UNIT II

Basic IR Models – Boolean Model – TF-IDF (Term Frequency/Inverse Document Frequency) Weighting – Vector Model – Probabilistic Model – Latent Semantic Indexing Model – Neural Network Model –Retrieval Evaluation – Retrieval Metrics – Precision and Recall – Reference Collection –User-based Evaluation – Relevance Feedback and Query Expansion – Explicit Relevance Feedback.

2.1 Introduction

Modeling

Modeling in IR is a complex process aimed at producing a ranking function

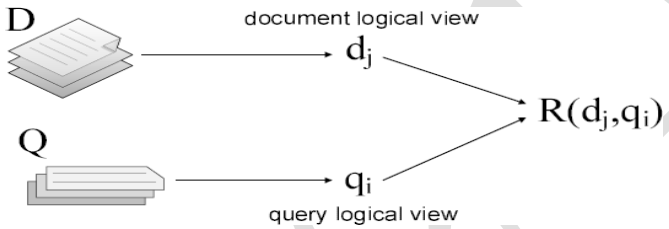
Ranking function: a function that assigns scores to documents with regard to a given query.

This process consists of two main tasks:

- The conception of a logical framework for representing documents and queries
- The definition of a ranking function that allows quantifying the similarities among documents and queries

IR systems usually adopt index terms to index and retrieve documents

IR Model Definition:



An IR model is a quadruple [D, Q, F, R(qi, dj)] where

1. D is a set of logical views for the documents in the collection
2. Q is a set of logical views for the user queries
3. F is a framework for modeling documents and queries
4. R(qi, dj) is a ranking function

A Taxonomy of IR Models

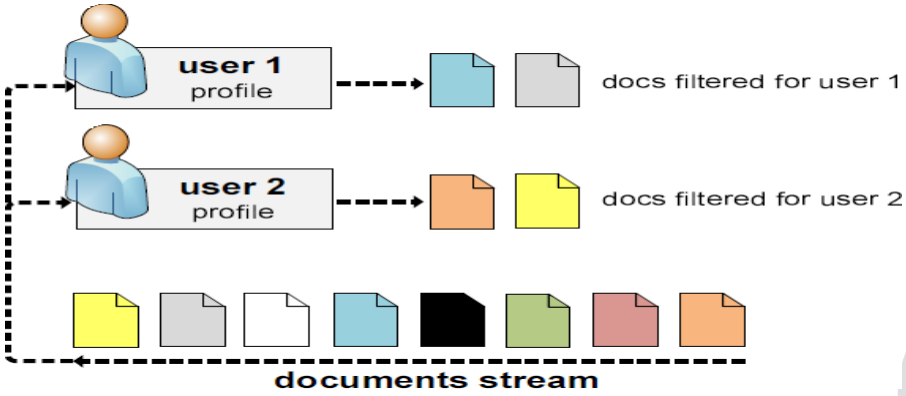
Retrieval models most frequently associated with distinct combinations of a document logical view and a user task. The users task includes retrieval and browsing. In retrieval

i) Ad Hoc Retrieval:

The documents in the collection remain relatively static while new queries are submitted to the system.

ii) Filtering

The queries remain relatively static while new documents come into the system



Classic IR model:

Each document is described by a set of representative keywords called index terms. Assign a numerical weights to distinct relevance between index terms.

Three classic models: Boolean, vector, probabilistic

Boolean Model :

The Boolean retrieval model is a model for information retrieval in which we can pose any query which is in the form of a Boolean expression of terms, that is, in which terms are combined with the operators AND, OR, and NOT. The model views each document as just a set of words. Based on a binary decision criterion without any notion of a grading scale. Boolean expressions have precise semantics.

Vector Model

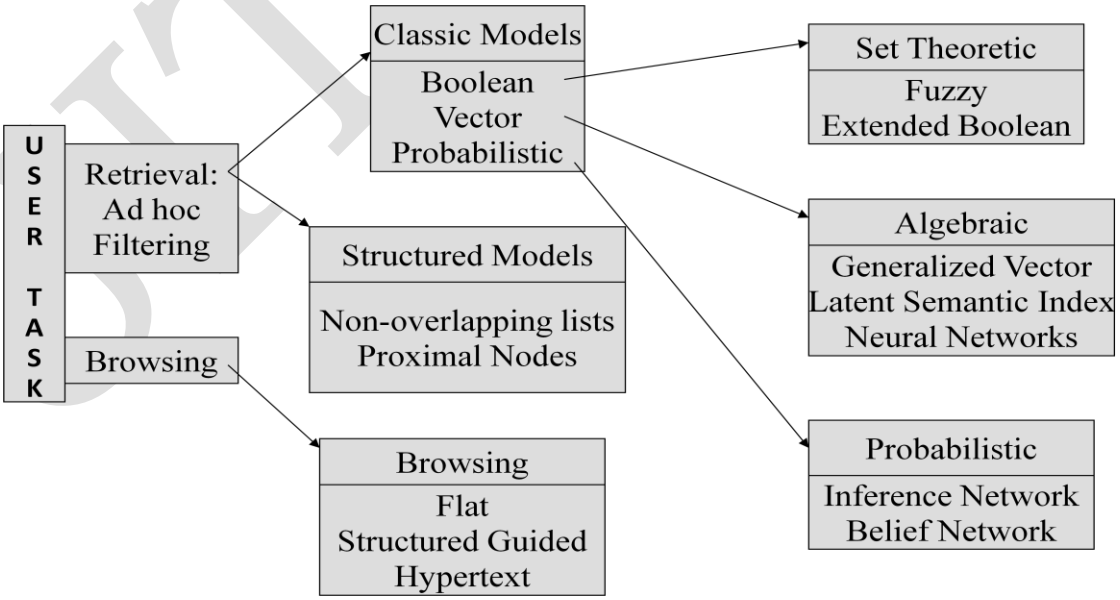
Assign non-binary weights to index terms in queries and in documents. Compute the similarity between documents and query. More precise than Boolean model.

Probabilistic Model

The probabilistic model tries to estimate the probability that the user will find the document dj relevant with ratio

P(dj relevant to q)/P(dj nonrelevant to q)

Given a user query q, and the ideal answer set R of the relevant documents, the problem is to specify the properties for this set. Assumption (probabilistic principle): the probability of relevance depends on the query and document representations only; ideal answer set R should maximize the overall probability of relevance



Basic Concepts

- Each document is represented by a set of representative keywords or index terms
- Index term:
  - In a restricted sense: it is a keyword that has some meaning on its own; usually plays the role of a noun
  - In a more general form: it is any word that appears in a document
- Let, t be the number of index terms in the document collection k_i be a generic index term Then,
- The vocabulary $V = \{k_1, \ldots, k_t\}$ is the set of all distinct index terms in the collection

$$V = \boxed{k_1 \quad k_2 \quad k_3 \quad \ldots \quad k_t}$$ vocabulary of *t* index terms

The Term-Document Matrix

- The occurrence of a term t_i in a document d_j establishes a relation between t_i and d_j
- A term-document relation between t_i and d_j can be quantified by the frequency of the term in the document
- In matrix form, this can written as

$$\begin{array}{c} \\ t1 \\ t2 \\ t3 \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{bmatrix} f_{1,1} & f_{1,2} \\ f_{2,1} & f_{2,2} \\ f_{3,1} & f_{3,2} \end{bmatrix} \end{array}$$

- where each $f_{i,j}$ element stands for the frequency of term t_i in document d_j
- Logical view of a document: from full text to a set of index terms

2.2 Boolean Retrieval Models

Definition : The Boolean retrieval model is a model for information retrieval in which the query is in the form of a Boolean expression of terms, combined with the operators AND, OR, and NOT. The model views each document as just a set of words.

- Simple model based on set theory and Boolean algebra
- The Boolean model predicts that each document is either relevant or non-relevant

Example :

A fat book which many people own is Shakespeare"s Collected Works.

Problem : To determine which plays of Shakespeare contain the words Brutus AND Caesar AND NOT Calpurnia.

Method1 : Using Grep

The simplest form of document retrieval is for a computer to do the linear scan through documents. This process is commonly referred to as *grepping* through text, after the Unix command grep. Grepping through text can be a very effective process, especially

given the speed of modern computers, and often allows useful possibilities for wildcard pattern matching through the use of regular expressions.

To Perform simple querying of modest collections , we need :

1. To process large document collections quickly.
2. To allow more flexible matching operations.
   For example, it is impractical to perform the query "Romans NEAR countrymen" with grep , where NEAR might be defined as "within 5 words" or "within the same sentence".
3. To allow ranked retrieval: in many cases we want the best answer to an information need among many documents that contain certain words. The way to avoid linearly scanning the texts for each query is to *index* documents in advance.

Method2: Using Boolean Retrieval Model

- The Boolean retrieval model is a model for information retrieval in which we can pose any query which is in the form of a Boolean expression of terms, that is, in which terms are combined with the operators AND, OR, and NOT. The model views each document as just a set of words.
- *Terms* are the indexed units . we have a vector for each term, which shows the documents it appears in, or a vector for each document, showing the terms that occur in it. *The result is a binary term-document incidence matrix, as in Figure* .

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello.	Macbeth
Anthony	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

A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise.

- To answer the query Brutus AND Caesar AND NOT Calpurnia, we take the vectors for Brutus, Caesar and Calpurnia, complement the last, and then do a bitwise AND:

```
110100 AND 110111 AND 101111 = 100100
```

Solution : Antony and Cleopatra and Hamlet

Antony and Cleopatra, Act III, Scene ii
Agrippa [Aside to Domitius Enobarbus]: Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii
Lord Polonius: I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Results from Shakespeare for the query Brutus AND Caesar AND NOT Calpurnia.

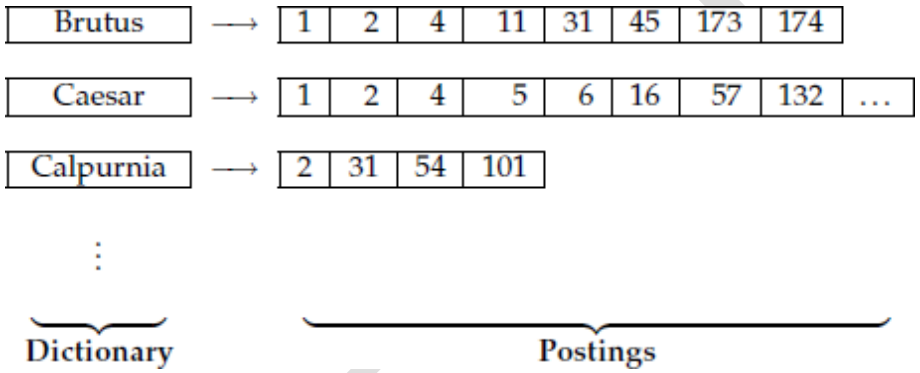
Consider $N = 10^6$ documents, each with about 1000 tokens ⇒ total of 10^9 tokens
On average 6 bytes per token, including spaces and punctuation ⇒ size of document collection is about $6 \cdot 10^9 = 6$ GB

Assume there are $M = 500{,}000$ distinct terms in the collection
$M = 500{,}000 \times 10^6$ = half a trillion 0s and 1s.
But the matrix has no more than one billion 1s. Matrix is extremely sparse.What is a better representations? We only record the 1s.(Inverted Index)

2.3 Inverted index / inverted File

It is the most efficient structure for supporting ad hoc text search. It has become the standard term in information retrieval. For each term *t*, we store a list of all documents that contain *t*. The two parts of an inverted index. The dictionary is commonly kept in memory, with pointers to each postings list, which is stored on disk.

i) **Dictionary / vocabulary /lexicon :** we use dictionary for the data structure and vocabulary for the set of terms, The dictionary in Figure has been sorted alphabetically
ii) **Posting :** for each term, we have a list of Document ID in which the term present .The list is then called a postings list (or inverted list), and each postings list is sorted by document ID.



