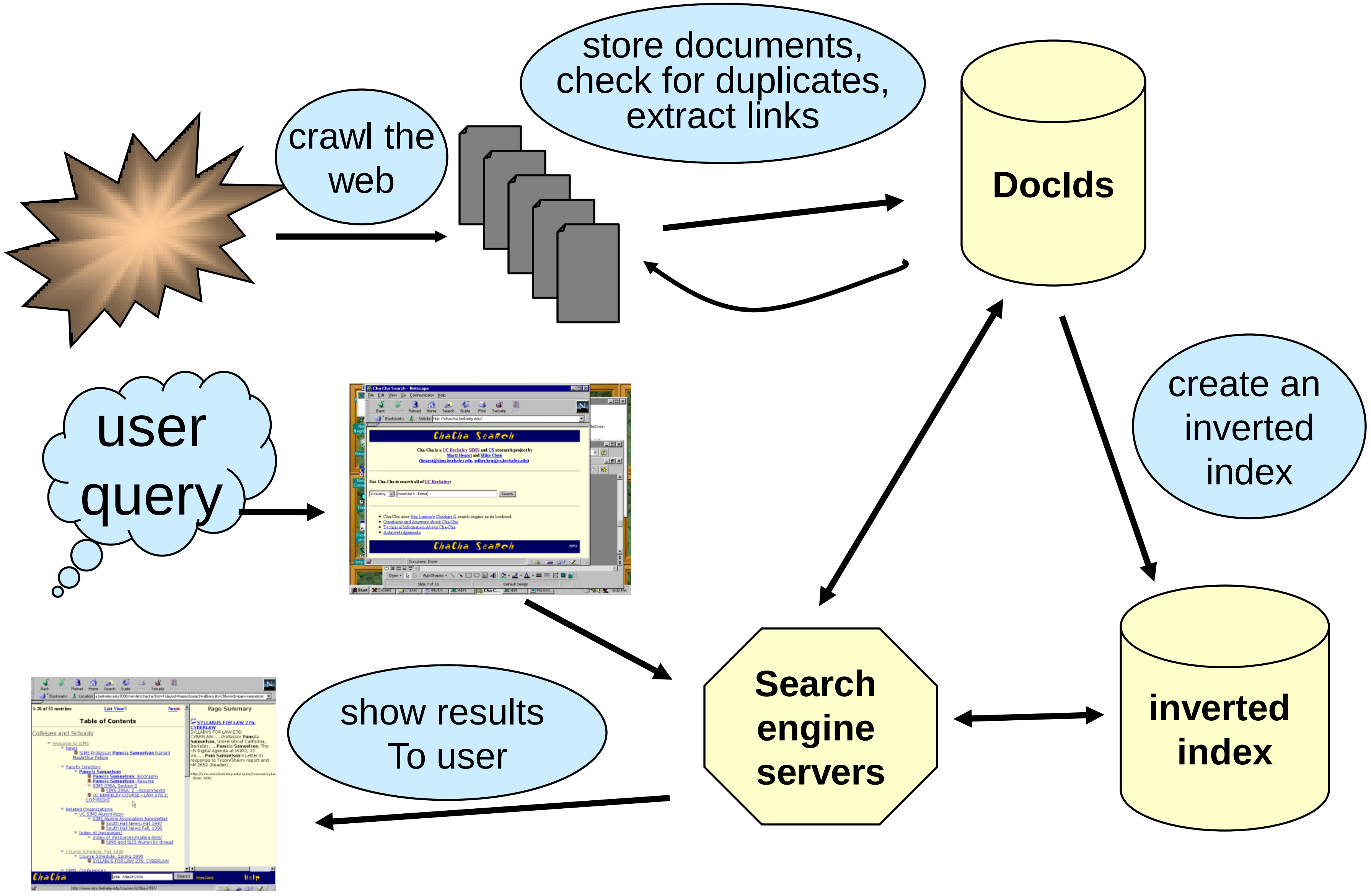
Document Similarity in Information Retrieval

