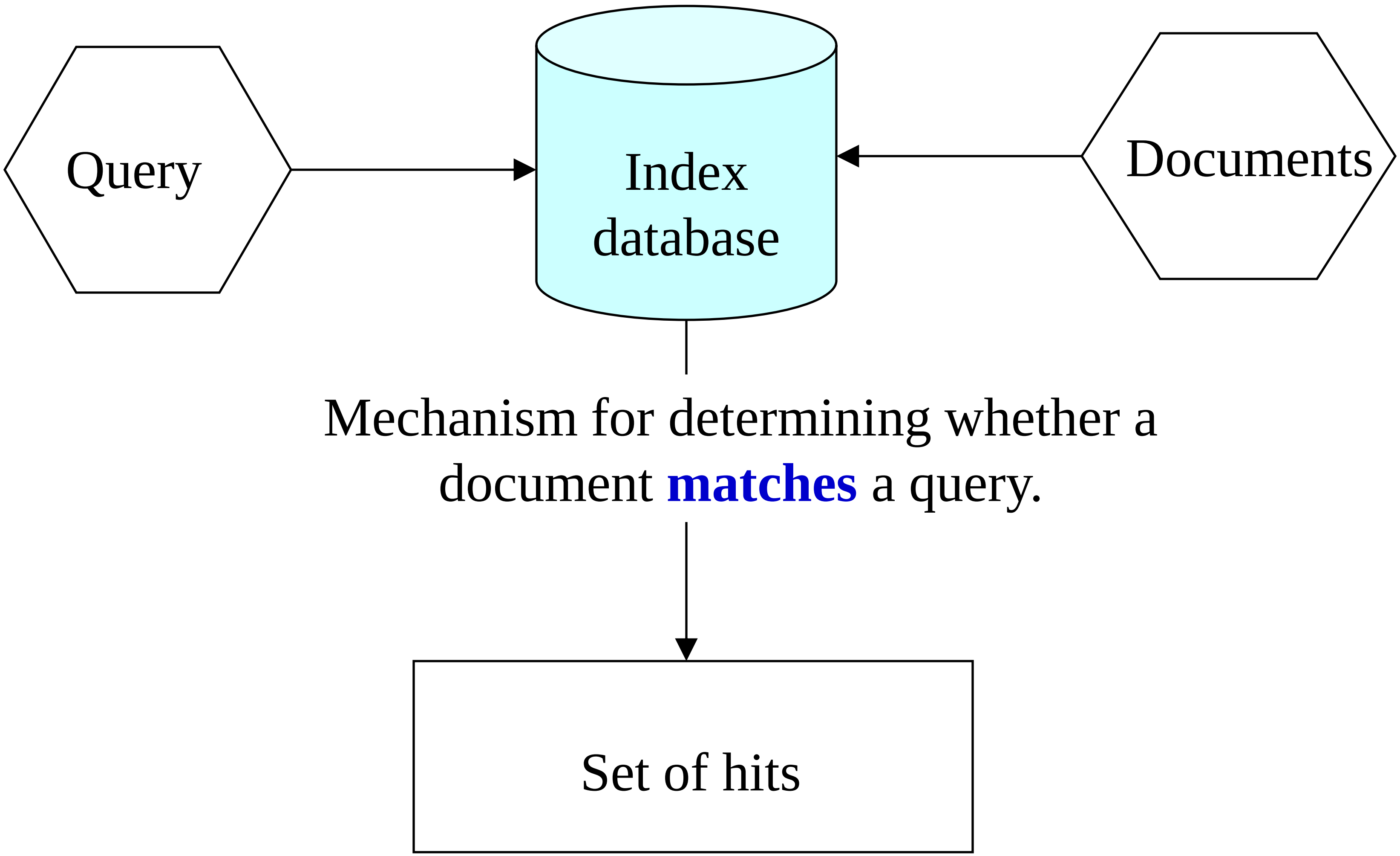
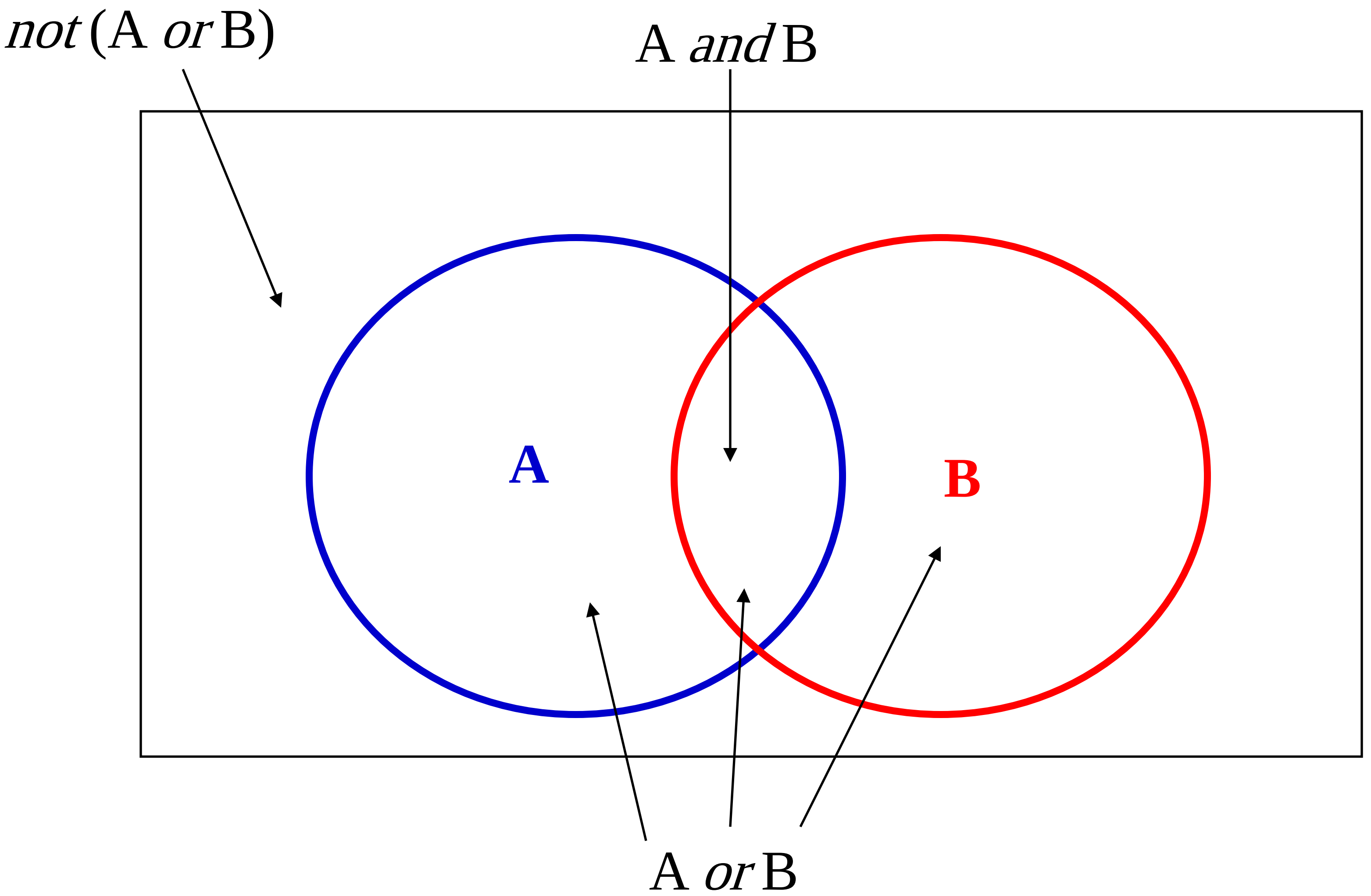
Exact Matching (Boolean Model)