The two parts of an inverted index. The dictionary is commonly kept in memory, with pointers to each postings list, which is stored on disk.

Building a Inverted index

To gain the speed benefits of indexing at retrieval time, we have to build the index in advance. Building a basic inverted index is *sort-based indexing* .

The major steps in this are:

1. Collect the documents to be indexed:

   Friends, Romans, countrymen. | So let it be with Caesar ...

2. Tokenize the text, turning each document into a list of tokens:

   Friends | Romans | countrymen | So ...

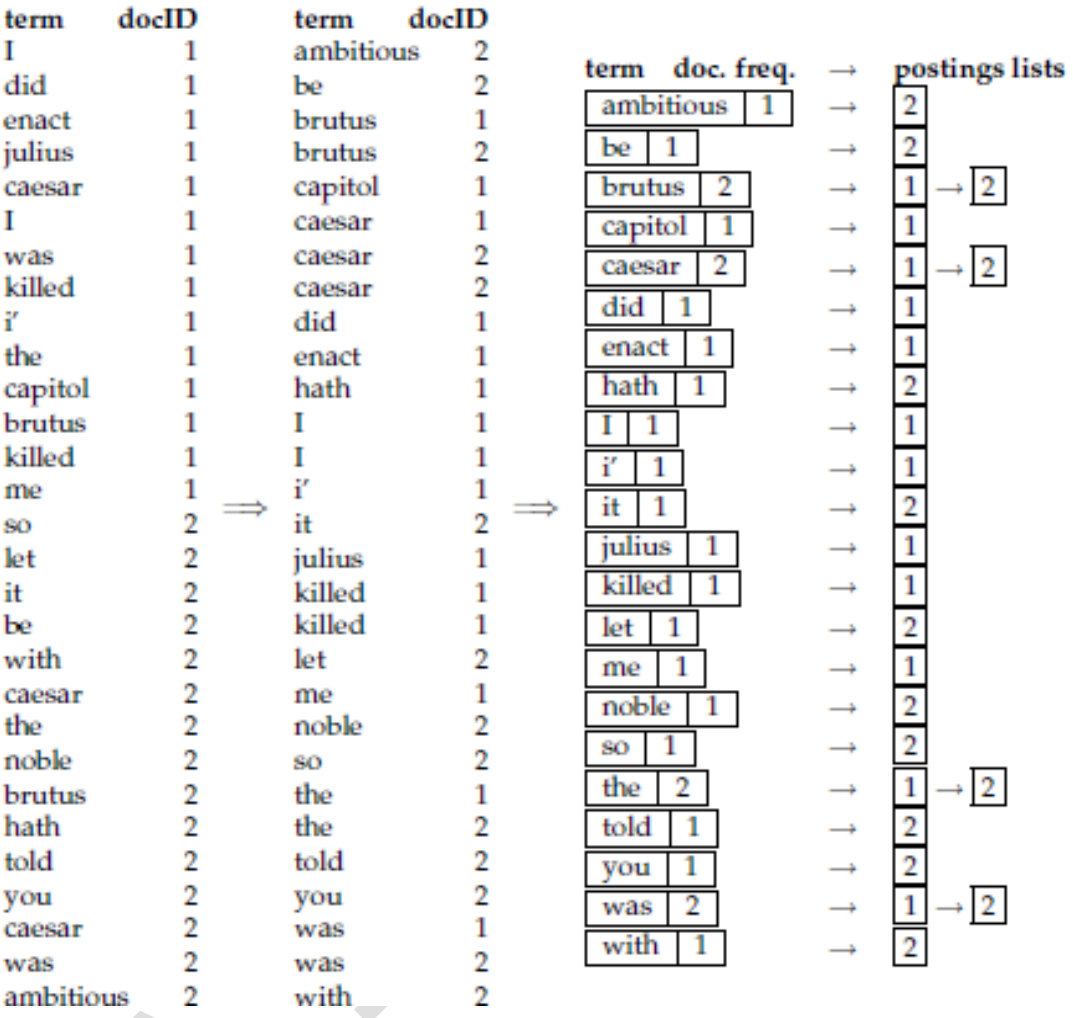
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms:

   friend | roman | countryman | so ...

4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Doc 1
I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Doc 2
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:



DocID :

Each document has a unique serial number, known as the document identifier (*docID*). During index construction, simply assign successive integers to each new document when it is first encountered.

Input Dictionary & posting :

The input to indexing is a list of normalized tokens for each document, which we can equally think of as a list of pairs of term and docID . The core indexing step is *sorting* this list , Multiple occurrences of the same term from the same document are then merged. Instances of the same term are then grouped, and the result is split into a *dictionary* and *postings*

Document Frequency :

The dictionary records some statistics, such as the number of documents which contain each term . This information is not vital for a basic Boolean search engine, but it allows us to improve the efficiency of the search engine at query time, and it is a statistic later used in many ranked retrieval models. The postings are secondarily sorted by docID. This provides the basis for efficient query processing.

Storage (dictionary & postings lists) :

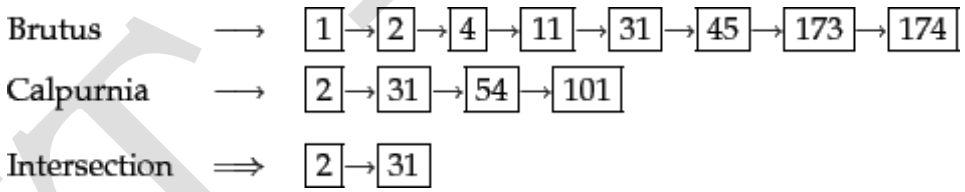
1. A fixed length array would be wasteful as some words occur in many documents, and others in very few.
2. For an in-memory postings list - two good alternatives
   a. singly linked lists : Singly linked lists allow cheap insertion of documents into postings lists, and naturally extend to more advanced indexing strategies such as skip lists, which require additional pointers.
   b. Variable length arrays : win in space requirements by avoiding the overhead for pointers and in time requirements because their use of contiguous memory increases speed on modern processors with memory caches. Variable length arrays will be more compact and faster to traverse.
3. A hybrid scheme with a linked list of fixed length arrays for each term. When postings lists are stored on disk, they are stored (perhaps compressed) as a contiguous run of postings without explicit pointers, so as to minimize the size of the postings list and the number of disk seeks to read a postings list into memory.

Processing Boolean queries

To process a query using an inverted index and the basic Boolean retrieval model Consider processing the *simple conjunctive query* : Brutus AND Calpurnia over the inverted index partially shown in Figure

Steps :

1. Locate Brutus in the Dictionary
2. Retrieve its postings
3. Locate Calpurnia in the Dictionary
4. Retrieve its postings
5. Intersect the two postings lists, as shown in Figure



Intersecting the postings lists for Brutus and Calpurnia Algorithm for the intersection of two postings lists P1 and P2.

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

There is a simple and effective method of intersecting postings lists using the merge algorithm. we maintain pointers into both lists and walk through the two postings lists simultaneously, in time linear in the total number of postings entries. At each step, we compare the docID pointed to by both pointers. If they are the same, we put that docID in the results list, and advance both pointers. Otherwise we advance the pointer pointing to the smaller docID. To use this algorithm, postings is be sorted by a single global ordering. Using a numeric sort by docID is one simple way to achieve this.

Query optimization

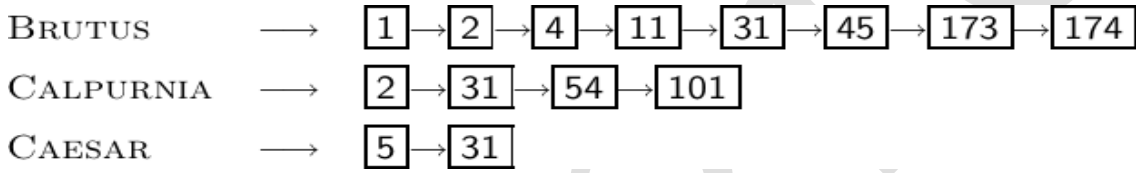
Case1:

Consider a query that is an and of n terms, n > 2
For each of the terms, get its postings list, then and them together Example

query: BRUTUS AND CALPURNIA AND CAESAR

Simple and effective optimization: Process in order of increasing frequency

Start with the shortest postings list, then keep cutting further In this example,
first CAESAR, then CALPURNIA, then BRUTUS



Case2:

Example query: (MADDING OR CROWD) and (IGNOBLE OR STRIFE)
Get frequencies for all terms, Estimate the size of each or by the sum of its frequencies (conservative),
Process in increasing order of or sizes

```
INTERSECT(⟨t1, ..., tn⟩)
1  terms ← SORTBYINCREASINGFREQUENCY(⟨t1, ..., tn⟩)
2  result ← postings(first(terms))
3  terms ← rest(terms)
4  while terms ≠ NIL and result ≠ NIL
5  do result ← INTERSECT(result, postings(first(terms)))
6     terms ← rest(terms)
7  return result
```

Drawbacks of the Boolean Model

- Retrieval based on binary decision criteria with no notion of partial matching
- No ranking of the documents is provided (absence of a grading scale)
- Information need has to be translated into a Boolean expression, which most users find awkward
- The Boolean queries formulated by the users are most often too simplistic
- The model frequently returns either too few or too many documents in response to a user query

2.4 Term weighting

Search Engine should return in order the documents most likely to be useful to the searcher . To achieve this , ordering documents with respect to a query - called Ranking

Term-Document Incidence Matrix

A Boolean model only records term presence or absence, Assign a score – say in [0, 1] – to each document , it measures how well document and query "match"

For One-term query "BRUTUS" , score is 1 if it is present in the document , 0 otherwise ,
More appearances of term in document have higher score

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello.	Macbeth
Anthony	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

Document represented by binary vector Є {0,1}|V|

Term Frequency tf

One of the weighting scheme is Term Frequency and is denoted tft,d, with the subscripts denoting the term and the document in order.
Term frequency TF(t, d) of term t in document d = number of times that t occurs in d

Ex: Term-Document Count Matrix

but we would like to give more weight to documents that have a term several times as opposed to ones that contain it only once. To do this we need term frequency information the number of times a term occurs in a document .
Assign a score to represent the number of occurrences

	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

Se

Document represented by count vector Є N^V

Bag of Words Model

The exact ordering of the terms in a document is ignored but the number of occurrences of each term is important.
Example : two documents with similar bag of words representations are similar in content.