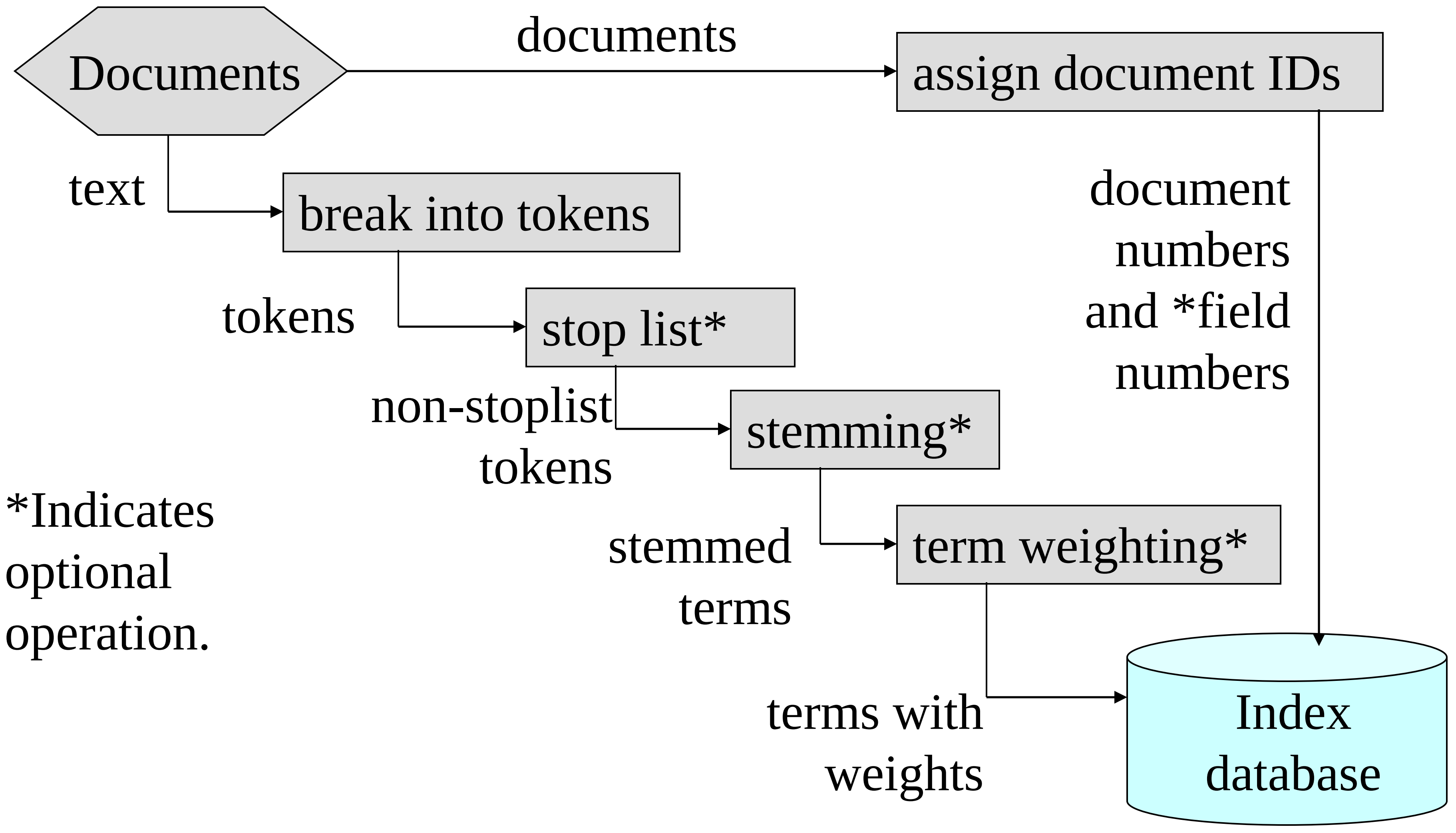
Mausam

(Based on slides of W. Arms, Thomas Hofmann, Ata Kaban, Melanie Martin)

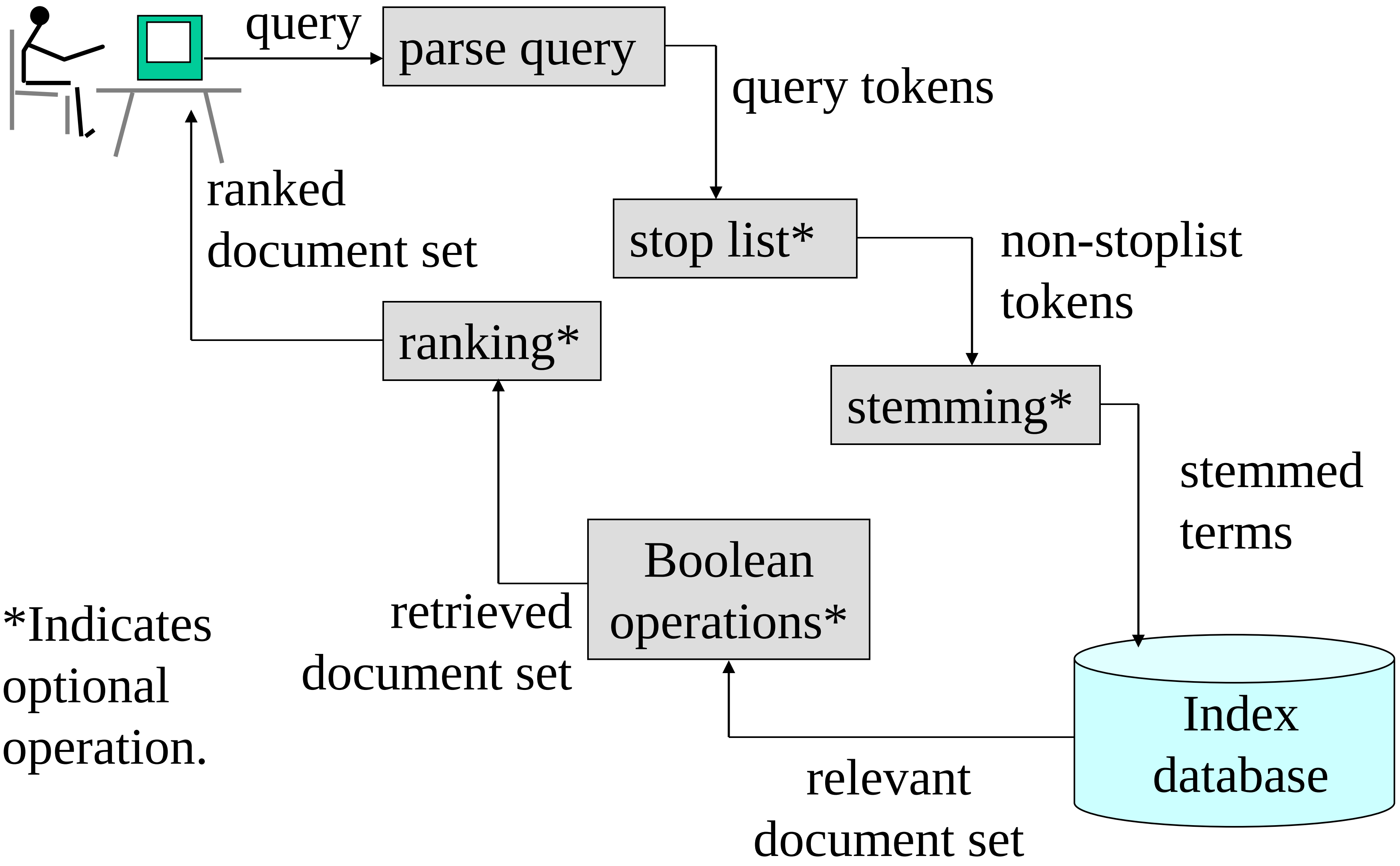
Standard Web Search Engine Architecture



Indexing Subsystem



Search Subsystem



*Indicates optional operation.

Terms vs tokens

- Terms are what results after tokenization and linguistic processing.
  - Examples
    - knowledge -> knowledg
    - The -> the
    - Removal of stop words

Matching/Ranking of Textual Documents

Major Categories of Methods

1. **Exact matching** (Boolean)
2. Ranking by **similarity to query** (vector space model)
3. Ranking of matches by **importance of documents** (PageRank)
4. Combination methods

   What happens in major search engines (Googlerank)

Vector representation of documents and queries

Why do this?