Boolean Diagram



Adjacent and Near Operators

abacus *adj* actor

Terms abacus and actor are adjacent to each other, e.g.,

"abacus actor"

abacus *near 4* actor

Terms abacus and actor are within 4 words of each other, e.g.,

"the actor has an abacus"

Some systems support other operators, such as *with* (two terms in the same sentence) or *same* (two terms in the same paragraph).

Boolean Queries

Boolean query: two or more search terms, related by logical operators, e.g.,

and *or* *not*

Examples:

abacus *and* actor

abacus *or* actor

(abacus *and* actor) *or* (abacus *and* atoll)

not actor

Find all documents that contain the exact words *abacus* and *actor*

Evaluation of Boolean Operators

Precedence of operators must be defined:

adj, *near*	high
and, *not*	↕
or	low

Example

A *and* B *or* C *and* B

is evaluated as

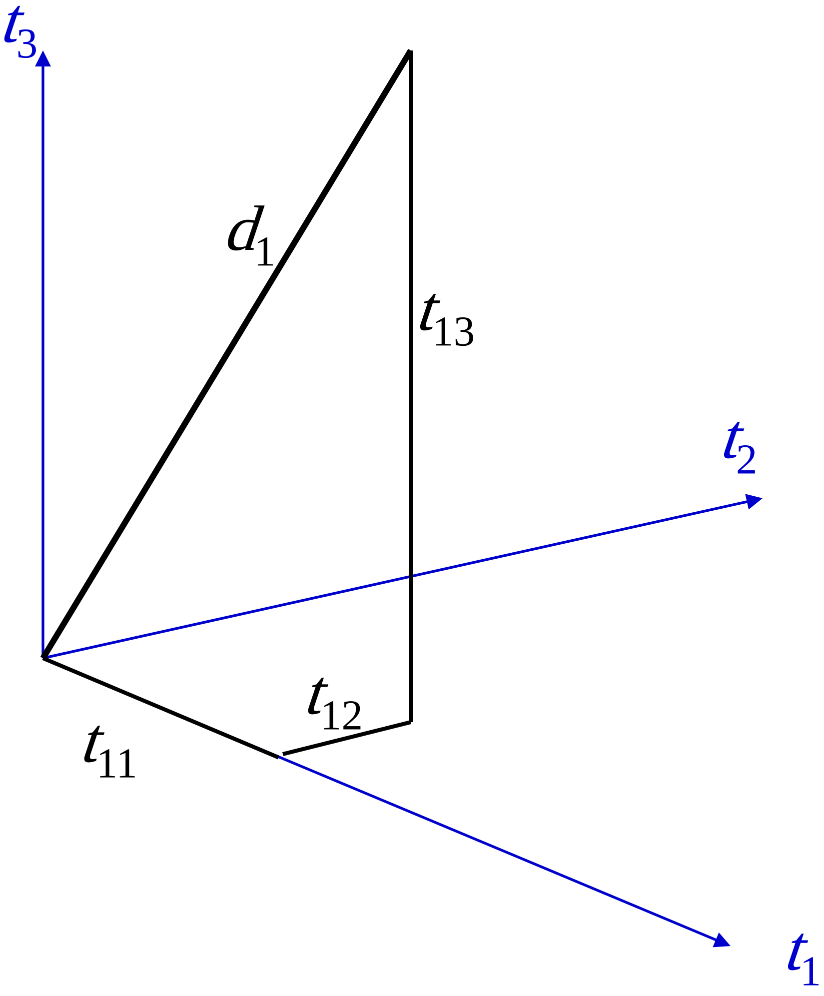
(A *and* B) *or* (C *and* B)

The Term Vector Space

Let n the number of distinct terms in the corpus.

The terms in a document can be represented as vectors in an n-dimensional vector space.

(In the figure $n = 3$.)



Term Vector Space

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*

In this corpus there are eight different terms. Therefore the term vector space, **T**, has 8-dimensions.

Term Vector Space

	d_1	d_2	d_3	d_4	d_5	d_6	d_7
ant	1	1			1		
bee	1	1		1			1
cat			1		1	1	1
dog		1	1				
eel			1				1
fox			1		1		
gnu			1			1	
hog		1		1			

Each document is a vector in the 8-dimensional term vector space T

t_{ij} = 1 if term *i* is in document j and zero otherwise

Term Vector Space with Weighting

Term vector space

n-dimensional space, where n is the number of different terms used to index a set of documents (i.e. size of the word list).

Vector

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

$t_{ij} > 0$ if term i occurs in document j

$t_{ij} = 0$ otherwise

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

Sparse Matrix

The term vector space is a very sparse matrix.

An **inverted file** is an efficient way to represent a term vector space. It also provides a convenient method to store additional data.

Most methods of storing sparse matrices are designed for either row processing or column processing. An inverted file is organized for row processing, i.e., all the information about a given **term** is stored together.

Inverted File

Inverted file:

An inverted file is list of search terms that are organized for **associative look-up**, i.e., to answer the questions:

- In which documents does a specified search term appear?
- Where within each document does each term appear? (There may be several occurrences.)

In a free text search system, the **word list** and the **postings** file together provide an inverted file system. In addition, they contain the data needed to calculate weights and information that is used to display results.

Inverted File -- Definitions

Word
bee
cat
dog
eel
fox
gnu
hog

ant

The **word list** is a list of all the distinct terms in the corpus after the removal of stop words and stemming. This is sometimes called a **vocabulary file**.

Inverted File -- Definitions

Posting: Entry in an inverted file system that applies to a single instance of a term within a document, e.g., there might be three postings for "abacus":

abacus	3

"abacus" is in document 3

abacus	19

abacus	22

Inverted List: A list of all the postings in an inverted file system that apply to a specific word, e.g.

abacus	3	19	22

"abacus" is in documents 3, 19 & 22

This is a sparse representation of a row in the term vector matrix

Use of Inverted Files for Evaluating a Boolean Query

Examples: abacus *and* actor

Postings for abacus

3	19	22

Postings for actor

2	19	29

Document 19 is the only document that contains both terms, "abacus" and "actor".

To evaluate the *and* operator, merge the two inverted lists with a logical *AND* operation.

Enhancements to Inverted Files -- Concept

Location: Each posting holds information about the location of each term within the document.

<u>Uses</u>

user interface design -- highlight location of search term

adjacency and *near* operators (in Boolean searching)

Frequency: Each inverted list includes the number of postings for each term.