"Mary is quicker than John" →"John is quicker than Mary"

This is called the bag of words model. In a sense, step back: The positional index was able to distinguish these two documents

How to use tf for query-document match scores?

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. We use Log frequency weighting.

Log-Frequency Weighting

Log-frequency weight of term t in document d is calculated as

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

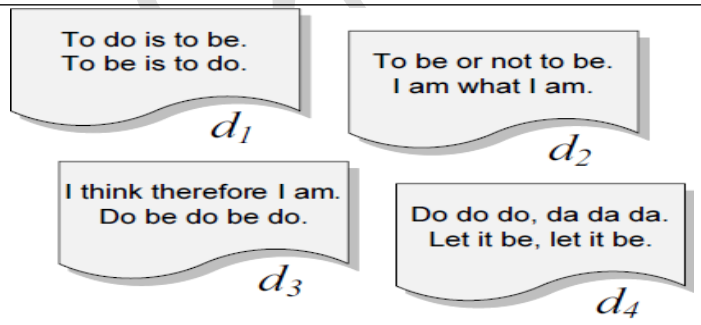
$tf_{t,d} \rightarrow w_{t,d}$: 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

Document Frequency & Collection Frequency

Document frequency DF(t): the number of documents in the collection that contain a term t
Collection frequency CF(t): the total number of occurrences of a term t in the collection

Example :

TF(do,d1) =2
TF(do,d2) =0
TF(do,d3) =3
TF(do,d4) =3
CF (do) =8
DF(do) =3



Rare terms are more informative than frequent terms, to capture this we will use document frequency (df)

Example: rare word ARACHNOCENTRIC
Document containing this term is very likely to be relevant to query ARACHNOCENTRIC
We want high weight for rare terms like ARACHNOCENTRIC

Example: common word THE
Document containing this term can be about anything
We want very low weight for common terms like THE

Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

Document frequency is more meaningful , as we see in the example above , there are few documents that contain "insurance" to get a higher boost for a query on "insurance" than the many documents containing "try" to get from a query on "try"

Inverse Document Frequency (idf Weight)

It estimates the rarity of a term in the whole document collection. idf_t is an inverse measure of the informativeness of t and $idf_t <= N$

df_t is the document frequency of t: the number of documents that contain t Informativeness idf (inverse document frequency) of t:

$$idf_t = \log_{10}(N/df_t)$$

log (N/df_t) is used instead of N/df_t to diminish the effect of idf.

N: the total number of documents in the collection (for example :806,791 documents)

- IDF(t) is high if t is a rare term
- IDF(t) is likely low if t is a frequent term

N = 1, 000, 000

$idf_t = \log_{10} \frac{1,000,000}{df_t}$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Effect of idf on Ranking

Does idf have an effect on ranking for one-term queries, like IPHONE? Ans: No , idf will be effect for >1 term queries.

Query CAPRICIOUS PERSON: idf puts more weight on CAPRICIOUS than PERSON, since it is a rare term.

2.5 TF-IDF Weighting

tf-idf weight of a term: product of tf weight and idf weight , Best known weighting scheme in information retrieval. TF(t, d) measures the importance of a term t in document d , IDF(t) measures the importance of a term t in the whole collection of documents

TF/IDF weighting: putting TF and IDF together

TFIDF(t, d) = TF(t, d) x IDF(t)

$$w_{t,d} = (1 + \log_{10} tf_{t,d}) \times \log_{10}(N/df_t)$$ (if log tf is used)

- High if t occurs many times in a small number of documents, i.e., highly discriminative in those documents
- Not high if t appears infrequent in a document, or is frequent in many documents, i.e., not discriminative
- Low if t occurs in almost all documents, i.e., no discrimination at all

Simple Query-Document Score

- scoring finds whether or not a query term is present in a zone (Zones: document features whose content can be arbitrary free text – Examples: title, abstracts) within a document.
- Score for a document-query pair: sum over terms t in both q and d:

$$\text{score} = \sum_{t \in q \cap d} (1 + \log_{10} \text{tf}_{t,d})$$

If the Query contains more than one terms , Score for a document-query pair is sum over terms t in both q and d:

The score is 0 if none of the query terms is present in the document

2.6 Vector Space Retrieval Model

The representation of a set of documents as vectors in a common vector space is known as the *vector space model* and is fundamental to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering.

Binary values are changed into count values , later it is represented as weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	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 as a binary vector

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
. . .						

Each document is now represented as a count vector

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
ANTONY	5.255	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	1
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

Document represented by tf-idf weight vector

Binary □Count □ Weight Matrix

Documents as Vectors

Each document is now represented as a real-valued vector of tf-idf weights ∈ R|V|. So we have a |V|-dimensional real-valued 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 web search engines. Each vector is very sparse - most entries are zero.

To represent the document as vector , We can consider each

document as □ a vector

each term of doc □ one component of vector

weight for each component is given by TFIDF(t, d) = TF(t, d) x IDF(t)

Queries as Vectors

Key idea 1: Represent queries as vectors in same space

Key idea 2: Rank documents according to proximity to query in this space proximity =

similarity of vectors

proximity ≈ inverse of distance

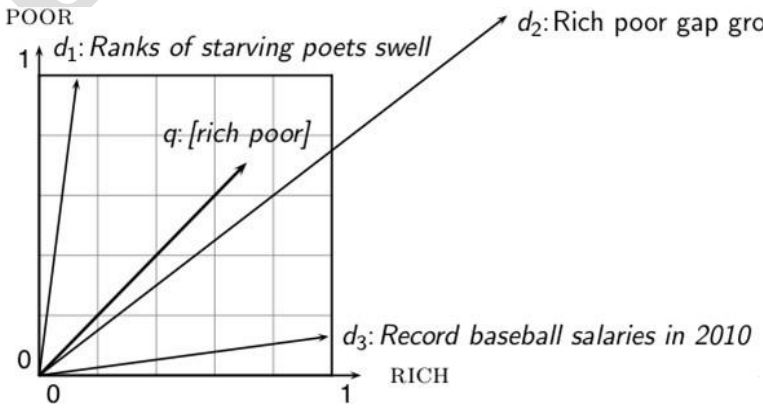
Get away from Boolean model , Rank more relevant documents higher than less relevant documents

Formalizing Vector Space Proximity

We start by calculating Euclidean distance:

$$\left\| q - d_n \right\|_2$$

Euclidean distance is a bad idea . . . because Euclidean distance is large for vectors of different lengths



The Euclidean distance of and is large although the distribution of terms in the query *q* and the distribution of terms in the document *d*2 are very similar.

Use Angle Instead of Distance

Consider an experiment where a document d and append it to itself. Call this document d" . "Semantically" d and d" have the same content , The Euclidean distance between the two documents can be quite large.
The angle between the two documents is 0, corresponding to maximal Similarity.

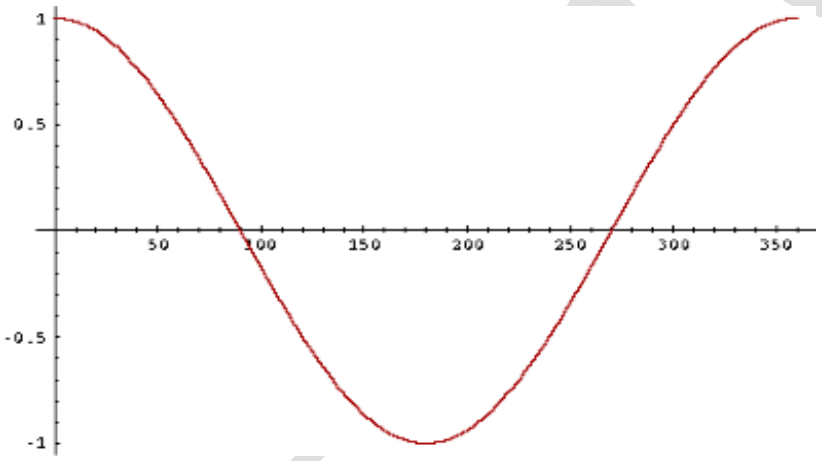
Key idea:

- Length unimportant
- Rank documents according to angle from query

Problems with Angle

Angles expensive to compute like arctan, Find a computationally cheaper, equivalent measure and Give same ranking order ! monotonically increasing/decreasing with angle

Cosine More Efficient Than Angle



Cosine is a monotonically decreasing function of the angle for the interval [0°, 180°]

Length Normalization / How do we compute the cosine

Computing cosine similarity involves length normalizing document and query vectors L2 norm:

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

Dividing a vector by its L2 norm makes it a unit (length) vector (on surface of unit hypersphere). This maps vectors onto the unit sphere . . .

As a result, longer documents and shorter documents have weights of the same order of magnitude. Effect on the two documents d and d′ (d appended to itself) from earlier example : they have identical vectors after length-normalization.

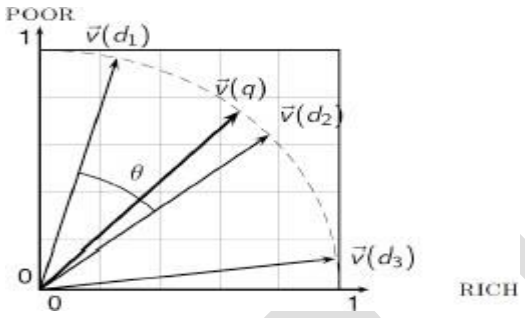
2.7 Cosine Similarity

Measure the similarity between document and the query using the cosine of the vector representations

dot / scalar / inner product

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}}{\|\vec{q}\|_2} \bullet \frac{\vec{d}}{\|\vec{d}\|_2} = \frac{\vec{q} \bullet \vec{d}}{\|\vec{q}\|_2 \|\vec{d}\|_2} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

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 q and d = the cosine of the angle between q and d.



In reality:

- Length-normalize when document added to index:

$$\vec{d} \leftarrow \frac{\vec{d}}{\|\vec{d}\|_2}$$

- Length-normalize query:

$$\vec{q} \leftarrow \frac{\vec{q}}{\|\vec{q}\|_2}$$

For normalized vectors, the cosine is equivalent to the dot product or scalar product:

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

- (if $\vec{q}$ and $\vec{d}$ are length-normalized).

Example1: To find similarity between 3 novels

Three novels are taken for discussion , namely

- SaS: Sense and Sensibility
- PaP: Pride and Prejudice
- WH: Wuthering Heights?