- Represents a large space for documents
- Compare
  - Documents
  - Documents with queries
- Retrieve and rank documents with regards to a specific query

  - *Enables methods of similarity*

All search engines do this.

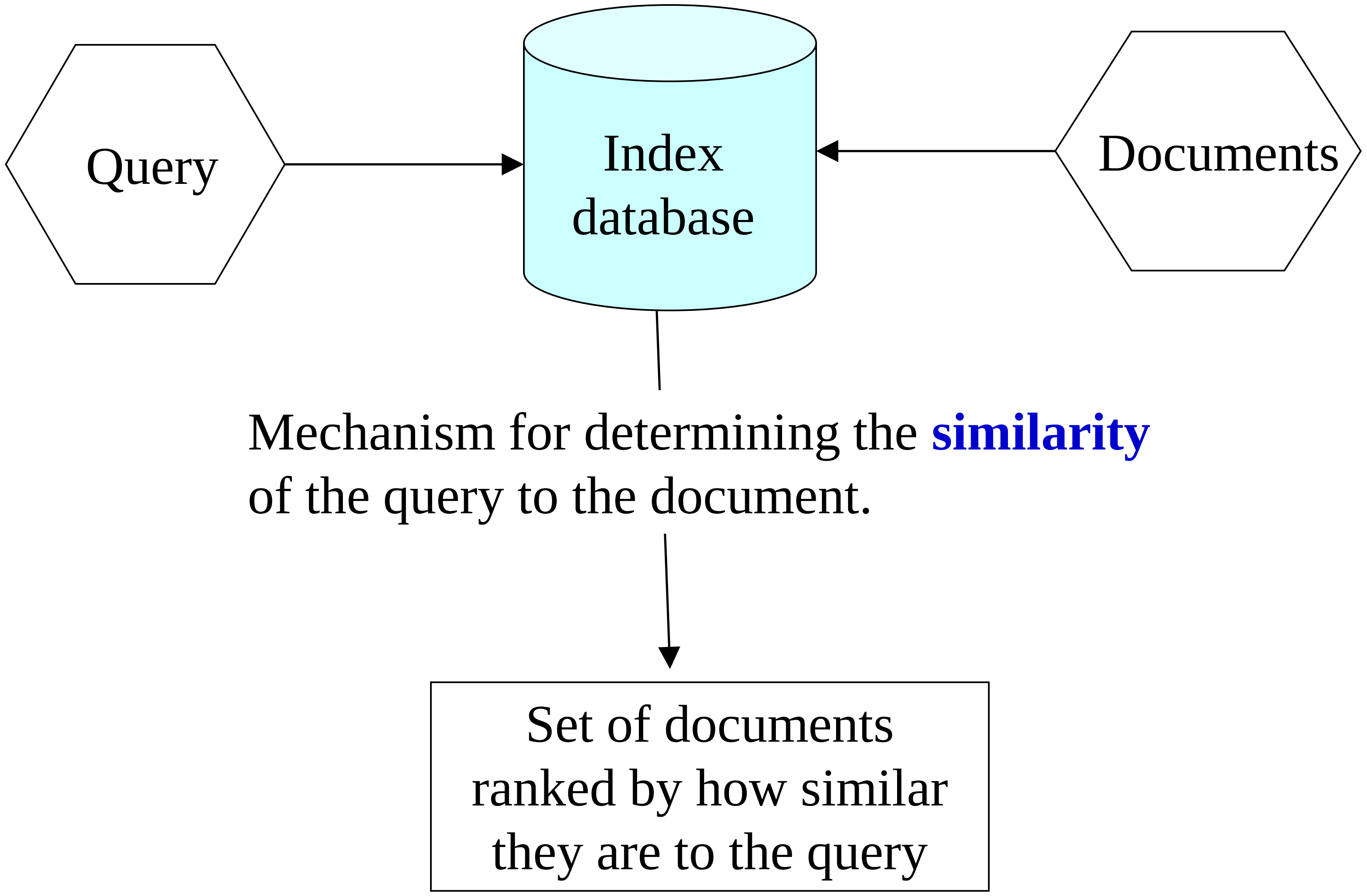
Boolean queries

- Document is relevant to a query of the *query itself* is in the document.
  - Query blue and red brings back all documents with blue and red in them
- Document is either relevant or not relevant to the query.
- What about relevance ranking – partial relevance. Vector model deals with this.

Similarity Measures and Relevance

- Retrieve the most similar documents to a query
- Equate similarity to relevance
  - Most similar are the most relevant
- This measure is one of “text similarity”
  - The matching of text or words

Similarity Ranking Methods


Query
Index database
Documents
Set of documents ranked by how similar they are to the query

Mechanism for determining the **similarity** of the query to the document.

Term Similarity: Example

Problem: Given two text documents, how **similar** are they?

[Methods that measure similarity do not assume exact matches.]

Example (assume tokens converted to terms)

Here are three documents. How similar are they?

d_1	*ant ant bee*
d_2	*dog bee dog hog dog ant dog*
d_3	*cat gnu dog eel fox*

Documents can be any length from one word to thousands.

A query is a special type of document.

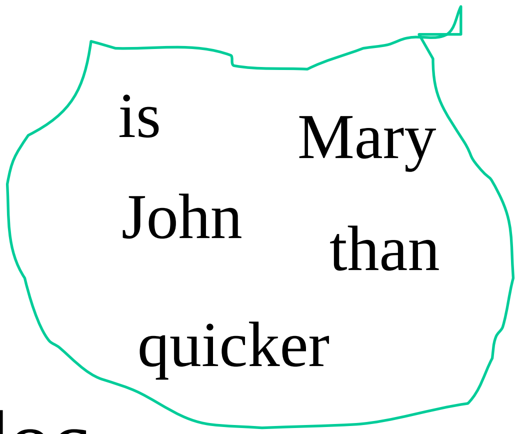
Bag of words view of a doc

Tokens are extracted from text and thrown into a “bag” without order and labeled by document.

