_j$)
Number of rows = dimension of vocabulary **|V|**
Number of columns = dimension of the document collection **N**

# 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 $d$.
- We want to use *tf* when computing query-document match scores. But how?

# Term-document count matrix

- This scheme considers the number of occurrences of a term in a document:
  - Each document is a $\boxed{\text{count vector}}$ in $\mathbb{N}^v$

| | 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 |

# Comulative Model

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

- $Sim(q,d) = \sum_{t \in q \cap d} tf_{t,d}$

    - **The score is 0 if none of the query terms is present in the document, and grows when the document includes many of the query terms, with a high frequency**

- *Raw* term frequency is *not* what we want:
    - A document with 10 occurrences of the term may be more relevant than a document with one occurrence of the term.
    - But not 10 times more relevant.
- One possibility is to normalize: $tf_{i,j}^{norm} = tf_i / \sum_{j \in d_j}(tf_j)$

- **Relevance does not increase proportionally with term frequency**.

# Log-frequency weighting

- The log frequency weight of term t in d is

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

- 0 → 0,
- 1 → 1,
- 2 → 1.3,
- 10 → 2,
- 1000 → 4, etc.

# Scoring similarity

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

- Sim(q,d) $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
  - **The score is 0 if none of the query terms is present in the document, and grows when the document includes many of the query terms, with a high frequency**

- However, frequency-based ranking (whether normalized or log) IS NOT FULLY APPROPRIATE

- WHY??

# Inverse Document Frequency

- **Rare terms are more informative than frequent terms**
  - Recall stop words!
  - Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
  - A document containing this term is very likely to be relevant to the query *"study on **arachnocentric** people"*
  - → **We want a higher weight for rare terms as:**
    - ***arachnocentric***

# Inverse Document Frequency (1)

- 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 lower weights than for rare terms, since they do not characterize a single document
- We will use <u>document frequency</u> (df) to capture the intuition that terms appearing in many documents of the collection should have a lower weight
- df $\leq N$ = number of documents that contain the term, N= dimension of the document collection

# Inverse Document Frequency (2)

- $\mathrm{df}_t$ is the document frequency of $t$:
  the number of documents in the collection that contain $t$
  - df is a measure of the informativeness of $t$

- 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 (N = 1M )

| term | $df_t$ = # *of documents including the term* | $idf_t$ |
|---|---|---|
| calpurnia | 1 | 6 |
| animal | 100 | 4 |
| sunday | 1,000 | 3 |
| fly | 10,000 | 2 |
| under | 100,000 | 1 |
| the | 1,000,000 | 0 |

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

# Collection vs. Document frequency

- The **in-collection frequency** of *a word i* is the number of occurrences of *i* in the collection, **counting multiple occurrences**.

| Word | Collection frequency | Document frequency |
|---|---|---|
| *insurance* | 10440 | 3997 |
| *try* | 10422 | 8760 |

- $df_i$ measures the document, not the collection, frequency. +1 every times a document includes one ore more instances of word *i*.

# Scoring Similarity: tf-idf

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

$$\mathrm{w}_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log \; N / \mathrm{df}_t$$