To find how similar are the novels using Term frequency tf_t , values are

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

Log frequency weights :

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

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

After length normalization

$$\vec{d} = [w_{1,d}, \ldots, w_{4,d}], \quad \vec{d} \leftarrow \frac{\vec{d}}{\|\vec{d}\|_2}$$

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 ≈ 0.94
cos(SaS,WH) ≈0.79
cos(PaP,WH) ≈ 0.69

cos(SAS,PaP) > cos(*,WH)

Computing Cosine Scores

```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5     for each pair(d, tf_{t,d}) in postings list
 6     do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

Example 2:

We often use different weightings for queries and documents.

Notation : ddd.qqq
Example : lnc.ltn

Document : logarithmic tf, no df weighting, cosine normalization Query
: logarithmic tf, idf, no normalization

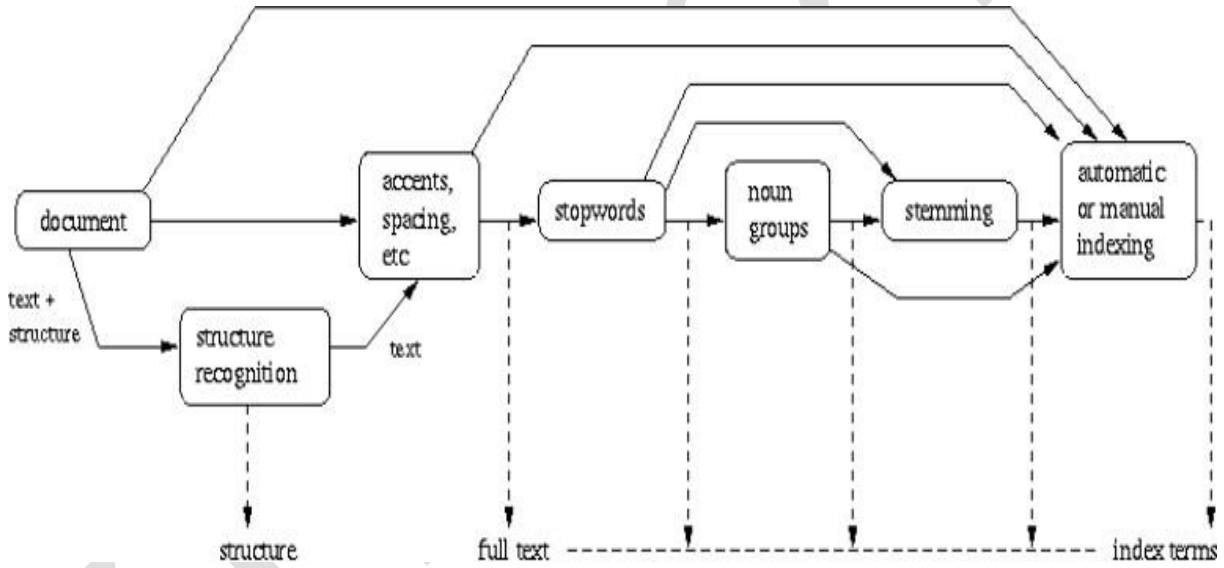
Example query: “best car insurance”
Example document: “car insurance auto insurance”
N=10,000,000

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Final similarity score between query and document:
 $wqi \cdot wdi = 0 + 0 + 1.04 + 2.04 = 3.08$

2.8 Preprocessing (from Manning TextBook)

We need to deal with format and language of each document. What format is it in? pdf, word, excel, html etc.
What language is it in?
What character set is in use? Each of these is a classification problem.



Format/Language: Complications

- A single index usually contains terms of several languages.
  - Sometimes a document or its components contain multiple languages/formats.
  - French email with Spanish pdf attachment
- What is the document unit for indexing?
  - A file?
  - An email?
  - An email with 5 attachments?
  - A group of files (ppt or latex in HTML)?
  - Upshot: Answering the question “what is a document?” is not trivial and requires some design decisions.

Determining the vocabulary of terms

Word – A delimited string of characters as it appears in the text.
Term – A "normalized" word (case, morphology, spelling etc); an equivalence class of words.
Token – An instance of a word or term occurring in a document.
Type – The same as a term in most cases: an equivalence class of tokens.

1) Tokenization:

Task of splitting the document into pieces called tokens.

- Input: Friends, Romans, countrymen. | So let it be with Caesar . . .
- Output: friend | roman | countryman | so . . .

Tokenization problems: One word or two? (or several)

Ex:

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver

Numbers

- 3/20/91 □date
- 20/3/91
- Mar 20, 1991
- B-52
- 100.2.86.144 □IP address
- (800) 234-2333 □ Phone Number
- 800.234.2333

2) Normalization

- Need to "normalize" terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
- Alternatively: do asymmetric expansion
  - window → window, windows
  - windows → Windows, windows
  - Windows (no expansion)
- More powerful, but less efficient

Case Folding-

Reduce all letters to lower case
Possible exceptions: capitalized words in mid-sentence MIT vs.
mit
Fed vs. fed
It"s often best to lowercase everything since users will use lowercase regardless of correct capitalization.

- Normalization and language detection interact.
  - *PETER WILL NICHT MIT.* → MIT = mit

- *He got his PhD from MIT.* → MIT ≠ mit

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Most important criterion: How are users likely to write their queries for these words?. Even in languages that standardly have accents, users often do not type them.

- Stop words

3) Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. "King of Denmark"
- Most web search engines index stop words

4) Lemmatization & Stemming

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car's, cars'* → *car*
- Example: *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).
- Inflectional morphology (*cutting* → *cut)* vs. derivational morphology (*destruction* → *destroy)*

- Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what "principled" lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Example : automate, automatic, automation all reduce to automat

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final *ement* if what remains is longer than 1 character
  - replacement → replac
  - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

- Porter stemmer: A few rules

Rule	Example
SSES → SS	caresses → caress
IES → I	ponies → poni

SS → SS S →	caress → caress cats → cat

Other Three stemmers

1) *Sample text:*
2) *Lovins stemmer:*
3) *Paice stemmer:*

Document Preprocessing (Refer Diagram)
Document pre-processing is the process of incorporating a new document into an information retrieval system. The goal is to Represent the document efficiently in terms of both space (for storing the document) and time (for processing retrieval requests) requirements. Maintain good retrieval performance (precision and recall). Document pre-processing is a complex process that leads to the representation of each document by a set of index terms. Logical view of a document is given below

Document pre-processing includes 5 stages:
1. Lexical analysis
2. Stopword elimination
3. Stemming
4. Index-term selection
5. Construction of thesauri

Lexical analysis
Objective: Determine the words of the document. Lexical analysis separates the input alphabet into

- Word characters (e.g., the letters a-z)
- Word separators (e.g., space, newline, tab) The

following decisions may have impact on retrieval

- Digits: Used to be ignored, but the trend now is to identify numbers (e.g., telephone numbers) and mixed strings as words.
- Punctuation marks: Usually treated as word separators.
- Hyphens: Should we interpret "pre-processing" as "pre processing" or as "preprocessing"?
- Letter case: Often ignored, but then a search for "First Bank" (a specific bank) would retrieve a document with the phrase "Bank of America was the first bank to offer its customers…"

Stopword Elimination
Objective: Filter out words that occur in most of the documents.
Such words have no value for retrieval purposes , These words are referred to as stopwords.
They include

- Articles (a, an, the, …)
- Prepositions (in, on, of, …)
- Conjunctions (and, or, but, if, …)
- Pronouns (I, you, them, it…)
- Possibly some verbs, nouns, adverbs, adjectives (make, thing, similar, …)
- A typical stopword list may include several hundred words.

the 100 most frequent words add-up to about 50% of the words in a document. Hence, stopword elimination improves the size of the indexing structures

Stemming

Objective: Replace all the variants of a word with the single stem of the word. Variants include plurals, gerund forms (ing-form), third person suffixes, past tense suffixes, etc.

Example: connect: connects, connected, connecting, connection,… All have similar semantics and relate to a single concept.

In parallel, stemming must be performed on the user query.

Stemming improves Storage and search efficiency: less terms are stored. Recall:

without stemming a query about "connection", matches only documents that have "connection".

With stemming, the query is about "connect" and matches in addition documents that originally had "connects" , "connected" , "connecting", etc.

However, stemming may hurt precision, because users can no longer target just a particular form.

Stemming may be performed using

- Algorithms that stripe of suffixes according to substitution rules.
- Large dictionaries, that provide the stem of each word.

Index term selection (indexing)

Objective: Increase efficiency by extracting from the resulting document a selected set of terms to be used for indexing the document.

If full text representation is adopted then all words are used for indexing.

Indexing is a critical process: User's ability to find documents on a particular subject is limited by the indexing process having created index terms for this subject.

Index can be done manually or automatically.

Historically, manual indexing was performed by professional indexers associated with library organizations. However, automatic indexing is more common now

Relative advantages of manual indexing:

1. Ability to perform abstractions (conclude what the subject is) and determine additional related terms,
2. Ability to judge the value of concepts.

Relative advantages of automatic indexing:

1. Reduced cost: Once initial hardware cost is amortized, operational cost is cheaper than wages for human indexers.
2. Reduced processing time
3. Improved consistency.
4. Controlled vocabulary: Index terms must be selected from a predefined set of terms (the domain of the index). Use of a controlled vocabulary helps standardize the choice of terms. Searching is improved, because users know the vocabulary being used. Thesauri can compensate for lack of controlled vocabularies.
5. Index exhaustivity: the extent to which concepts are indexed. Should we index only the most important concepts, or also more minor concepts?
6. Index specificity: the preciseness of the index term used. Should we use general indexing terms or more specific terms? Should we use the term "computer", "personal computer", or "Gateway E-3400"?
7. Main effect: High exhaustivity improves recall (decreases precision). High specificity improves precision (decreases recall).

8. Related issues: Index title and abstract only, or the entire document? Should index terms be weighted?

Reducing the size of the index: Recall that articles, prepositions, conjunctions, pronouns have already been removed through a stopword list. Recall that the 100 most frequent words account for 50% of all word occurrences. Words that are very infrequent (occur only a few times in a collection) are often removed, under the assumption that they would probably not be in the user"s vocabulary. Reduction not based on probabilistic arguments: Nouns are often preferred over verbs, adjectives, or adverbs.

Indexing Types : It may also assign weights to terms.

1. Non-weighted indexing: No attempt to determine the value of the different terms assigned to a document. Not possible to distinguish between major topics and casual references. All retrieved documents are equal in value. Typical of commercial systems through the 1980s.