<u>Uses</u>

term weighting

query processing optimization

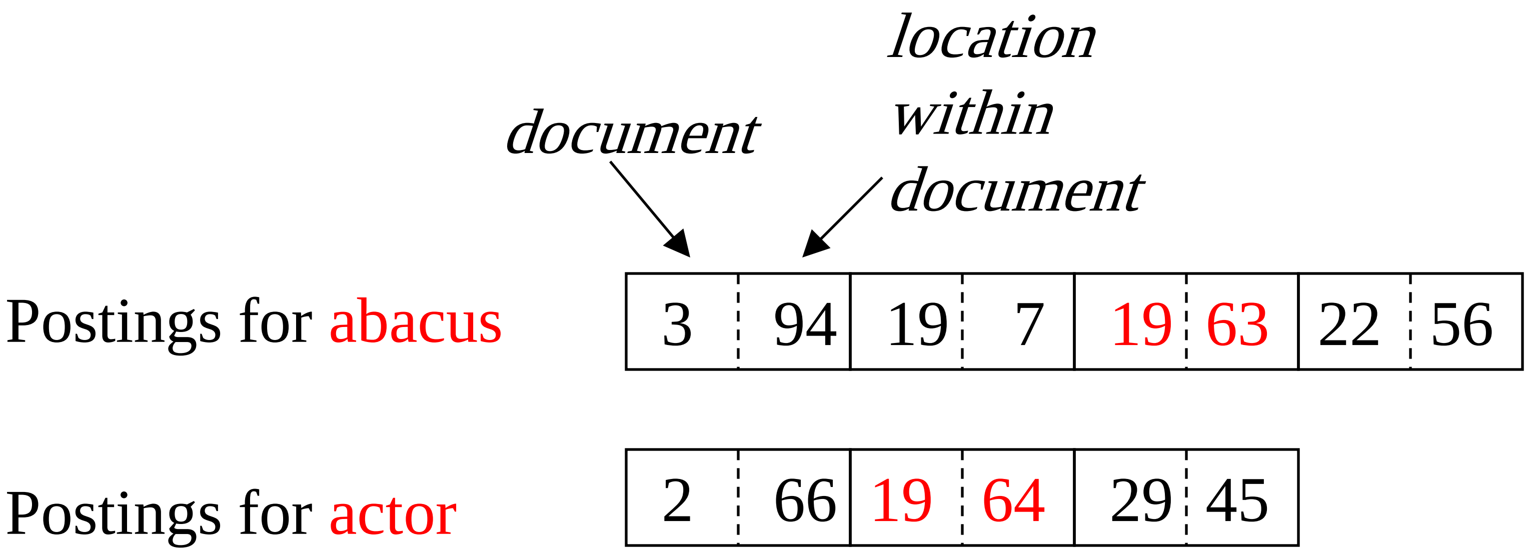
Inverted File -- Concept (Enhanced)

Word	*Postings*	*Document*	*Location*
abacus	4	3	94
		19	7
		19	63
		22	56
actor	3	2	66
		19	64
		29	45
aspen	1	5	43
atoll	3	11	3
		11	70
		34	40

Inverted list
for term *actor*

Evaluating an Adjacency Operation

Example: abacus *adj* actor



Document 19, locations 63 and 64, is the only occurrence of the terms "abacus" and "actor" adjacent.