- Thus the doc
  - ***John is quicker than Mary***.

is indistinguishable from the doc

  - ***Mary is quicker than John***.



Term Similarity: Basic Concept

Two documents are similar if they contain some of the same terms.

Possible measures of similarity might take into consideration:

(a) The lengths of the documents

(b) The number of terms in common

(c) Whether the terms are common or unusual

(d) How many times each term appears

TERM VECTOR SPACE

Term vector space

n-dimensional space, where *n* is the number of different terms/tokens used to index a set of documents.

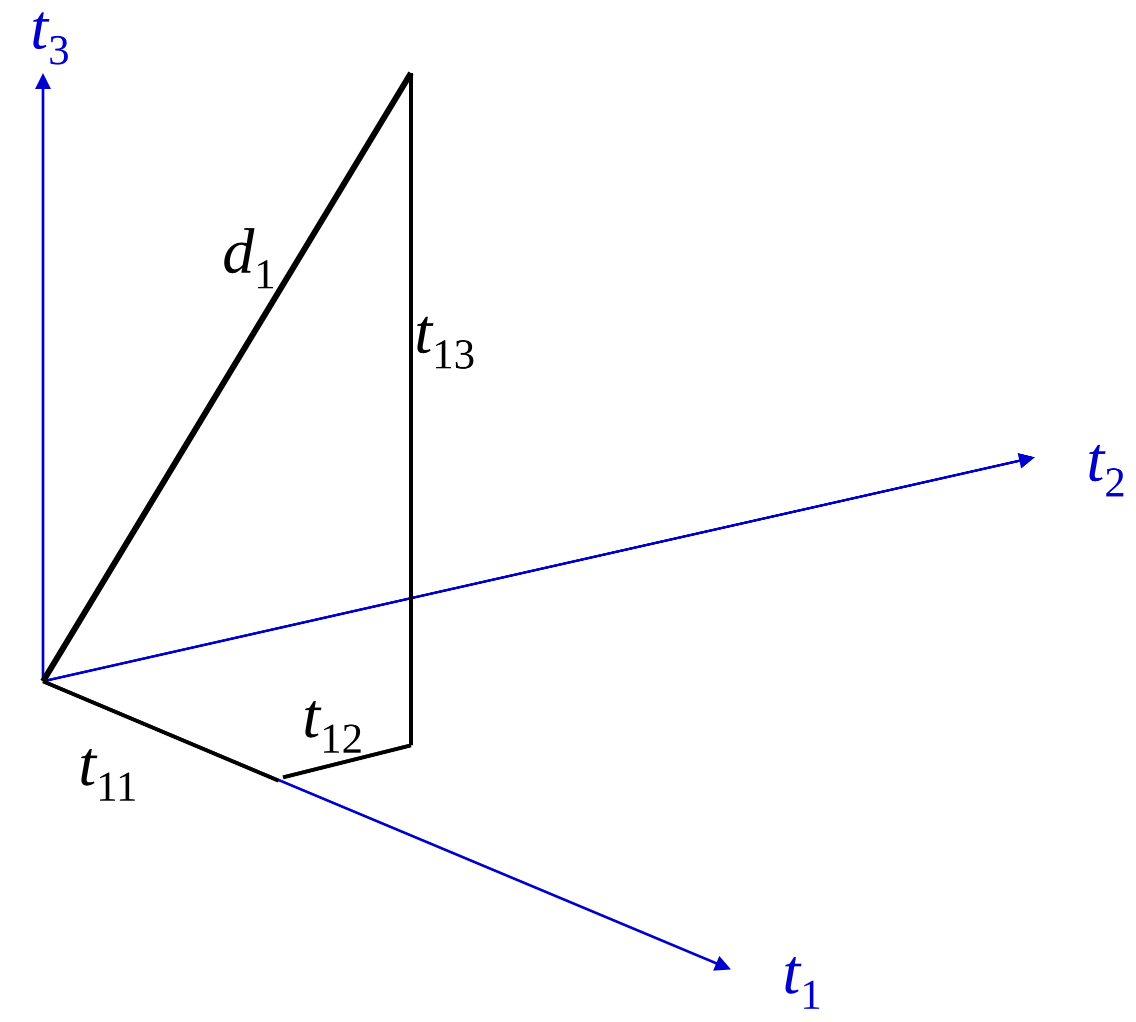
Vector

Document i, d_i, represented by a vector. Its magnitude in dimension j is w_{ij}, where:

$w_{ij} > 0$	if term j occurs in document i
$w_{ij} = 0$	otherwise

w_{ij} is the **weight** of term j in document i.

A Document Represented in a 3-Dimensional Term Vector Space



Basic Method: Incidence Matrix (Binary Weighting)

document	text	terms
d_1	*ant ant bee*	*ant bee*
d_2	*dog bee dog hog dog ant dog*	*ant bee dog hog*
d_3	*cat gnu dog eel fox*	*cat dog eel fox gnu*