- 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

# Binary → count → weight matrix

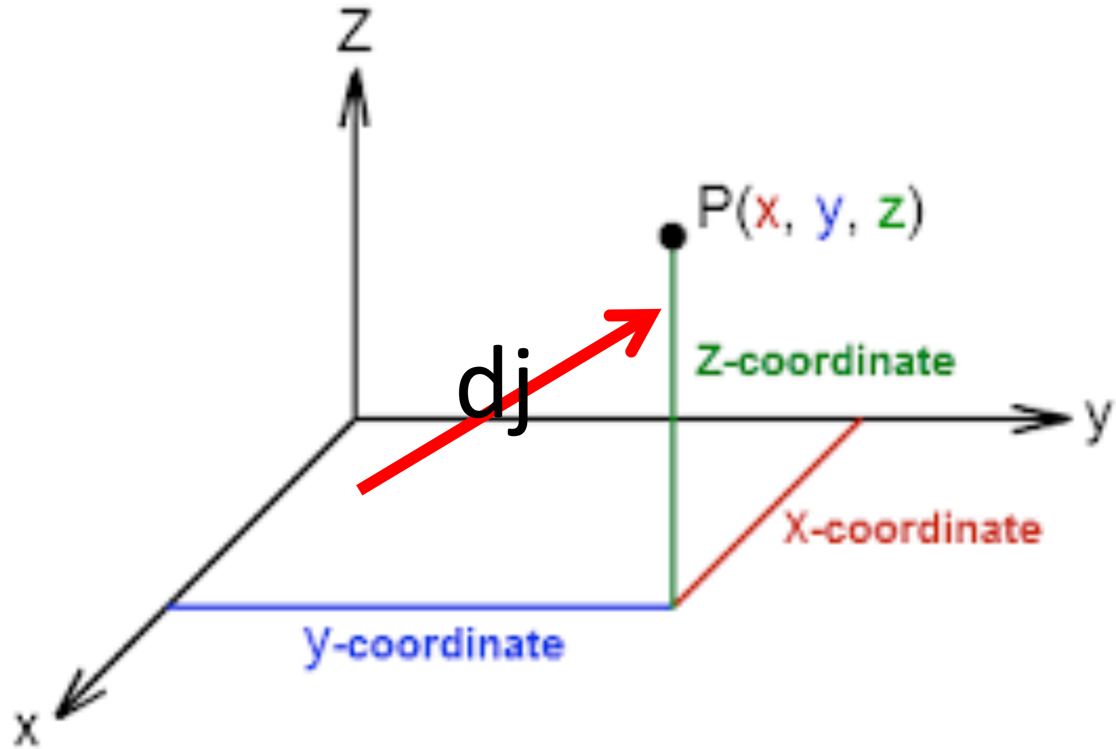
| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 5.25 | 3.18 | 0 | 0 | 0 | 0.35 |
| Brutus | 1.21 | 6.1 | 0 | 1 | 0 | 0 |
| Caesar | 8.59 | 2.54 | 0 | 1.51 | 0.25 | 0 |
| Calpurnia | 0 | 1.54 | 0 | 0 | 0 | 0 |
| Cleopatra | 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 $\in \mathbb{R}^{|V|}$**

# Documents as vectors

- So we have a |V|-dimensional vector space, one dimension for each term.
- Terms are axes of the space
- Documents are **points or vectors** in this space.
- The **coordinate** of a vector $d_j$ on dimension *i* is the tf-idf weight of word *i* in document *j*.
- **Very high-dimensional**: hundreds of thousand of dimensions when you apply this to a search engine
- It is a very sparse vector - **most entries are zero** (will see later in this course how to reduce dimensionality).

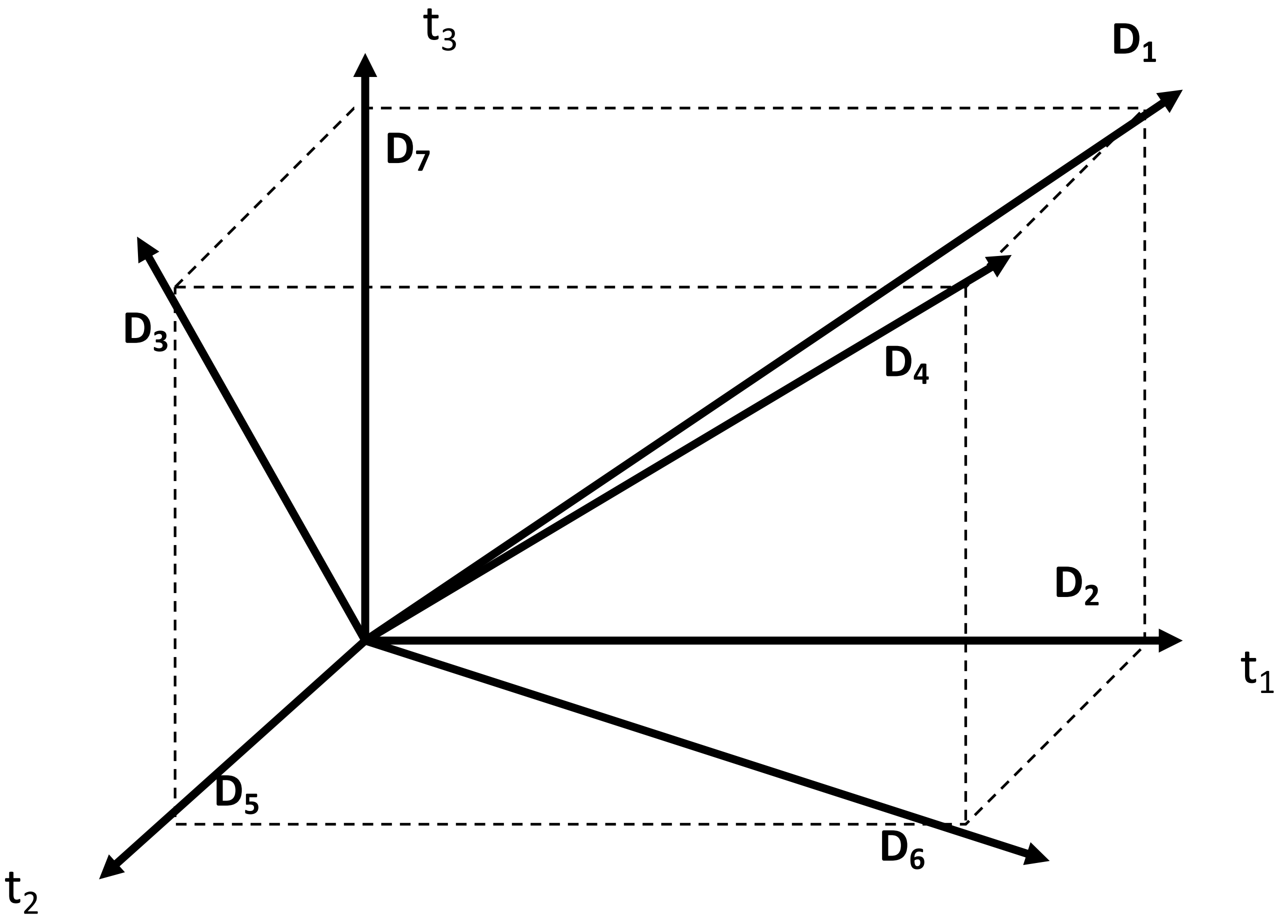
# Vector space model (for |V|=3)