Query Matching: Boolean Methods

Query: (abacus *or* asp*) *and* actor

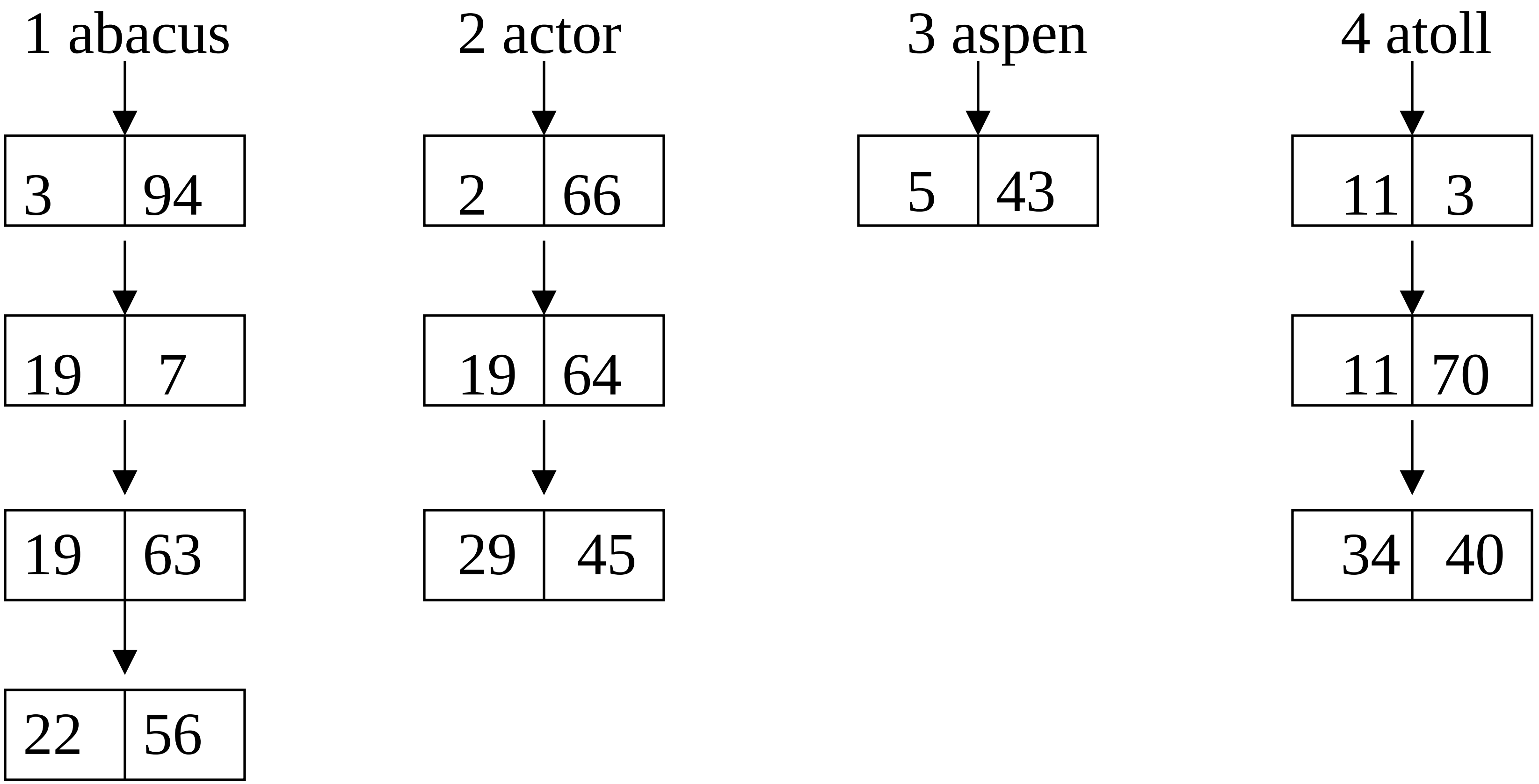
1. From the index file (word list), find the postings file for:

   "abacus"
   every word that begins "asp"