2. Weighted indexing: Attempt made to place a value on each term as a description of the document. This value is related to the frequency of occurrence of the term in the document (higher is better), but also to the number of collection documents that use this term (lower is better). Query weights and document weights are combined to a value describing the likelihood that a document matches a query

Thesauri

Objective: Standardize the index terms that were selected. In its simplest form a thesaurus is A list of "important" words (concepts). For each word, an associated list of synonyms. A thesaurus may be generic (cover all of English) or concentrate on a particular domain of knowledge. The role of a thesaurus in information retrieval is to

- Provide a standard vocabulary for indexing.
- Help users locate proper query terms.
- Provide hierarchies for automatic broadening or narrowing of queries.

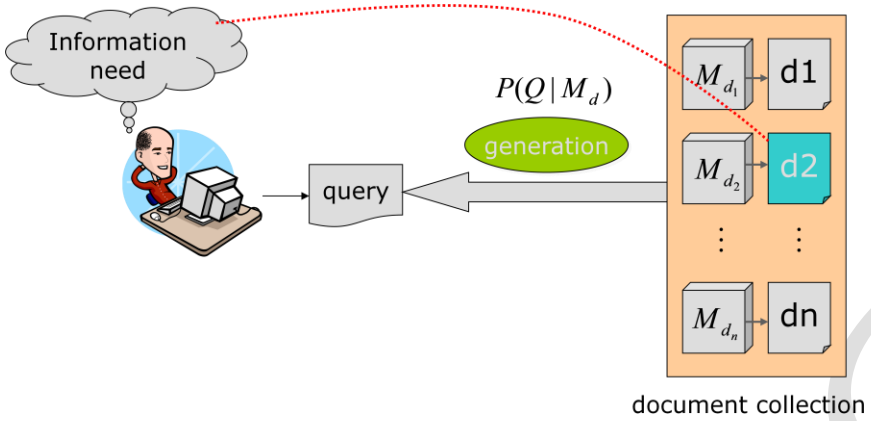
Here, our interest is to provide a standard vocabulary (a controlled vocabulary). This is final stage, where each indexing term is replaced by the concept that defines its thesaurus class.

2.9 Language models

IR approaches

1. Boolean retrieval - Boolean constrains of term occurrences in documents , no ranking
2. Vector space model - Queries and vectors are represented as vectors in a high dimensional space, Notions of similarity (cosine similarity) implying ranking
3. Probabilistic model - Rank documents by the probability P(R|d,q) , Estimate P(R|d,q) using relevance feedback technique
4. Language model approach - A document is a good match to a query, if the document model is likely to generate the query i.e If document contains query words often

Ex:



A language model is a probability distribution over sequences of words. Given such a sequence, say of length m, it assigns a probability $P(w_1, \ldots, w_m)$ to the whole sequence. These distributions can be used to predict the likelihood that the next token in the sequence is a given word . These probability distributions are called language models. It is useful in many natural language processing applications.

- Ex: part-of-speech tagging, speech recognition, machine translation, and information retrieval

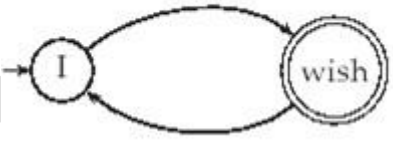
In speech recognition, the computer tries to match sounds with word sequences. The language model provides context to distinguish between words and phrases that sound similar. For example, in American English, the phrases "recognize speech" and "wreck a nice beach" are pronounced almost the same but mean very different things. These ambiguities are easier to resolve when evidence from the language model is incorporated with the pronunciation model and the acoustic model.

A language model for IR is composed of the following components

- A set of document language models, one per document dj of the collection
- A probability distribution function that allows estimating the likelihood that a document language model Mj generates each of the query terms
- A ranking function that combines these generating probabilities for the query terms into a rank of document dj with regard to the query

Traditional language model

The traditional language model uses Finite automata and it is a Generative model.



I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish
...

CANNOT GENERATE: wish I wish

This diagram shows a simple finite automaton and some of the strings in the language it generates. → shows the start state of the automaton and a double circle indicates a (possible) finishing state. For example, the finite automaton

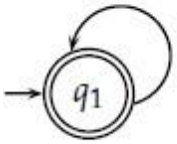
shown can generate strings that include the examples shown. The full set of strings that can be generated is called the *language* of the automaton.

Definition of language model

Each node has a probability distribution over generating different terms.A *language model* is a function that puts a probability measure over strings drawn from some vocabulary. That is, for a language model M over an alphabet Σ:

$$\sum_{s \in \Sigma^*} P(s) = 1$$

Language model example : unigram language model



$P(\text{STOP}|q_1) = 0.2$

state emission probabilities

the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
…	…

Probability that some text (e.g. a query) was generated by the model:

P(frog said that toad likes frog) = 0.01 x 0.03 x 0.04 x 0.01 x 0.02 x 0.01
(We ignore continue/stop probabilities assuming they are fixed for all queries)

A one-state finite automaton that acts as a unigram language model. We show a partial specification of the state emission probabilities. If instead each node has a probability distribution over generating different terms, we have a language model. The notion of a language model is inherently probabilistic, it places a probability distribution over any sequence of words.

Example 2: Query likelihood

Language models used in information retrieval is the query likelihood model. Here a separate language model is associated with each document in a collection. Documents are ranked based on the probability of the query Q in the document's language model $P(Q \mid M_d)$.

s	frog	said	that	toad	likes	that	dog
M1	0.01	0.03	0.04	0.01	0.02	0.04	0.005
M2	0.0002	0.03	0.04	0.0001	0.04	0.04	0.01

Query (q) = frog likes toad
P(q | M1) = 0.01 x 0.02 x 0.01

P(q | M2) = 0.0002 x 0.04 x 0.0001
P(q|M1) > P(q|M2)
=> **M1** is more likely to generate query q

One simple language model is equivalent to a probabilistic finite automaton consisting of just a single node with a single probability distribution over producing different terms ,
so that $\sum_{t \in V} P(t) = 1$. After generating each word, we decide whether to stop or to loop around and then produce another word, and so the model also requires a probability of stopping in the finishing state.

Types of language models

To build probabilities over sequences of terms , We can always use the chain rule to decompose the probability of a sequence of events into the probability of each successive event conditioned on earlier events:

$$P(t_1 t_2 t_3 t_4) = P(t_1)P(t_2|t_1)P(t_3|t_1 t_2)P(t_4|t_1 t_2 t_3)$$

1) Unigram *language model*
2) Bigram language models
3) N-gram Language Models

1) Unigram *language model* :

The simplest form of language model simply throws away all conditioning context, and estimates each term independently. Such a model is called a *unigram language model* :

$$P_{uni}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2)P(t_3)P(t_4)$$

A unigram model used in information retrieval can be treated as the combination of several one-state finite automata.

It splits the probabilities of different terms in a context, In this model, the probability to hit each word all depends on its own, so we only have one-state finite automata as units. For each automaton, we only have one way to hit its only state, assigned with one probability. Viewing from the whole model, the sum of all the one-state-hitting probabilities should be 1. Followed is an illustration of a unigram model of a document.

$$\sum_{\text{term in doc}} P(\text{term}) = 1$$

The probability generated for a specific query is calculated as

$$P(\text{query}) = \prod_{\text{term in query}} P(\text{term})$$

For Example: considering 2 language models M1 & M2 with their word emission probabilities

Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
…	…	…	…

Partial specification of two unigram language models

To find probability of a word sequence = probabilities of word (given by model) X probability of continuing / stopping after producing each word.

For example,

$$P(\text{frog said that toad likes frog}) = (0.01 \times 0.03 \times 0.04 \times 0.01 \times 0.02 \times 0.01) \times (0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.8 \times 0.2) \approx 0.000000000001573$$

the probability of a particular string/document, is usually a very small number! Here we stopped after generating *frog* the second time.
The first line of numbers are the term emission probabilities,
the second line gives the probability of continuing (0.8) or stopping (0.2) after generating each word.

To compare two models for a data set, we can calculate their *likelihood ratio* , which results from simply dividing the probability of the data according to one model by the probability of the data according to the other model.

In information retrieval contexts, unigram language models are often smoothed to avoid instances where P(term) = 0. A common approach is to generate a maximum-likelihood model for the entire collection and linearly interpolate the collection model with a maximum-likelihood model for each document to create a smoothed document model.

2) Bigram language models

There are many more complex kinds of language models, such as *bigram language models* , in which the condition is based on the previous term,

$$P_{bi}(t_1 t_2 t_3 t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3)$$

Ex: In a bigram (n = 2) language model, the probability of the sentence “I saw the red house” is

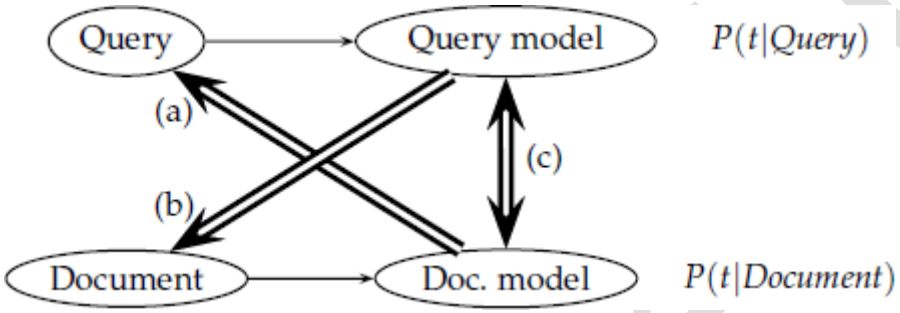
approximated as (<s> means stop)

$$P(\text{I, saw, the, red, house})$$
$$\approx P(\text{I} \mid \langle s \rangle)P(\text{saw} \mid \text{I})P(\text{the} \mid \text{saw})P(\text{red} \mid \text{the})P(\text{house} \mid \text{red})P(\langle /s \rangle \mid \text{house})$$

Most language-modeling work in IR has used unigram language models. IR is not the place where most immediately need complex language models, since IR does not directly depend on the structure of sentences to the extent that other tasks like speech recognition do. Unigram models are often sufficient to judge the topic of a text.

Three ways of developing the language modeling approach:

(a) query likelihood - Probability of generating the query text from a document language model
(b) document likelihood - Probability of generating the document text from a query language model
(c) model comparison - Comparing the language models representing the query and document topics



1) The query likelihood model in IR

The original and basic method for using language models in IR is the *query likelihood model* .