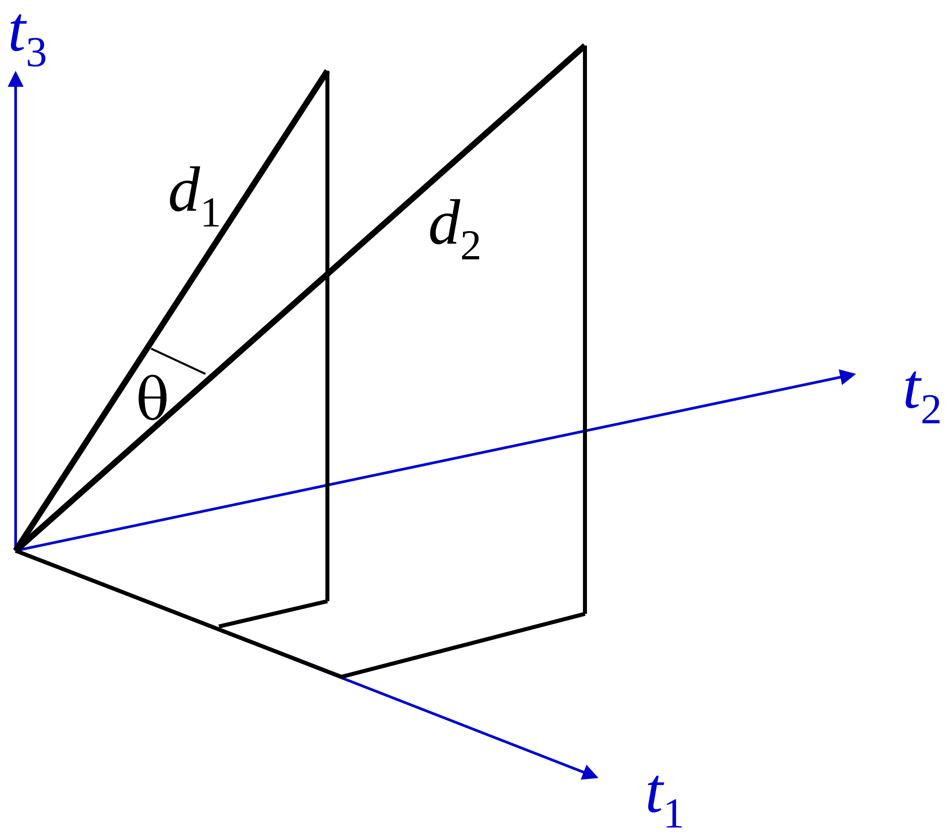
	ant	bee	cat	dog	eel	fox	gnu	hog
d_1	1	1						
d_2	1	1		1				1
d_3			1	1	1	1	1	

3 vectors in 8-dimensional term vector space

Weights: t_{ij} = 1 if document *i* contains term *j* and zero otherwise

Basic Vector Space Methods: Similarity between 2 documents

The **similarity** between two documents is a function of the **angle** between their **vectors** in the term vector space.



Vector Space Revision

$\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$ is a vector in an n-dimensional vector space

Length of $\mathbf{x}$ is given by (extension of Pythagoras's theorem)

$$|\mathbf{x}|^2 = x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2$$

$$|\mathbf{x}| = (x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2)^{1/2}$$

If $\mathbf{x}_1$ and $\mathbf{x}_2$ are vectors:

Inner product (or dot product) is given by

$$\mathbf{x}_1.\mathbf{x}_2 = x_{11}x_{21} + x_{12}x_{22} + x_{13}x_{23} + \ldots + x_{1n}x_{2n}$$

Cosine of the angle between the vectors $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\cos(\theta) = \frac{\mathbf{x}_1.\mathbf{x}_2}{|\mathbf{x}_1|\,|\mathbf{x}_2|}$$

Document similarity

$\mathbf{d} = (x_1, x_2, x_3, ..., x_n)$ is a vector in an *n*-dimensional vector space

<u>Length</u> of **x** is given by (extension of Pythagoras's theorem)

$$|\mathbf{d}|^2 = x_1{}^2 + x_2{}^2 + x_3{}^2 + ... + x_n{}^2$$

$$|\mathbf{d}| = (x_1{}^2 + x_2{}^2 + x_3{}^2 + ... + x_n{}^2)^{1/2}$$

If $\mathbf{d}_1$ and $\mathbf{d}_2$ are document vectors:

<u>Inner product</u> (or dot product) is given by

$$\mathbf{d}_1.\mathbf{d}_2 = x_{11}x_{21} + x_{12}x_{22} + x_{13}x_{23} + ... + x_{1n}x_{2n}$$

<u>Cosine angle</u> between the docs $\mathbf{d}_1$ and $\mathbf{d}_2$ determines doc similarity

$$\cos(\theta) = \frac{\mathbf{d}_1.\mathbf{d}_2}{|\mathbf{d}_1|\ |\mathbf{d}_2|}$$

$\cos(\theta) = 1$; documents exactly the same; $= 0$, totally different

Example 1
No Weighting

	ant	bee	cat	dog	eel	fox	gnu	hog	*length*
d_1	1	1							√2
d_2	1	1		1				1	√4
d_3			1	1	1	1	1		√5

Ex: $\textit{length}\ d_1 = (1^2+1^2)^{1/2}$

Example 1 (continued)

Similarity of documents in example:

	d_1	d_2	d_3
d_1	1	0.71	0
d_2	0.71	1	0.22
d_3	0	0.22	1

Digression: terminology

- WARNING: In a lot of IR literature, "frequency" is used to mean "count"
  - Thus *term frequency* in IR literature is used to mean *number of occurrences* in a doc
  - Not divided by document length (which would actually make it a frequency)
- We will conform to this misnomer
  - In saying term frequency we mean the number of occurrences of a term in a document.

Example 2
Weighting by Term Frequency (*tf*)

document	text	terms
d_1	*ant ant bee*	*ant bee*
d_2	*dog bee dog hog dog ant dog*	*ant bee dog hog*
d_3	*cat gnu dog eel fox*	*cat dog eel fox gnu*

	ant	bee	cat	dog	eel	fox	gnu	hog	*length*
d_1	2	1							$\sqrt{5}$
d_2	1	1		4				1	$\sqrt{19}$
d_3			1	1	1	1	1		$\sqrt{5}$

Weights: t_{ij} = frequency that term j occurs in document i

Example 2 (continued)

Similarity of documents in example:

	d_1	d_2	d_3
d_1	1	0.31	0
d_2	0.31	1	0.41
d_3	0	0.41	1

Similarity depends upon the weights given to the terms.

[Note differences in results from Example 1.]