   "actor"

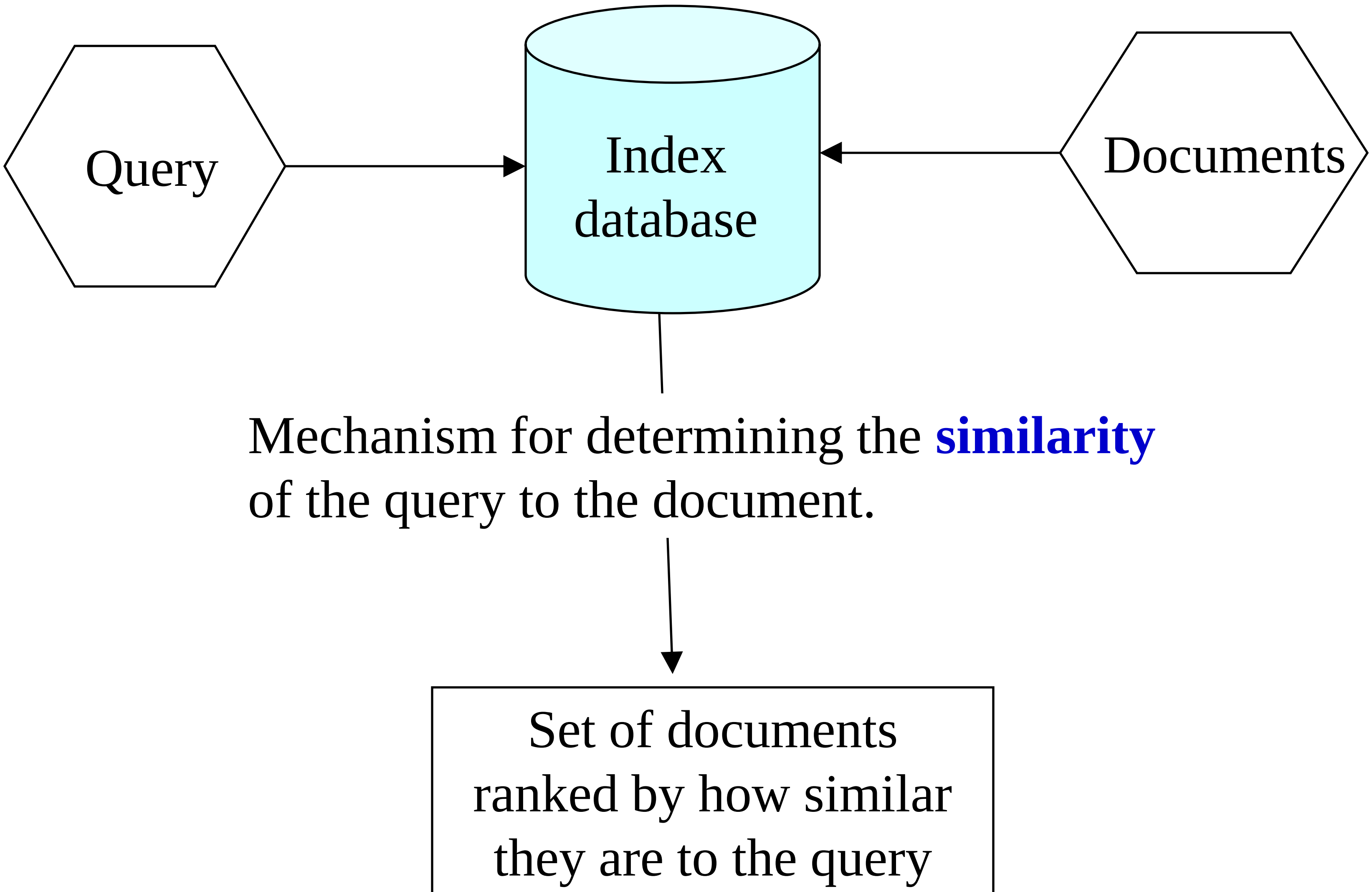
2. Merge these posting lists. For each document that occurs in any of the postings lists, evaluate the Boolean expression to see if it is *true* or *false*.