Goal : Construct from each document d in the collection a language model Md, goal is to rank documents by P(d|q), where the probability of a document is interpreted as the likelihood that it is relevant to the query.

Using Bayes rule , we have:

P(d|q) = P(q|d)P(d)/P(q)

P(q) – same for all documents □ ignored
P(d) – often treated as uniform across documents □ ignored
Could be non uniform prior based on criteria like authority, length, type, newness, and number of previous people who have read the document.

o Rank by P(q|d) □ the probability of the query q under the language model derived from d. The probability that a query would be observed as a random sample from the respective document model

Algorithm:

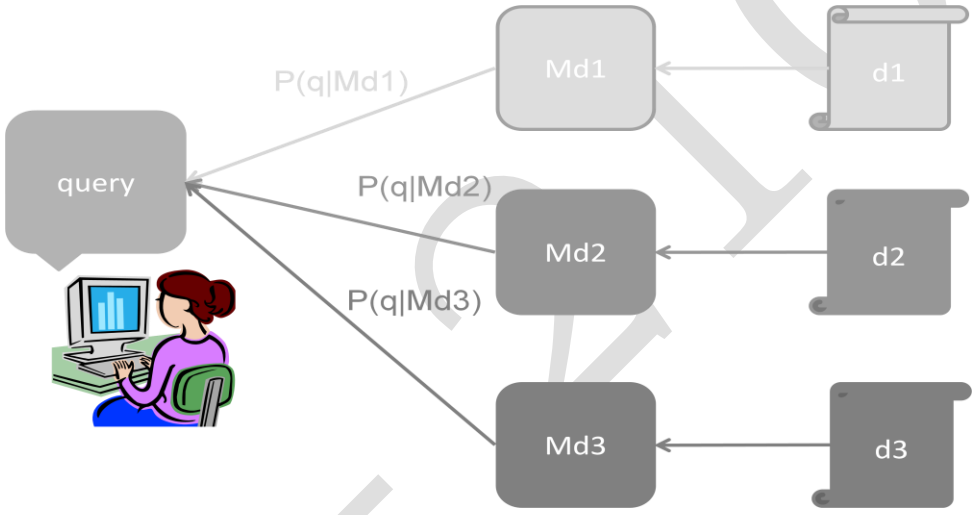
1. Infer a LM Md for each document d
2. Estimate P(q|Md)

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{mle}(t|M_d) = \prod_{t \in q} \frac{tf_{t,d}}{L_d}$$

M_d → language model of document d,
$tf_{t,d}$ → (raw) term frequency of term t in document d
L_d → number of tokens in document d.

3. Rank the documents according to these probabilities

The probability of producing the query given the LM M_d of document d using *maximum likelihood estimation* (*MLE*) and the unigram assumption.



E.g., P(q|Md3) > P(q|Md1) > P(q|Md2) □ d3 is first, d1 is second, d2 is third

Sparse Data Problem

- The classic problem with using language models is terms appear very sparsely in documents , Some words don"t appear in the document - a term missing from a document . In such cases
  - In particular, some of the query terms
    □ P(q|Md) = 0 ; zero probability problem
    We get Conjunctive semantics i.e documents will only give a query non-zero probability if all of the query terms appear in the document
- Occurring words are poorly estimated
  - A single documents is small training set
    □ Occurring words are over estimated - Their occurrence was partly by chance

Solution: smoothing

Smoothing

- Decreasing the estimated probability of seen events and increasing the probability of unseen events is referred to as smoothing

- The role of smoothing in this model is not only to avoid zero probabilities. The smoothing of terms actually implements major parts of the term weighting component. Thus, we need to smooth probabilities in our document language models:
  - to discount non-zero probabilities
  - to give some probability mass to unseen words.

- The probability of a non occurring term should be close to its probability to occur in the collection

$$P(t|Mc) = cf(t)/T$$

cf(t) = #occurrences of term t in the collection
T – length of the collection = sum of all document lengths

The general approach is that a non-occurring term should be possible in a query, but its probability should be somewhat close to but no more likely than would be expected by chance from the whole collection.

Smoothing Methods

Linear Interpolation (Mixer Model)	$\hat{P}(t\|M_d) = \lambda \hat{P}_{MLE}(t\|M_d) + (1-\lambda)\hat{P}_{MLE}(t\|M_C);\ 0 < \lambda < 1$ ▪ Mixes the probability from the document with the general collection frequency of the word. ▪ High value of λ: "conjunctive-like" search – tends to retrieve documents containing all query words. ▪ Low value of λ: more disjunctive, suitable for long queries ▪ Correctly setting λ is very important for good performance.
Bayesian smoothing	$\hat{P}(t\|M_d) = \frac{tf_{t,d} + \alpha \hat{P}_{MLE}(t\|M_c)}{L_d + \alpha}$
Summary, with linear interpolation	$P(d\|q) \propto P(d) \prod_{t \in q} ((1-\lambda)P(t\|M_c) + \lambda P(t\|M_d))$

In practice, log in taken from both sides of the equation to avoid multiplying any small numbers

Example1:

Question: Suppose the document collection contains two documents: d1: Xerox reports a profit but revenue is down
d2: Lucent narrows quarter loss but revenue decreases further A user submitted the query: "*revenue down*"

Rank *D1* and *D2* - Use an MLE unigram model and a linear interpolation smoothing with lambda parameter *0.5*

Solution :

$$P(q|d_1) = [(1/8+2/16)/2] \times [(1/8+1/16)/2]$$

$$= 1/8 \times 3/32 = 3/256$$

$$P(q|d_2) = [(1/8+2/16)/2] \times [(0/8+1/16)/2]$$

$$= 1/8 \times 1/32 = 1/256$$

So, the ranking is d1>d2

Example2:

Collection: *d1* and *d2*

d1 : Jackson was one of the most talented entertainers of all time

d2: Michael Jackson anointed himself King of Pop Query *q*:

Michael Jackson

Use mixture model with $\lambda = 1/2$

$P(q|d1) = [(0/11 + 1/18)/2] \cdot [(1/11 + 2/18)/2] \approx 0.003$

$P(q|d2) = [(1/7 + 1/18)/2] \cdot [(1/7 + 2/18)/2] \approx 0.013$

Ranking: $d2 > d1$

2) Document likelihood model

There are other ways to think of using the language modeling idea in IR settings , It is the probability of a query language model Mq generating the document.

Reason for creating a document likelihood model

- There is much less text available to estimate a language model based on the query text, and so the model will be worse estimated
- More dependency towards smoothing with some other language model

P(d|q) – the probability of query LM to generate document. The problem of this model is that queries are short this leads to bad model estimation. So Zhai and Lafferty 2001 suggesting to expand the query with terms taken from relevant documents in the usual way and hence update the language model

3) Model comparison

Make LM from both query and document ,Measure `how different` these LMs from each other , Uses KL divergence

KL divergence (Kullback–Leibler (KL) divergence)

An asymmetric divergence measure from information theory , which measures the how bad the probability distribution Mq is at modelingMd

$$R(d;q) = KL(M_d||M_q) = \sum_{t \in V} P(t|M_q) log \frac{P(t|M_q)}{P(t|M_d)}$$

Rank by KLD - the closer to 0 the higher is the rank

LMs vs. vector space model

- LMs have some things in common with vector space models.
- Term frequency is directed in the model.
  - But it is not scaled in LMs.
- Probabilities are inherently "length-normalized".
  - Cosine normalization does something similar for vector space.
- Mixing document and collection frequencies has an effect similar to idf.
  - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.
- LMs vs. vector space model: commonalities
  - Term frequency is directly in the model.
  - Probabilities are inherently "length-normalized".
  - Mixing document and collection frequencies has an effect similar to idf.
  - LMs vs. vector space model: differences
    - LMs: based on probability theory
  - Vector space: based on similarity, a geometric/ linear algebra notion
  - Collection frequency vs. document frequency
  - Details of term frequency, length normalization etc.

2.10 Probabilistic information retrieval

It is introduced by Roberston and Sparck Jones, 1976 and most important or original one is Binary independence retrieval (BIR) model

Idea: Given a user query q, and the ideal answer set R of the relevant documents, the problem is to specify the properties for this set

- Assumption (probabilistic principle): the probability of relevance depends on the query and document representations only; ideal answer set R should maximize the overall probability of relevance
- The probabilistic model tries to estimate the probability that the user will find the document dj relevant with ratio
  P(dj relevant to q)/P(dj nonrelevant to q)

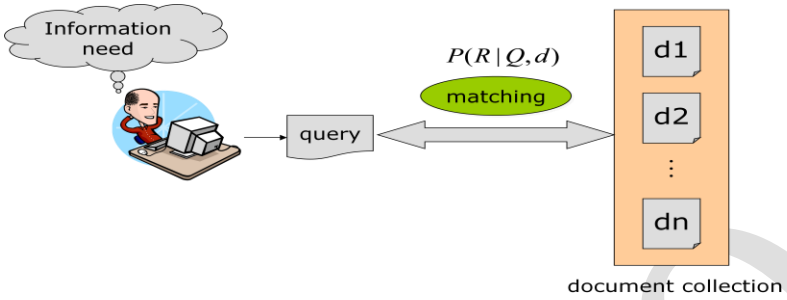
Given a query q, there exists a subset of the documents R which are relevant to q

But membership of R is uncertain (not sure) , A Probabilistic retrieval model ranks documents in decreasing order of probability of relevance to the information need: P(R | q,d_i)

Users gives with *information needs*, which they translate into *query representations*. Similarly, there are *documents*, which are converted into *document representations* . Given only a query, an IR system has an uncertain understanding of the information need. So IR is an uncertain process , Because

- Information need to query
- Documents to index terms
- Query terms and index terms mismatch

Probability theory provides a principled foundation for such reasoning under uncertainty. This model provides how likely a document is relevant to an information need.



Document can be relevant and nonrelevant document, we can estimate the probability of a term t appearing in a relevant document P(t | R=1) .

Probabilistic methods are one of the oldest but also one of the currently hottest topics in IR .

Probabilistic IR Models :

- Classical probabilistic retrieval model
  - Probability ranking principle
    - Binary Independence Model, BestMatch25 (Okapi)
- Bayesian networks for text retrieval
- Language model approach to IR

Basic probability theory

For events A , the probability of the event lies between $0 \leq P(A) \leq 1$, For 2 events A and B

- Joint probability P(*A*, *B*) of both events occurring
- Conditional probability P(*A*|*B*) of event *A* occurring given that event B has occurred
- Chain rule gives fundamental relationship between joint and conditional probabilities:

$$P(A, B) = P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

Similarly for the complement of an event P($\bar{A}$,B) :

$$P(\overline{A}, B) = P(B|\overline{A})P(\overline{A})$$

- Partition rule: if B can be divided into an exhaustive set of disjoint sub cases, then P(B) is the sum of the probabilities of the sub cases. A special case of this rule gives:

$$P(B) = P(A, B) + P(\overline{A}, B)$$

- Bayes‟ Rule for inverting conditional probabilities:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \left[\frac{P(B|A)}{\sum_{X\in\{A,\overline{A}\}} P(B|X)P(X)}\right] P(A)$$

Can be thought of as a way of updating probabilities:

- Start off with prior probability $P(A)$ (initial estimate of how likely event A is in the absence of any other information)
- Derive a posterior probability $P(A|B)$ after having seen the evidence B, based on the likelihood of B occurring in the two cases that A does or does not hold

- Odds of an event (is the ratio of the probability of an event to the probability of its complement.) provide a kind of multiplier for how probabilities change:

$$\text{Odds:} \quad O(A) = \frac{P(A)}{P(\overline{A})} = \frac{P(A)}{1 - P(A)}$$

The Document Ranking Problem