X, Y, Z are the 3 dimensions associated to keywords $k_x$, $k_y$, $k_z$
x, y, z are the 3 weights of keywords $k_x$, $k_y$, $k_z$ in $d_j$

The coordinate of a vector $d_j$ on dimension *i* is the tf-idf weight of word *i* in document *j*.

# Documents in Vector Space



# Vector Space Scoring Model

- 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 ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model and the Cumulative one.
- **Rank more relevant documents higher than less relevant documents**

# Vector Space Proximity

- First cut: distance between two points
  ( = distance between the end points of the two vectors)

- Euclidean distance? $$d(d_j, q) = \sqrt{\sum_i (w_{ij} - w_{iq})^2}$$

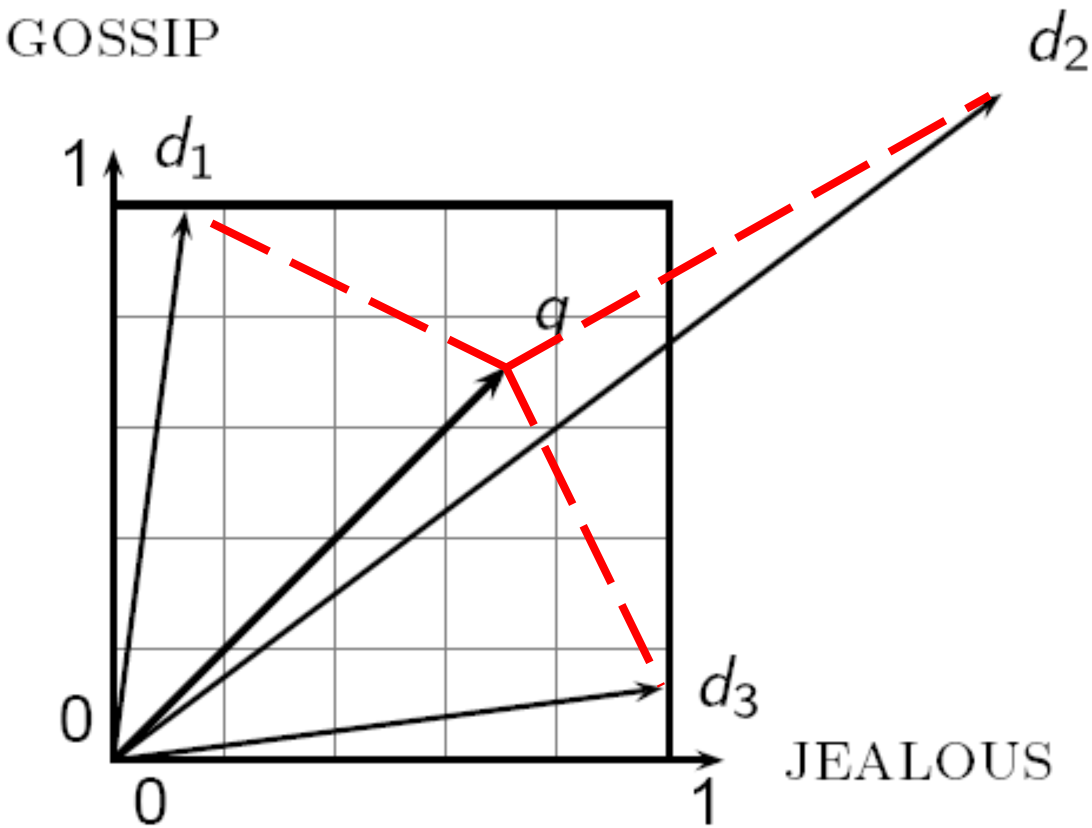
- Euclidean distance is a bad idea . . .