Step 2 should be carried out in a single pass.

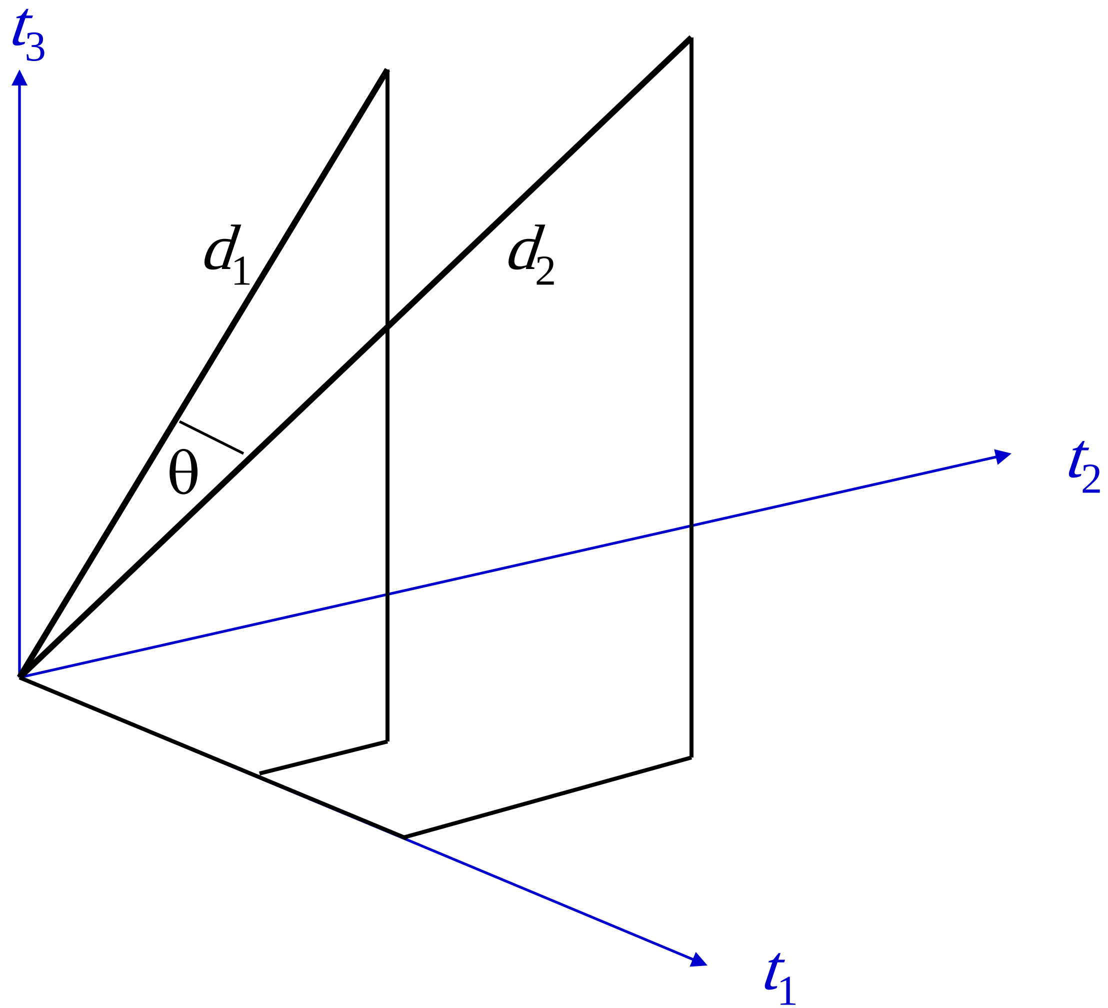
Use of Postings File for Query Matching



Similarity Ranking Methods



Two Documents Represented in 3-Dimensional Term Vector Space



Vector Space Revision

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

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

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

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

Inner product (or dot product) is given by

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