In Ranked retrieval setup, for a given collection of documents, the user issues a query, and an ordered list of documents is returned. Assume binary notion of relevance: Rd,q is a a random dichotomous variable (A categorical variable that can take on exactly two values is termed a *binary variable* or *dichotomous variable*), such that

$R_{d,q}$ = 1 if document d is relevant w.r.t query q $R_{d,q}$ = 0 otherwise

Probabilistic ranking orders documents decreasingly by their estimated probability of relevance w.r.t. query: P(R = 1|d, q)

The Probability Ranking Principle

PRP in brief

If the retrieved documents (w.r.t a query) are ranked decreasingly on their probability of relevance, then the effectiveness of the system will be the best that is obtainable

PRP in full

If [the IR] system‟s response to each [query] is a ranking of the documents [...] in order of decreasing probability of relevance to the [query], where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data.

1/0 loss :

- either returning a non relevant document or failing to return a relevant document is called as 1/0 loss.
- The goal is to return the best possible results as the top k documents, for any value of k the user chooses to examine.
- The PRP then says to simply rank all documents in decreasing order of P (R=1 | d,q) . If a set of retrieval results is to be returned, rather than an ordering, the

Bayes Optimal Decision Rule , the decision which minimizes the risk of loss, is to simply return documents that are more likely relevant than nonrelevant:

$$d \text{ is relevant iff } P(R = 1|d, q) > P(R = 0|d, q)$$

The PRP with retrieval costs

Let C_1 □ cost of not retrieving a relevant document C_0 □ cost of retrieval of a non relevant document d□ next document to be retrieved

d‟ □ not yet retrieved document Then the

Probability Ranking Principle says that

$$C_0 \cdot P(R = 0|d) - C_1 \cdot P(R = 1|d) \leq C_0 \cdot P(R = 0|d') - C_1 \cdot P(R = 1|d')$$

Such a model gives a formal framework where we can model differential costs of false positives and false negatives and even system performance issues at the modeling stage, rather than simply at the evaluation stage.

The Binary Independence Model (BIM)

This model is a traditionally used with the PRP, „Binary‟ (equivalent to Boolean) means documents and queries represented as binary term incidence vectors

E.g., document d represented by vector x = (x1, . . . , xM), where

x_t = 1 if term t occurs in d and x_t = 0

otherwise

Different documents may have the same vector representation . „Independence‟ means no association between terms (not true, but practically works - „naive‟ assumption of Naive Bayes models).

To make a probabilistic retrieval strategy precise, need to estimate how terms in documents contribute to relevance

1) Find measurable statistics (term frequency, document frequency, document length) that affect judgments about document relevance
2) Combine these statistics to estimate the probability of document relevance
3) Order documents by decreasing estimated probability of relevance P(R|d, q)
4) Assume that the relevance of each document is independent of the relevance of other documents (not true, in practice allows duplicate results)

$P(R|d, q)$ modeled using term incidence vectors as $P(R|\vec{x}, \vec{q})$

$$P(R = 1|\vec{x}, \vec{q}) = \frac{P(\vec{x}|R = 1, \vec{q})P(R = 1|\vec{q})}{P(\vec{x}|\vec{q})}$$

$$P(R = 0|\vec{x}, \vec{q}) = \frac{P(\vec{x}|R = 0, \vec{q})P(R = 0|\vec{q})}{P(\vec{x}|\vec{q})}$$

$P(\vec{x}\|R = 1, \vec{q})$ $P(\vec{x}\|R = 0, \vec{q})$	probability that if a relevant or non relevant document is retrieved, then that document‟s representation is $\vec{x}$
$P(R = 1\|\vec{q})$ $P(R = 0\|\vec{q})$	prior probability of retrieving a relevant or nonrelevant document for a query q , Estimate from percentage of

	relevant documents in the collection

Since a document is either relevant or nonrelevant to a query, we must have that:

$$P(R = 1|\vec{x}, \vec{q}) + P(R = 0|\vec{x}, \vec{q}) = 1$$

To make a probabilistic retrieval strategy precise, need to estimate how terms in documents contribute to relevance

- Find measurable statistics (term frequency, document frequency, document length) that affect judgments about document relevance
- Combine these statistics to estimate the probability $P(R|d, q)$ of document relevance

Deriving a ranking function for query terms

Given a query q, ranking documents by P(R = 1|d, q) is modeled under BIM as ranking them by $P(R = 1|\vec{x}, \vec{q})$

Easier: rank documents by their odds of relevance (gives same

$$O(R|\vec{x}, \vec{q}) = \frac{P(R = 1|\vec{x}, \vec{q})}{P(R = 0|\vec{x}, \vec{q})} = \frac{\frac{P(R=1|\vec{q})P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|\vec{q})}}{\frac{P(R=0|\vec{q})P(\vec{x}|R=0,\vec{q})}{P(\vec{x}|\vec{q})}} = \frac{P(R = 1|\vec{q})}{P(R = 0|\vec{q})} \cdot \frac{P(\vec{x}|R = 1, \vec{q})}{P(\vec{x}|R = 0, \vec{q})}$$

$\frac{P(R=1|\vec{q})}{P(R=0|\vec{q})}$ is a constant for a given query - can be ignored

It is at this point that we make the Naive Bayes conditional independence assumption that the presence or absence of a word in a document is independent of the presence or absence of any other word (given the query):

$$\frac{P(\vec{x}|R = 1, \vec{q})}{P(\vec{x}|R = 0, \vec{q})} = \prod_{t=1}^{M} \frac{P(x_t|R = 1, \vec{q})}{P(x_t|R = 0, \vec{q})}$$

So:

$$O(R|\vec{x}, \vec{q}) = O(R|\vec{q}) \cdot \prod_{t=1}^{M} \frac{P(x_t|R = 1, \vec{q})}{P(x_t|R = 0, \vec{q})}$$

Since each x_t is either 0 or 1, we can separate the terms to give:

$$O(R|\vec{x}, \vec{q}) = O(R|\vec{q}) \cdot \prod_{t:x_t=1} \frac{P(x_t = 1|R = 1, \vec{q})}{P(x_t = 1|R = 0, \vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t = 0|R = 1, \vec{q})}{P(x_t = 0|R = 0, \vec{q})}$$

Let $p_t = P(x_t = 1|R = 1,\vec{q})$ be the probability of a term appearing in a document relevant to the query,

Let $u_t = P(x_t = 1|R = 0,\vec{q})$ be the probability of a term appearing in a non relevant document. It can be contingency table :

	document	relevant ($R = 1$)	nonrelevant ($R = 0$)
Term present	$x_t = 1$	p_t	u_t
Term absent	$x_t = 0$	$1 - p_t$	$1 - u_t$

Additional simplifying assumption: terms not occurring in the query are equally likely to occur in relevant and irrelevant documents, If $q_t = 0$, then $p_t = u_t$

Now we need only to consider terms in the products that appear in the query:

$$O(R|\vec{q},\vec{x}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t}$$

The left product is over query terms found in the document and the right product is over query terms not found in the document.

Including the query terms found in the document into the right product, but simultaneously dividing through by them in the left product, so the value is unchanged

$$O(R|\vec{q},\vec{x}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} \cdot \prod_{t:q_t=1} \frac{1-p_t}{1-u_t}$$

The left product is still over query terms found in the document, but the right product is now over all query terms, hence constant for a particular query and can be ignored.

→ The only quantity that needs to be estimated to rank documents w.r.t a query is the left product, Hence the Retrieval Status Value (RSV) in this model:

$$RSV_d = \log \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

Equivalent: rank documents using the log odds ratios for the terms in the query *ct*

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{(1-p_t)} + \log \frac{1-u_t}{u_t}$$

The odds ratio is the ratio of two odds: (i) the odds of the term appearing if the document is relevant ($p_t/(1 - p_t)$), and (ii) the odds of the term appearing if the document is irrelevant ($u_t/(1 - u_t)$)

- $c_t = 0$: term has equal odds of appearing in relevant and irrelevant docs

- ct positive: higher odds to appear in relevant documents

- c_t negative: higher odds to appear in irrelevant documents
  c_t functions as a term weight. Retrieval status value for document *d*:

$$RSV_d = \sum_{x_t=q_t=1} c_t$$

- So BIM and vector space model are identical on an operational level , except that the term weights are different. In particular: we can use the same data structures (inverted index etc) for the two models.

How to compute Probability Estimate(in practice)

For each term t in a query, estimate ct in the whole collection using a contingency table of counts of documents in the collection, where df t is the number of documents that contain term t:

(I)

documents		relevant	nonrelevant	Total
Term present	$x_t = 1$	s	$\mathrm{df}_t - s$	df_t
Term absent	$x_t = 0$	$S - s$	$(N - \mathrm{df}_t) - (S - s)$	$N - \mathrm{df}_t$
	Total	S	$N - S$	N

p_t=s/S and u_t =(df_t – s)/N-S)

$$c_t = K(N, \mathrm{df}_t, S, s) = \log \frac{s/(S-s)}{(\mathrm{df}_t - s)/((N - \mathrm{df}_t) - (S-s))}$$

Avoiding Zeroes :

If any of the counts is a zero, then the term weight is not well-defined. Maximum likelihood estimates do not work for rare events.

To avoid zeros: add 0.5 to each count (expected likelihood estimation = ELE) For example,

use S − s + 0.5 in formula for S – s

Assuming that relevant documents are a very small percentage of the collection, approximate statistics for irrelevant documents by statistics from the whole collection

Hence, ut (the probability of term occurrence in irrelevant documents for a query) is dft/N and

$$\log[(1-u_t)/u_t] = \log[(N - \mathrm{df}_t)/\mathrm{df}_t] \approx \log N/\mathrm{df}_t$$

The above approximation cannot easily be extended to

How different are vector space and BIM?

They are not that different. In either case you build an information retrieval scheme in the exact same way.

For probabilistic IR, at the end, you score queries not by cosine similarity and tf- idf in a vector space, but by a slightly different formula motivated by probability theory. Next to add term frequency and length normalization to the probabilistic model.

2.11 Latent semantic indexing

Classic IR might lead to poor retrieval due to:

Unrelated documents might be included in the answer set

Relevant documents that do not contain at least one index term are not retrieved **Reasoning:** retrieval based on index terms is vague and noisy , The user information need is more related to concepts and ideas than to index terms.

Key Idea :