Summary: Vector Similarity Computation with Weights

Documents in a collection are assigned **terms** from a set of *n* terms

The **term vector space** W is defined as:

if term *k* does not occur in document d_i, $w_{ik} = 0$

if term *k* occurs in document d_i, w_{ik} is greater than zero
(${w_{ik}}$ is called the **weight** of term *k* in document d_i)

Similarity between d_i and d_j is defined as:

$$\cos(\mathbf{d}_i, \mathbf{d}_j) = \frac{\sum_{k=1}^{n} w_{ik} w_{jk}}{|\mathbf{d}_i|\ |\mathbf{d}_j|}$$

Where $\mathbf{d}_i$ and $\mathbf{d}_j$ are the corresponding weighted term vectors and $|\mathbf{d}_i|$ is the length of the document vector $\mathbf{d}_i$

Summary: Vector Similarity Computation with Weights

Inner product (or dot product) between documents

$$\mathbf{d}_1.\mathbf{d}_2 = w_{11}w_{21} + w_{12}w_{22} + w_{13}w_{23} + \ldots + w_{1n}w_{2n}$$

Inner product (or dot product) is between a document and query

$$\mathbf{d}_1.\mathbf{q}_1 = w_{11}w_{q11} + w_{12}w_{q12} + w_{13}w_{q13} + \ldots + w_{1n}w_{q1n}$$

where $w_{q\mathrm{ij}}$ is the weight of the $_{\mathrm{j}}$th term of the $_{\mathrm{i}}$th query

Simple Uses of Vector Similarity in Information Retrieval

Threshold

For query *q*, retrieve all documents with similarity above a threshold, e.g., similarity > 0.50.

Ranking

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

[This is the standard practice.]

Simple Example of Ranking (Weighting by Term Frequency)

query		
q	*ant dog*	
document	**text**	**terms**
d_1	*ant ant bee*	*ant bee*
d_2	*dog bee dog hog dog ant dog*	*ant bee dog hog*
d_3	*cat gnu dog eel fox*	*cat dog eel fox gnu*

	ant	bee	cat	dog	eel	fox	gnu	hog	*length*
q	1			1					$\sqrt{2}$
d_1	2	1							$\sqrt{5}$
d_2	1	1		4				1	$\sqrt{19}$
d_3			1	1	1	1	1		$\sqrt{5}$

Calculate Ranking

Similarity of query to documents in example:

	d_1	d_2	d_3
q	$2/\sqrt{10}$ 0.63	$5/\sqrt{38}$ 0.81	$1/\sqrt{10}$ 0.32

If the query *q* is searched against this document set, the ranked results are:

d_2, d_1, d_3

Bigger Corpora

- Consider
  - n = 1M documents,
  - each with about 1K terms.

- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data.
- Say there are m = 500K *distinct* terms….

Can’t Build the Matrix

- 500K x 1M matrix: 500 Billion 0’s and 1’s.
- But it has no more than 1 billion 1’s.
  - matrix is extremely sparse.
- What’s a better representation?

Why?

Inverted index

Documents are parsed to extract words and these are saved with the document ID.

Doc 1	Doc 2
I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.	So let it be with Caesar. The Noble Brutus hath told you Caesar was ambitious

Term	Doc
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2

Later, sort inverted file by terms

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1

$\rightarrow$

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1

- Multiple term entries in a single document are merged and frequency information added

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

Best Choice of Weights?

query		
q	*ant dog*	
document	**text**	**terms**
d_1	*ant ant bee*	*ant bee*
d_2	*dog bee dog hog dog ant dog*	*ant bee dog hog*
d_3	*cat gnu dog eel fox*	*cat dog eel fox gnu*

	ant	bee	cat	dog	eel	fox	gnu	hog
q	?			?				
d_1	?	?						
d_2	?	?		?				?
d_3			?	?	?	?	?	

What weights lead to the best information retrieval?

Weighting
Term Frequency (tf)

Suppose term *j* appears f_{ij} times in document *i*. What weighting should be given to a term *j*?

Term Frequency: Concept

A term that appears many times within a document is likely to be more important than a term that appears only once.

Term Frequency: Free-text Document

Length of document$_i$

Simple method is to use w_{ij} as the term frequency.

...*but*, <u>in free-text documents</u>, terms are likely to appear more often in long documents. Therefore w_{ij} should be scaled by some variable related to document length.

Term Frequency: Free-text Document

A standard method for free-text documents

Scale f_{ij} relative to the frequency of other terms in the document$_i$. This partially corrects for variations in the length of the documents.

Let $m_i = \max (f_{ij})$ i.e., m_i is the maximum frequency of any term in document *i*.

Term frequency (tf):

$$tf_{ij} = f_{ij} / m_i \qquad \text{when } f_{ij} > 0$$

Note: There is no special justification for taking this form of term frequency except that it works well in practice and is easy to calculate.

Weighting
Inverse Document Frequency (idf)

Suppose term *j* appears f_{ij} times in document *i*. What weighting should be given to a term *j*?

Inverse Document Frequency: Concept

A term that occurs in a few documents is likely to be a better discriminator than a term that appears in most or all documents.

Inverse Document Frequency

Suppose there are n documents and that the number of documents in which term j occurs is n_j.

A possible method might be to use n/n_j *as the inverse document frequency.*

A standard method

The simple method over-emphasizes small differences. Therefore use a logarithm.

Inverse document frequency (idf):

$$idf_j = \log_2 (n/n_j) + 1 \qquad\qquad n_j > 0$$

Note: There is no special justification for taking this form of inverse document frequency except that it works well in practice and is easy to calculate.

Example of Inverse Document Frequency

Example

n = 1,000 documents; n_j # of docs term appears in

term j	n_j	idf_j
A	100	4.32
B	500	2.00
C	900	1.13
D	1,000	1.00

idf_j modifies only the columns not the rows!

From: Salton and McGill

Full Weighting:
A Standard Form of tf.idf

Practical experience has demonstrated that weights of the following form perform well in a wide variety of circumstances:

(weight of term *j* in document *i*)

= (term frequency) * (inverse document frequency)

A standard tf.idf weighting scheme, <u>for free text documents</u>, is:

$$t_{ij} = tf_{ij} * idf_j$$

$$= (f_{ij} / m_i) * (\log_2 (n/n_j) + 1) \quad \text{when } n_j > 0$$

Discussion of Similarity

The choice of similarity measure is widely used and works well on a wide range of documents, but has no theoretical basis.