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

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

Similarity
(No Weighting)

How similar are the following documents?

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*

Term Vector Space (No Weighting)

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

$t_{ij} = 1$ if term i is in document j and zero otherwise

Example: Comparing Documents No Weighting

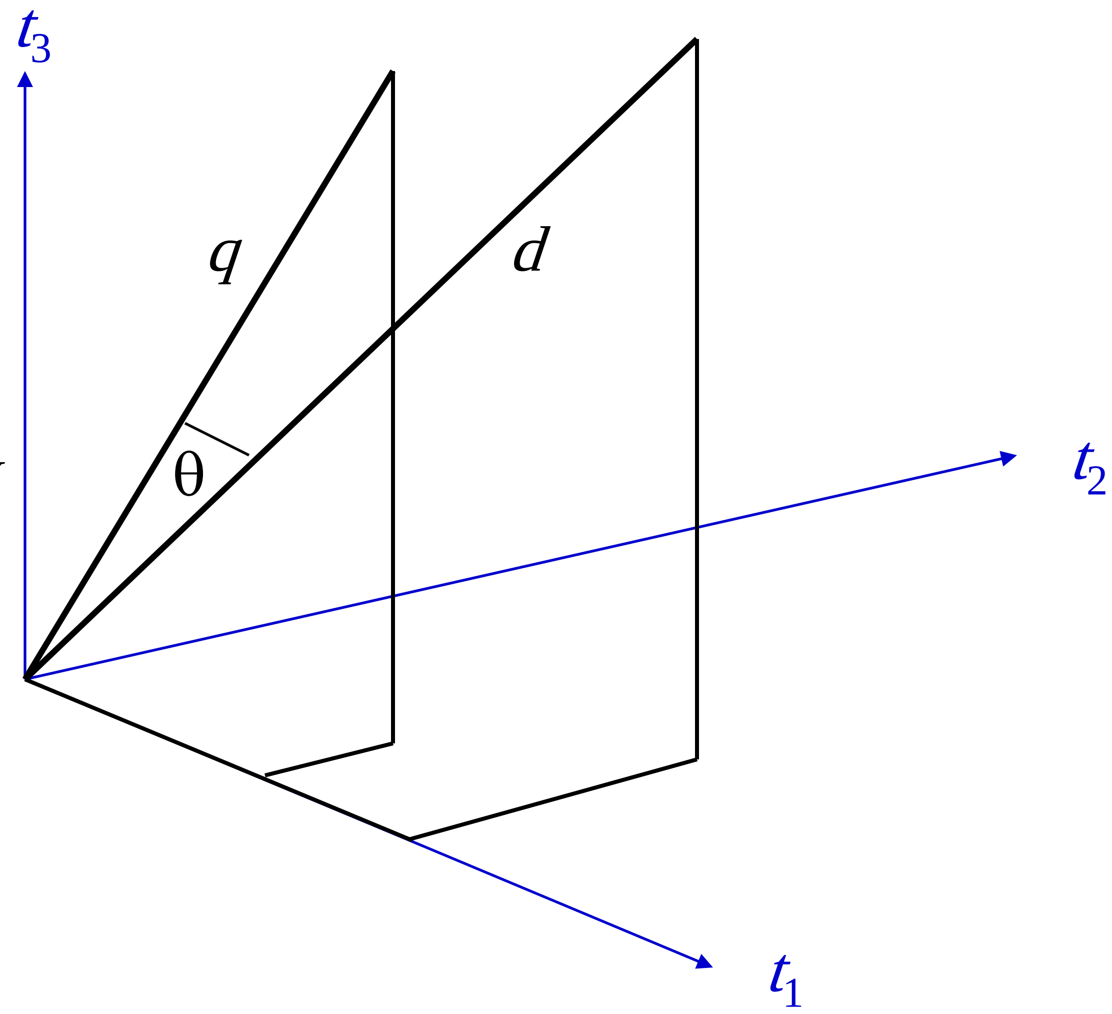
Similarity of documents in example

$$\mathrm{sim}(d_1, d_2) = \frac{\mathbf{d}_1 . \mathbf{d}_2}{|\mathbf{d}_1| \, |\mathbf{d}_2|}$$