- . . . because Euclidean distance is large for vectors of different lengths (more terms).

# Why Euclidean distance is a bad idea?

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ (red dashed line) is large even though the distribution of terms in the query $\vec{q}$ and the **distribution** of terms in the document $\vec{d_2}$ are very similar (about 50% gossip, 50% Jealous).

Absolute frequencies cause the difference.



# Why Euclidean distance is a bad idea?

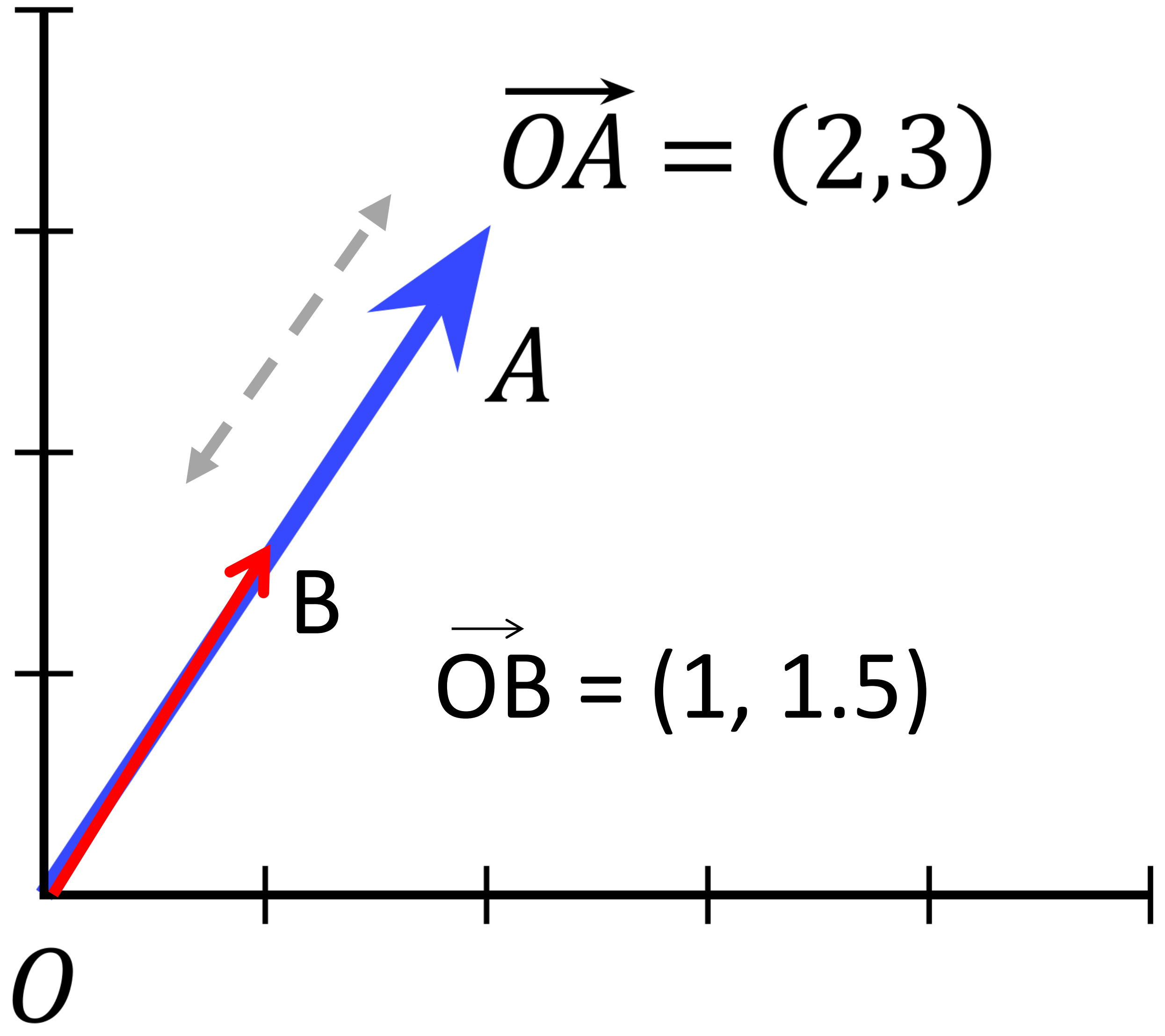
- Experiment:

  *take a document d and append it to itself. Call this document d'.*

- "Semantically" d and d' have the **exactly same content**

- The Euclidean distance between the two documents can be quite large (word frequency doubles in d');

# Example



# Length normalization

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

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

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

  - Effect on the two documents d and d′ (d appended to itself) from earlier slide: t
    - hey have **identical vectors** after length-normalization.

# Measure the Angle between Documents

- In previous example, the angle between the two documents is 0;
- Key idea: **Rank** documents according to **angle** with query;
- In previous example (where the angle is zero) corresponding to maximum similarity!
- In fact the two documents have the same words, with same relative weight.

# From angles to cosines

- The following two notions are equivalent.
  - Rank documents in <u>decreasing</u> order of the angle between query and document
  - Rank documents in <u>increasing</u> order of **cosine**(query,document)
- Cosine is a monotonically decreasing function in the interval [$0^o$, $180^o$]

# Vector Space Model: cosine-similarity

Dot product

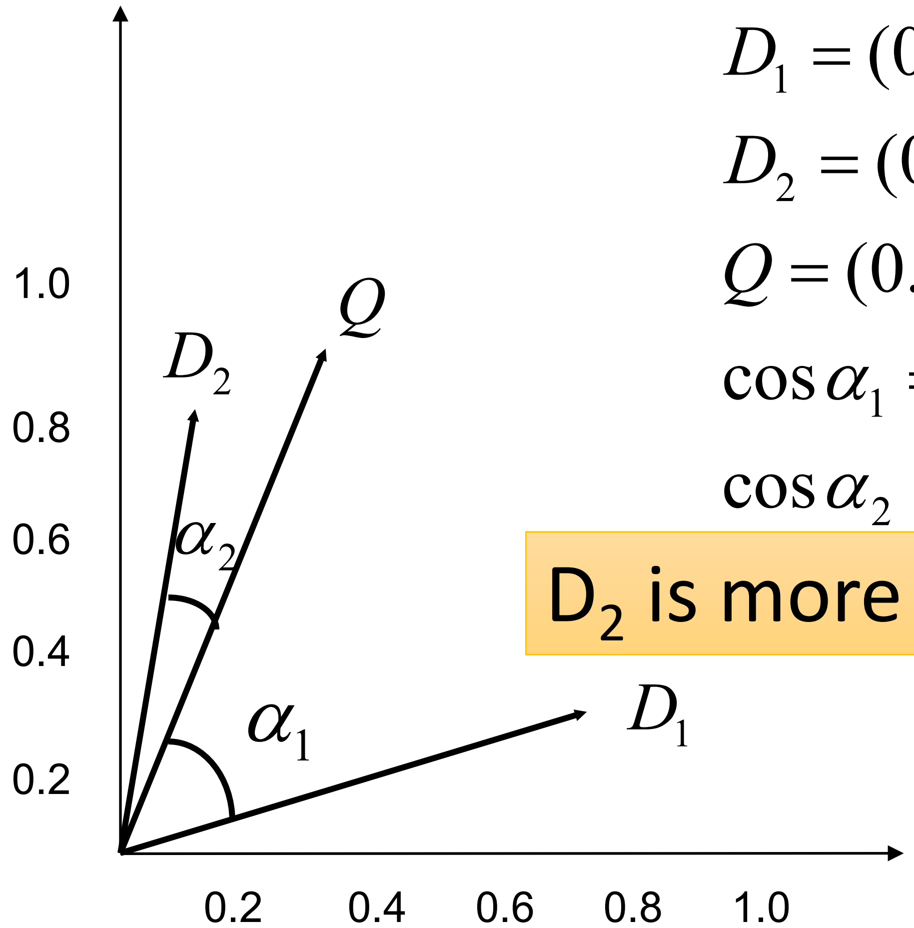
Unit vectors

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

$q_i$ 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($\vec{q},\vec{d}$) is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Cosine-similarity is the cosine of the angle between normalized query end document vectors.