1. There are several possible measures other that angle between vectors

2. There is a choice of possible definitions of *tf* and *idf*

3. With fielded searching, there are various ways to adjust the weight given to each field.

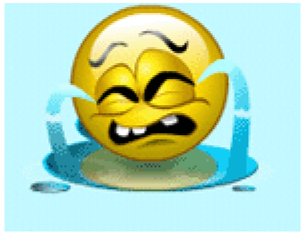
Similarity Measures Compared

$\lvert Q \cap D \rvert$	Simple matching (coordination level match)
$2\dfrac{\lvert Q \cap D \rvert}{\lvert Q \rvert + \lvert D \rvert}$	Dice's Coefficient
$\dfrac{\lvert Q \cap D \rvert}{\lvert Q \cup D \rvert}$	Jaccard's Coefficient
$\dfrac{\lvert Q \cap D \rvert}{\lvert Q \rvert^{1/2} \times \lvert D \rvert^{1/2}}$	Cosine Coefficient (what we studied)
$\dfrac{\lvert Q \cap D \rvert}{\min(\lvert Q \rvert, \lvert D \rvert)}$	Overlap Coefficient

Similarity Measures

- A similarity measure is a function which computes the *degree of similarity* between a pair of vectors or documents
  - since queries and documents are both vectors, a similarity measure can represent the similarity between two documents, two queries, or one document and one query
- There are a large number of similarity measures proposed in the literature, because the *best* similarity measure doesn't exist (yet!)
- With similarity measure between query and documents
  - it is possible to rank the retrieved documents in the order of presumed importance
  - it is possible to enforce certain threshold so that the size of the retrieved set can be controlled
  - the results can be used to reformulate the original query in relevance feedback (e.g., combining a document vector with the query vector)

Problems



- <u>Synonyms</u>: separate words that have the same meaning.
  - E.g. ‘car’ & ‘automobile’
  - They tend to reduce recall
- <u>Polysems</u>: words with multiple meanings
  - E.g. ‘Java’
  - They tend to reduce precision

→ The problem is more general: there is a disconnect between *<u>topics</u>* and *<u>words</u>*

- ‘… a more appropriate model should consider some *conceptual* dimensions instead of words.’ (Gardenfors)

Latent Semantic Analysis (LSA)

- LSA aims to discover something about the meaning behind the words; about the topics in the documents.
- What is the difference between topics and words?
  - Words are observable
  - Topics are not. They are latent.
- How to find out topics from the words in an automatic way?
  - We can imagine them as a compression of words
  - A combination of words
  - Try to formalise this

Latent Semantic Analysis

- Singular Value Decomposition (SVD)

  - A(m*n) = U(m*r) E(r*r) V(r*n)

  - Keep only k eigen values from E
    - A(m*n) = U(m*k) E(k*k) V(k*n)

  - Convert terms and documents to points in k-dimensional space

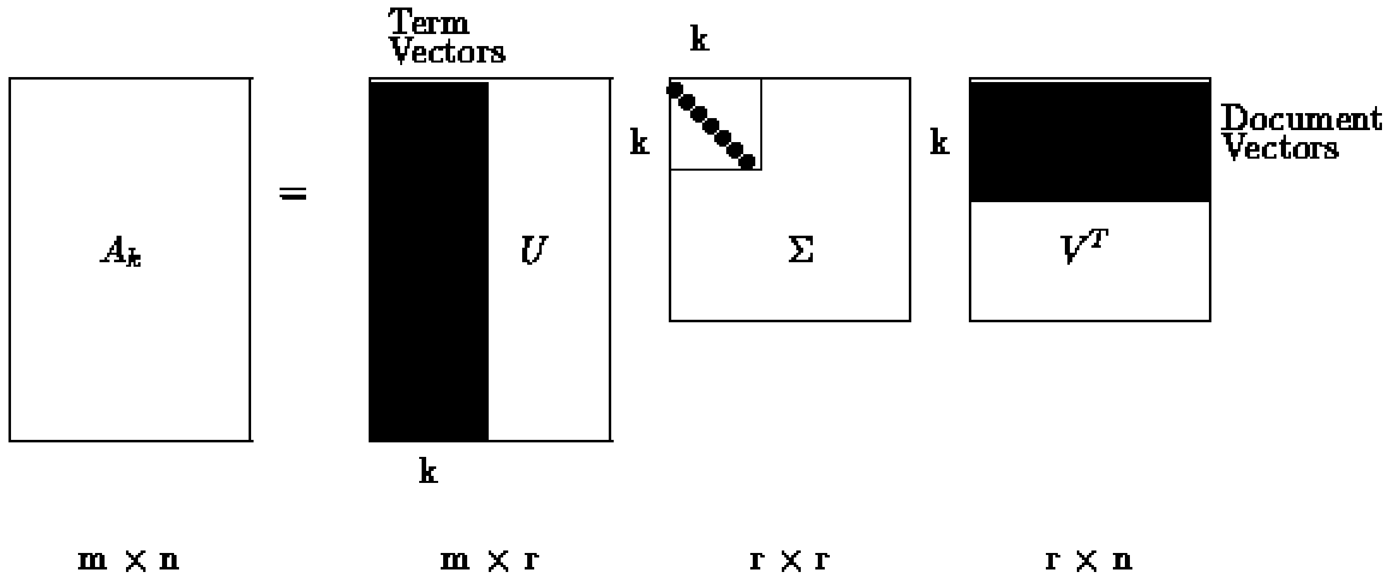
Latent Semantic Analysis

- Singular Value Decomposition

  {A}={U}{S}{V}^{T}

- Dimension Reduction

  {~A}~={~U}{~S}{~V}^{T}



Latent Semantic Analysis

- LSA puts documents together even if they don’t have common words if
  - The docs share frequently co-occurring terms

- Disadvantages:
  - Statistical foundation is missing

PLSA addresses this concern!