	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

Similarity between a Query and a Document in 3-Dimensional Term Vector Space

cos(θ) is used as a measure of similarity



Similarity of Query to Documents

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*

Term Vector Space:
(Term Incidence Matrix: no Weighting)

	q	d_1	d_2	d_3
ant	1	1	1	
bee		1	1	
cat				1
dog	1		1	1
eel				1
fox				1
gnu				1
hog			1	
<u>length</u>	$\sqrt{2}$	$\sqrt{2}$	$\sqrt{4}$	$\sqrt{5}$

Calculate Ranking

Similarity of query to documents in example:

	d_1	d_2	d_3
q	$1/2$ 0.5	$1/\sqrt{2}$ 0.71	$1/\sqrt{10}$ 0.32

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

d_2, d_1, d_3

Simple Uses of Vector Similarity in Information Retrieval

Ranking

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

[This is the standard practice.]

Extending the Basic Concept with Term Weighting

An improved **measure of similarity** might take account of:

(a) Whether the terms are common or unusual

(c) How many times each term appears in a document

(d) The lengths of the documents

(e) The place in the document that a term appears

(f) Terms that are adjacent to each other (phrases)

Weighting: Unnormalized Term Frequency (*tf*)

t_{ij} = number of times that term *i* appears in document *j*

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

Example: Unnormalized Form of Term Frequency (*tf*)

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 previous example.]

Similarity of query to documents: (Unnormalized 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*

Term Vector Space: (Weighting by Unnormalized Form of Term Frequency)

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

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

Term Vector Space with Weighting

Term vector space

n-dimensional space, where *n* is the number of different terms used to index a set of documents (i.e. size of the word list).

Vector

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

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

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

Vector Similarity Computation with Weights

Similarity between documents d_p and d_q is defined as:

$$\mathrm{sim}(d_i, d_j) = \frac{\sum_{i=1}^{n} t_{ip} t_{iq}}{|\mathbf{d}_p| \, |\mathbf{d}_q|}$$

Where $\mathbf{d}_p$ and $\mathbf{d}_q$ are the corresponding weighted term vectors

Choice of Weights

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

What weights lead to the best information retrieval?

Evaluation

Before we can decide whether one system of weights is **better** than another, we need systematic and repeatable methods to **evaluate** methods of information retrieval.

Methods for Selecting Weights

Empirical

Test a large number of possible weighting schemes with actual data.

Model based

Develop a mathematical model of word distribution and derive weighting scheme theoretically. (*Probabilistic model of information retrieval.*)

Weighting
Term Frequency (tf)

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

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.

Normalized Form of Term Frequency: Free-text Document

Length of document

Unnormalized method is to use f_{ij} as the term frequency.

...but, in free-text documents, terms are likely to appear more often in long documents. Therefore f_{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 j. This partially corrects for variations in the length of the documents.

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

Term frequency (tf):

$$tf_{ij} = f_{ij} / m_j$$

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)

Inverse Document Frequency: Concept

Some terms appear much more often than others across the documents of a corpus.

A term that occurs in only a few documents is likely to be a better discriminator that 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 i occurs is n_i.

Simple method

We could define document frequency as n_i/N.

A possible method might be to use the inverse, N/n_i, as a weight. This would give greater weight to words that appear in fewer documents.

Inverse Document Frequency

A standard method

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

Inverse document frequency (idf):

$$idf_j = \log_2 (N/n_i) + 1 \qquad\qquad n_i > 0$$

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

<u>Example</u>

N = 1,000 documents

term i	n_i	N/n_i	idf_i
t_1	100	10.00	4.32
t_2	500	2.00	2.00
t_3	900	1.11	1.13
t_4	1,000	1.00	1.00

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 *i* in document *j*)

= (term frequency) * (inverse document frequency)

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

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

$$= (f_{ij} / m_j) * (\log_2 (N/n_i) + 1) \quad \text{when } n_i > 0$$

Structured Text

Structured text

Structured texts, e.g., queries, catalog records or abstracts, have different distribution of terms from free-text. A modified expression for the term frequency is:

$$tf_{ij} = K + (1 - K)*f_{ij} / m_j \quad \text{when } f_{ij} > 0$$

K is a parameter between 0 and 1 that can be tuned for a specific collection.

Structured Text

Query

To weigh terms in the query, Salton and Buckley recommend K equal to 0.5.

However, in practice it is rare for a term to be repeated in a query. Therefore the standard form of *tf* can be used, i.e., with $K = 0$ and $m = 1$.