# Example




$$D_1 = (0.8, 0.3)$$

$$D_2 = (0.2, 0.7)$$

$$Q = (0.4, 0.8)$$

$$\cos\alpha_1 = 0.74$$

$$\cos\alpha_2 = 0.98$$

$D_2$ is more similar to Q than $D_1$!!

# A complete example

A small collection of N=3 documents, |V|=6 words

$d_1$: " new york times"
$d_2$: " new york post"
$d_3$: " los angeles times"

Compute idf

| | |
|---|---|
| angeles | $\log_2(3/1) = 1.584$ |
| lod | $\log_2(3/1) = 1.584$ |
| new | $\log_2(3/2) = 0.584$ |
| post | $\log_2(3/1) = 1.584$ |
| times | $\log_2(3/2) = 0.584$ |
| york | $\log_2(3/2) = 0.584$ |

# A complete example

Document-term matrix (we use **normalized tf**, however here each word appears just once in each document)

| | angeles | los | new | post | times | york |
|---|---|---|---|---|---|---|
| d1 | 0 | 0 | 1 | 0 | 1 | 1 |
| d2 | 0 | 0 | 1 | 1 | 0 | 1 |
| d3 | 1 | 1 | 0 | 0 | 1 | 0 |

tf-idf: multiply tf by idf values

| | angeles | los | new | post | times | york |
|---|---|---|---|---|---|---|
| d1 | 0 | 0 | 0.584 | 0 | 0.584 | 0.584 |
| d2 | 0 | 0 | 0.584 | 1.584 | 0 | 0.584 |
| d3 | 1.584 | 1.584 | 0 | 0 | 0.584 | 0 |

# A complete example

Query: “new new times”

When computing the *tf-idf* values for the query terms we divide the frequency by the maximum frequency (2) to normalize, and multiply with the *idf* values

| | | | | | | |
|---|---|---|---|---|---|---|
| q | 0 | 0 | (2/2)*0.584=0.584 | 0 | (1/2)*0.584=0.292 | 0 |

We calculate the length (the NORM) of each document vector and of the query:

Length of d1 = sqrt(0.584^2+0.584^2+0.584^2)=1.011
Length of d2 = sqrt(0.584^2+1.584^2+0.584^2)=1.786
Length of d3 = sqrt(1.584^2+1.584^2+0.584^2)=2.316
Length of q = sqrt(0.584^2+0.292^2)=0.652

# A complete example

Similarity values are computed using cosin-sim formula:

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

cosSim(d1,q) = (0*0+0*0+0.584*0.584+0*0+0.584*0.292+0.584*0) / (1.011*0.652) = 0.776
cosSim(d2,q) = (0*0+0*0+0.584*0.584+1.584*0+0*0.292+0.584*0) / (1.786*0.652) = 0.292
cosSim(d3,q) = (1.584*0+1.584*0+0*0.584+0*0+0.584*0.292+0*0) / (2.316*0.652) = 0.112

According to the computed similarity values, the final order in which the documents are presented as result to the query will be: d1, d2, d3.

# Cosine Similarity

How similar are the novels:

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

Cosine similarity amongst 3 documents

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

Term frequencies (counts)

# 3 documents example contd

**Log Term 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 |

**Tf-idf and normalize**

| 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

# Similarity and Distances

All **data mining problems**, such as clustering, outlier detection, and classification, **require** the **computation** of **similarity**.

*A methodical way of quantifying similarity between data objects is required.*

A formal statement is:

Given **two objects** $O_1$ and $O_2$, determine a value of the similarity Sim($O_1$, $O_2$) (or distance Dist($O_1$, $O_2$) between the two objects.

In **similarity** functions, **larger** values imply greater **similarity**, whereas in **distance** functions, **smaller** values imply **greater** similarity.