PLSA

- Latent Variable model for general co-occurrence data
  - Associate each observation (w,d) with a class variable z Є Z{z_1,…,z_K}

- Generative Model
  - Select a doc with probability P(d)
  - Pick a latent class z with probability P(z|d)
  - Generate a word w with probability p(w|z)

$$\begin{tikzcd} {} \arrow[r, "P(d)"] & d \arrow[r, "P(z|d)"] & z \arrow[r, "P(w|z)"] & w \end{tikzcd}$$

PLSA

- To get the joint probability model

$$
\begin{aligned}
P(d,w) &= P(d)P(w|d)\,, \ \text{where} \\
P(w|d) &= \sum_{z \in \mathcal{Z}} P(w|z)P(z|d)\ .
\end{aligned}
$$

$$P(d,w) = \sum_{z \in \mathcal{Z}} P(z)P(w|z)P(d|z).$$

Model fitting with EM

- We have the equation for log-likelihood function from the aspect model, and we need t

$$\mathcal{L} = \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} n(d, w) \log P(d, w),$$

- Expectation Maximization (EM) is used for this purpose

EM Steps

- E-Step
  - Expectation step where expectation of the likelihood function is calculated with the current parameter values
- M-Step
  - Update the parameters with the calculated posterior probabilities
  - Find the parameters that maximizes the likelihood function

E Step

- It is the probability that a word w occurring in a document d, is explained by aspect z

$$P(z|d,w) = \frac{P(z)P(d|z)P(w|z)}{\sum_{z'} P(z')P(d|z')P(w|z')},$$

(based on some calculations)

M Step

- All these equations use p(z|d,w) calculated in E Step

$$P(w|z) = \frac{\sum_d n(d,w) P(z|d,w)}{\sum_{d,w'} n(d,w') P(z|d,w')},$$

$$P(d|z) = \frac{\sum_w n(d,w) P(z|d,w)}{\sum_{d',w} n(d',w) P(z|d',w)},$$

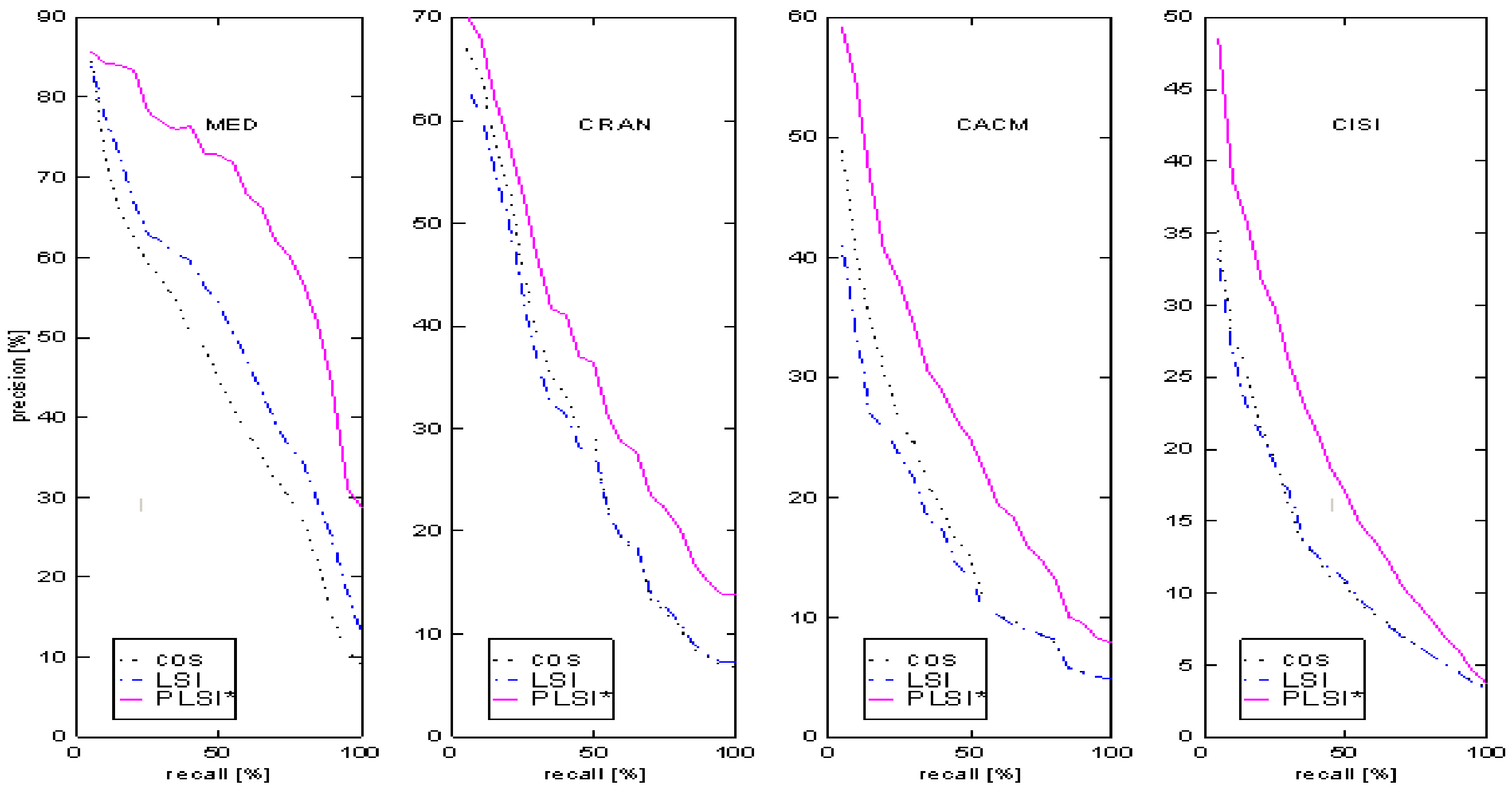
$$P(z) = \frac{1}{R} \sum_{d,w} n(d,w) P(z|d,w), \ R \equiv \sum_{d,w} n(d,w).$$

- Converges to local maximum of the likelihood function

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

The performance of a retrieval system based on this model (PLSI) was found superior to that of both the vector space based similarity (cos) and a non-probabilistic latent semantic indexing (LSI) method. (We skip details here.)



From Th. Hofmann, 2000

Comparing PLSA and LSA

- LSA and PLSA perform dimensionality reduction
  - In LSA, by keeping only K singular values
  - In PLSA, by having K aspects
- Comparison to SVD
  - U Matrix related to P(d|z) (doc to aspect)
  - V Matrix related to P(z|w) (aspect to term)
  - E Matrix related to P(z) (aspect strength)
- The main difference is the way the approximation is done
  - PLSA generates a model (aspect model) and maximizes its predictive power
  - Selecting the proper value of K is heuristic in LSA
  - Model selection in statistics can determine optimal K in PLSA