*In some domains, such as **spatial data**, it is more **natural** to talk about **distance** functions, whereas in other **domains**, such as text, it is more **natural** to talk about **similarity** functions.*

# Symiliarty - Notes

- Nevertheless, the **principles** involved in the design of such functions are generally **invariant** across **different** data **domains**.
- Similarity and distance **functions** are often **expressed** in **closed** form (e.g., Euclidean distance), **but** in **some domains**, such as time-series data, they are **defined algorithmically**.
- Distance functions are **fundamental** to the **effective** design of data mining algorithms, because a **poor choice** in this respect may be very **detrimental** to the **quality** of the results.
- Distance **functions** are highly **sensitive** to the data **distribution**, **dimensionality**, and data **type**.
- In **some** data **types**, such as multidimensional data, it is much **simpler** to **define** and **compute** distance functions.

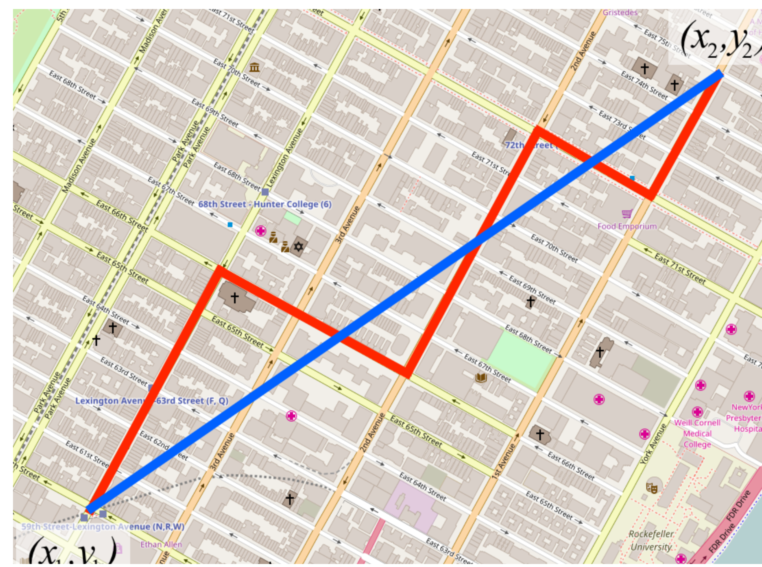
# Quantitative Data ($L_p$-norm)

The most **common distance** function for quantitative data is the $L_p$-norm. The $\boldsymbol{L_p}$**-norm** between two data points $\bar{X} = (x_1 \cdots x_d)$ and $\bar{Y}$ = $(y_1 \cdots y_d)$ is **defined** as follows:

$$Dist(\bar{X}, \bar{Y}) = \left( \sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$$

Special cases of the $L_p$-norm are the:
*Euclidean* $(p = 2)$
and the
*Manhattan* $(p = 1)$
metrics: $\sum_{i=1}^{d} |x_i - y_i|$

*Infinity* where p = ∞ : $max_{i=1}^{d} |x_i - y_i|$



# Quantitative Data ($L_2$-norm)

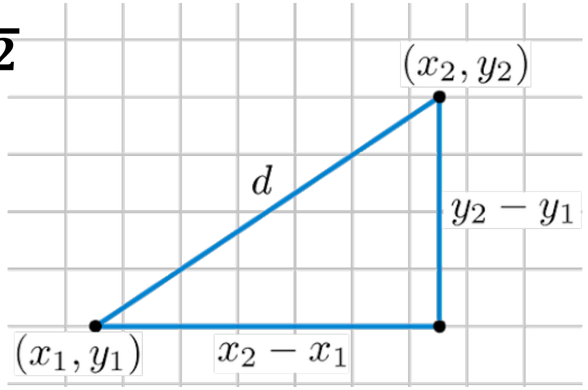
The $\boldsymbol{L_p}$**-norm** between two data points $\bar{X} = (x_1 \cdots x_d)$ and $\bar{Y}$ = $(y_1 \cdots y_d)$ is **defined** as follows:

$$Dist(\bar{X}, \bar{Y}) = \left( \sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$$

*Euclidean* $(p = 2)$ is a special cases of the $L_p$-norm

$$Dist(\bar{X}, \bar{Y}) = \left( \sum_{i=1}^{d} |x_i - y_i|^2 \right)^{1/2} = \sqrt[2]{\sum_{i=1}^{d} |x_i - y_i|^2} =$$

$$= \sqrt[2]{|x_1 - y_1|^2 + |x_2 - y_2|^2 + \cdots\cdots + |x_d - y_d|^2}$$



# $L_p$-norm

- The *Euclidean* distance is the **straight-line** distance between two data points.
- The *Manhattan* distance is the **"city block"** driving distance in a region in which the streets are arranged as a **rectangular grid**, such as the Manhattan Island of New York City.

- A nice property of the **Euclidean** distance is that it is **rotation-invariant** because the straight-line distance between two data points **does not change** with the orientation of the **axis** system.

- This property also means that **transformations**, such as ***PCA**, SVD*, or the wavelet transformation for time series **can be used** on the data **without affecting** the **distance**.

- These distance functions may **not work** very **well** when the data are **high dimensional** because of the varying impact of data **sparsity**, **distribution**, **noise**, and **feature** relevance

# Minkowski Distance

In some **cases**, an analyst may **know** which **features** are **more important** than others for a particular application.
E.g., for a **credit-scoring application**, an attribute such as **salary** is much **more relevant** to the design of the distance function **than** an attribute such as **gender**, though both may have some impact.

The **generalized** $L_p$-distance is most suitable for this case and is defined in a similar way to the $L_p$-norm, except that a **coefficient** $\boldsymbol{a_i}$ is associated with the $i^{th}$ feature. This coefficient is **used to weight** the corresponding **feature** component in the $L_p$-norm:

$$Dist(\bar{X}, \bar{Y}) = \left(\sum_{i=1}^{d} a_i \cdot |x_i - y_i|^p\right)^{1/p}$$

This distance is **also referred** to as the generalized **Minkowski distance**. In many cases, such **domain knowledge** is **not available**.

# Impact of High Dimensionality (1)

Many distance-based data mining applications lose their effectiveness as the **dimensionality** of the data **increases**. E.g., a **distance-based clustering** algorithm may group unrelated data points because the distance function may **poorly reflect** the **intrinsic semantic** distances between **data points** with increasing dimensionality.

*This phenomenon is referred to as the "curse of dimensionality," a term first coined by Richard Bellman.*

# Impact of High Dimensionality (2)

Consider the **unit cube** of dimensionality ***d*** that is fully located in the **nonnegative** quadrant, with one **corner** at the origin $\bar{\boldsymbol{O}}$.

*What is the **Manhattan distance** of the corner of this cube (say, at the origin) to a **randomly chosen** point $\bar{X}$ **inside** the **cube**?*

In this case, because **one end** point is the **origin**, and all coordinates are nonnegative, the Manhattan distance **will sum** up the **coordinates** of $\bar{X}$ over the **different dimensions**.

**Each** of these **coordinates** is uniformly **distributed** in $[0, 1]$. Therefore, if $Y_i$ represents the uniformly distributed **random variable** in $[0, 1]$, it follows that the **Manhattan distance** is as follows:

$$Dist(\bar{O}, \bar{X}) = \sum_{i=1}^{d} |Y_i - 0|$$

# Impact of High Dimensionality (3)

The result is a **random variable** with a **mean** of $\mu = \mathrm{d}\,{}^{1}/_{2}$ and a **standard deviation** of $\sigma = \sqrt{{}^{d}/_{12}}$ .
For **large values** of $d$, it can be shown by the law of large numbers that the vast **majority** of **randomly** chosen **points** inside the cube will lie in the **range** $[D_{min}, D_{max}] = [\mu - 3\sigma, \mu + 3\sigma]$.
Therefore, most of the points in the cube lie within a distance range of $D_{max} - D_{min} = 6\sigma = \sqrt{3d}$ from the origin. Note that the expected ***Manhattan* distance grows** with dimensionality at a rate that is **linearly** proportional **to *d***.
The **ratio** of the **variation** in the distances to the absolute values that is **referred** to as **Contrast**($\mathcal{D}$), is given by:

$$\text{Contrast (D)} = \frac{D_{max} - D_{min}}{\mu} = \sqrt{{}^{12}/_{d}}$$

# Impact of High Dimensionality (4)

This **ratio** can be interpreted as the **distance contrast** between the **different** data **points**, in terms of how different the **minimum** and **maximum** distances from the **origin** might be considered.

The **contrast reduces** with $\sqrt{d}$ , it means that there is **virtually no contrast** with increasing dimensionality.

***Lower contrasts** means that the data mining **algorithm** will **score** the **distances** between all pairs of data points in **approximately** the **same**.*