- The process of matching documents to a given query could be concept matching instead of index term matching
- A document that shares concepts with another document known to be relevant might be of interest
- The key idea is to map documents and queries into a lower dimensional space (i.e., composed of higher level concepts which are in fewer number than the index terms), Retrieval in this reduced concept space might be superior to retrieval in the space of index terms
- LSI increases recall and hurts precision.

Definition : Latent semantic indexing (LSI) is an indexing and retrieval method that uses a mathematical technique called singular value decomposition (SVD) to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings.

Problems with Lexical Semantics

- Ambiguity and association in natural language
  - **Polysemy**: Words often have a **multitude of meanings** and different types of usage (more severe in very heterogeneous collections).
  - The vector space model is unable to discriminate between different meanings of the same word.

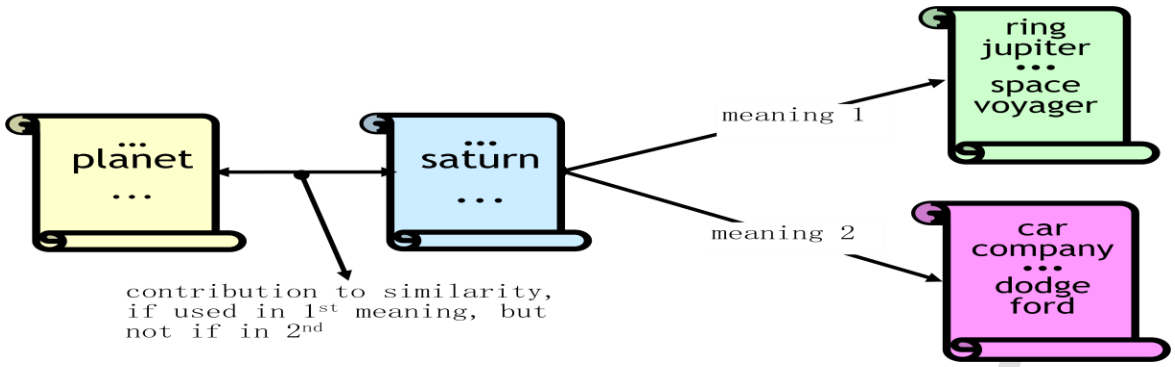
$$\mathrm{sim}_{\mathrm{true}}(d, q) < \cos(\angle(\vec{d}, \vec{q}))$$

  - Synonymy: Different terms may have an identical or a similar meaning (weaker: words indicating the same topic).
  - No associations between words are made in the vector space representation.

$$\mathrm{sim}_{\mathrm{true}}(d, q) > \cos(\angle(\vec{d}, \vec{q}))$$

  Polysemy and Context

- Document similarity on single word level: polysemy and context
- LSI Perform a **low-rank approximation** of **document-term matrix**) .



In LSI

- Map documents (and terms) to a **low-dimensional** representation.
- Design a mapping such that the low-dimensional space reflects **semantic associations** (latent semantic space).
- Compute document similarity based on the **inner product** in this **latent semantic space**
- We will decompose the term-document matrix into a product of matrices.
- The particular decomposition we"ll use: singular value decomposition (SVD).
- SVD: $C = U\Sigma V^T$ (where C = term-document matrix)
- We will then use the SVD to compute a new, improved term-document matrix C'.
- We"ll get better similarity values out of C' (compared to C).
- Using SVD for this purpose is called latent semantic indexing or LSI.

Steps:

1. decompose the term-document matrix into a product of matrices.The particular decomposition is called as singular value decomposition (SVD).

   SVD: $C = U\Sigma V^T$ (where C = term-document matrix)

   - The term matrix U – consists of one (row) vector for each term
   - The document matrix V^T – consists of one (column) vector for each document
   - The singular value matrix Σ – diagonal matrix with singular values, reflecting importance of each dimension
2. Use the SVD to compute a new, improved term-document matrix C' by reducing the space which gives better similarity values compared to C.
3. Map the query into the reduced space $\vec{q}_2^T = \Sigma_2^{-1} U_2^T \vec{q}^T.$
4. This follows from: $C_2 = U\Sigma_2 V^T \Rightarrow \Sigma_2^{-1} U^T C = V_2^T$
5. Compute similarity of q2 with all reduced documents in V2.
6. Output ranked list of documents

Example:

This is a standard term-document matrix.

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

The matrix U - consists of one (row) vector for each term

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

One row per term, one column per min(M,N) where M is the number of terms and N is the number of documents. This is an orthonormal matrix:

(i) Row vectors have unit length. (ii) Any two distinct row vectors are orthogonal to each other. Think of the dimensions as "semantic" dimensions that capture distinct topics like politics, sports, economics. Each number uij in the matrix indicates how strongly related term i is to the topic represented by semantic dimension j .

This is a standard term-document matrix. Actually, we use a non-weighted matrix here to simplify the example.

The matrix Σ

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

This is a square, diagonal matrix of dimensionality min(M,N) × min(M,N). The diagonal

consists of the singular values of *C*. The magnitude of the singular value measures the importance of the corresponding semantic dimension. We‟ll make use of this by mitting unimportant dimensions.

The matrix V^T

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

One column per document, one row per min(*M*,*N*) where *M* is the number of terms and *N* is the number of documents. Again: This is an orthonormal matrix: (i) Column vectors have unit length. (ii) Any two distinct column vectors are orthogonal to each other. These are again the semantic dimensions from the term matrix *U* that capture distinct topics like politics, sports, economics. Each number *vij* in the matrix indicates how strongly related document *i* is to the topic represented by semantic dimension *j* .

Reducing the dimensionality to 2

U	1	2	3	4	5
ship	−0.44	−0.30	0.00	0.00	0.00
boat	−0.13	−0.33	0.00	0.00	0.00
ocean	−0.48	−0.51	0.00	0.00	0.00
wood	−0.70	0.35	0.00	0.00	0.00
tree	−0.26	0.65	0.00	0.00	0.00

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00

Actually, we only zero out singular values in Σ. This has the effect of setting the corresponding dimensions in U and V^T to zero when computing the product $C = U\Sigma V^T$.

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

=

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

×

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

×

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

Why the reduced matrix is "better" ¿

Original matrix C vs. reduced $C2 = U\Sigma 2V^T$

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

We can view $C2$ as a two-dimensional representation of the matrix. We have performed a dimensionality reduction to two dimensions.

Similarity of d2 and d3 in the original space: 0. Similarity

of d2 und d3 in the reduced space:

0.52 * 0.28 + 0.36 * 0.16 + 0.72 * 0.36 + 0.12 * 0.20 + - 0.39 * - 0.08 ≈ 0.52

Why we use LSI in information retrieval

LSI takes documents that are semantically similar (= talk about the same topics), but are not similar in the vector space (because they use different words) and re- represents them in a reduced vector space in which they have higher similarity.
Thus, LSI addresses the problems of synonymy and semantic related ness. Standard vector space: Synonyms contribute nothing to document similarity.

How LSI addresses synonymy and semantic relatedness

The dimensionality reduction forces us to omit a lot of "detail".
We have to map different words (= different dimensions of the full space) to the same dimension in the reduced space.
The "cost" of mapping synonyms to the same dimension is much less than the cost of collapsing unrelated words.
SVD selects the "least costly" mapping ,Thus, it will map synonyms to the same dimension. But it will avoid doing that for unrelated words.

LSI: Comparison to other approaches

- Relevance feedback and query expansion are used to increase recall in information retrieval – if query and documents have (in the extreme case) no terms in common.
- LSI increases recall and hurts precision.
- Thus, it addresses the same problems as (pseudo) relevance feedback and query expansion . . .

2.12 Relevance feedback and query expansion

Interactive relevance feedback: improve initial retrieval results by telling the IR system which docs are relevant / nonrelevant . Best known relevance feedback method: Rocchio feedback

Query expansion: improve retrieval results by adding synonyms / related terms to the query. Sources for related terms: Manual thesauri, automatic thesauri, query logs.Two ways of improving recall: relevance feedback and query expansion

Synonymy : In most collections, the same concept may be referred to using different words. This is called as *synonymy* ,

As an example consider query *q*: [aircraft] . . .
. . . and document *d* containing "plane", but not containing "aircraft" A simple IR system will not return *d* for *q*.

Even if *d* is the most relevant document for *q*! We want
to change this:
Return relevant documents even if there is no term match with the (original) query

Recall:

- increasing the number of relevant documents returned to user"
- This may actually decrease recall on some measures, e.g., when expanding "jaguar" with "panthera"
  - . . .which eliminates some relevant documents, but increases relevant documents returned on top pages

Options for improving recall :

1) Global methods are techniques for expanding or reformulating query terms independent of the query and results returned from it , so that changes in the query wording will cause the new query to match other semantically similar terms. Global methods include:

   - Query expansion/reformulation with a thesaurus or WordNet
   - Query expansion via automatic thesaurus generation
   - Techniques like spelling correction

2) Local methods adjust a query relative to the documents that initially appear to match the query. The basic methods are:
   - Relevance feedback
   - Pseudo relevance feedback, also known as Blind relevance feedback
   - (Global) indirect relevance feedback

Google examples for query expansion

One that works well - Ex : *~flights –flight*

One that doesn"t work so well – Ex : *~hospitals -hospital*

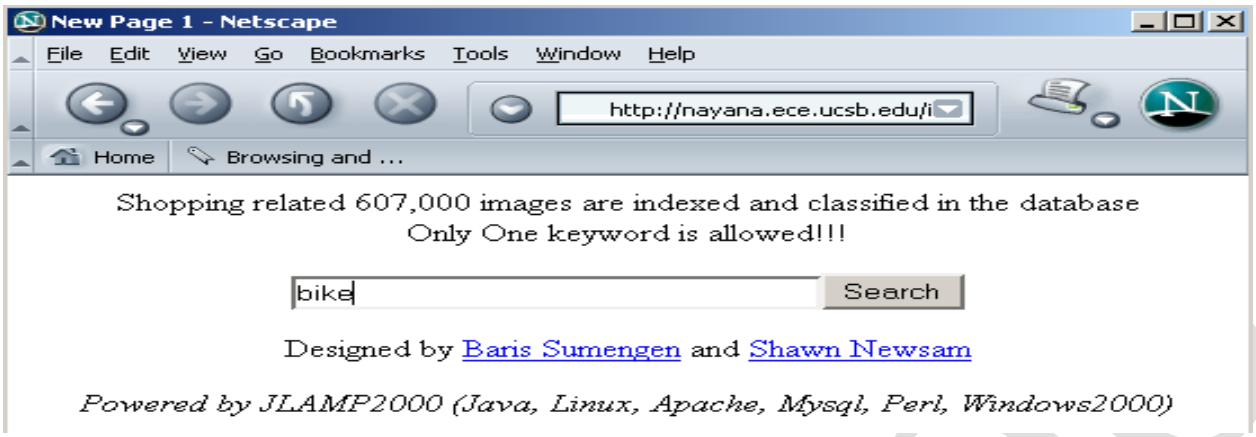
Relevance feedback and pseudo relevance feedback

The idea of *relevance feedback* ( ) is to involve the user in the retrieval process so as to improve the final result set. In particular, the user gives feedback on the relevance of documents in an initial set of results. The basic procedure is:

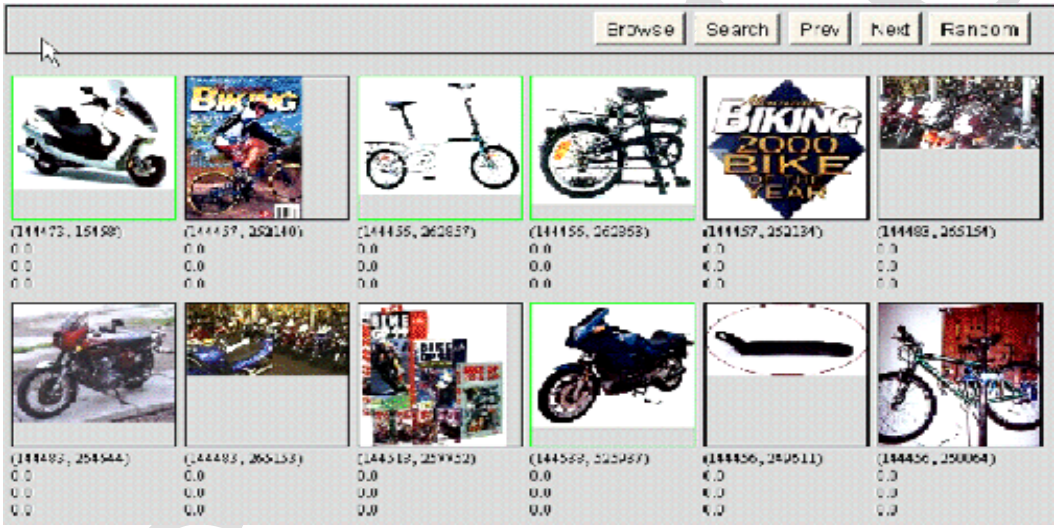
- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant, some as nonrelevant.
- Search engine computes a new representation of the information need. Hope: better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall.

We can iterate this: several rounds of relevance feedback. We will use the term ad hoc retrieval to refer to regular retrieval without relevance feedback. We will now look at Two different examples of relevance feedback that highlight different aspects of the process.

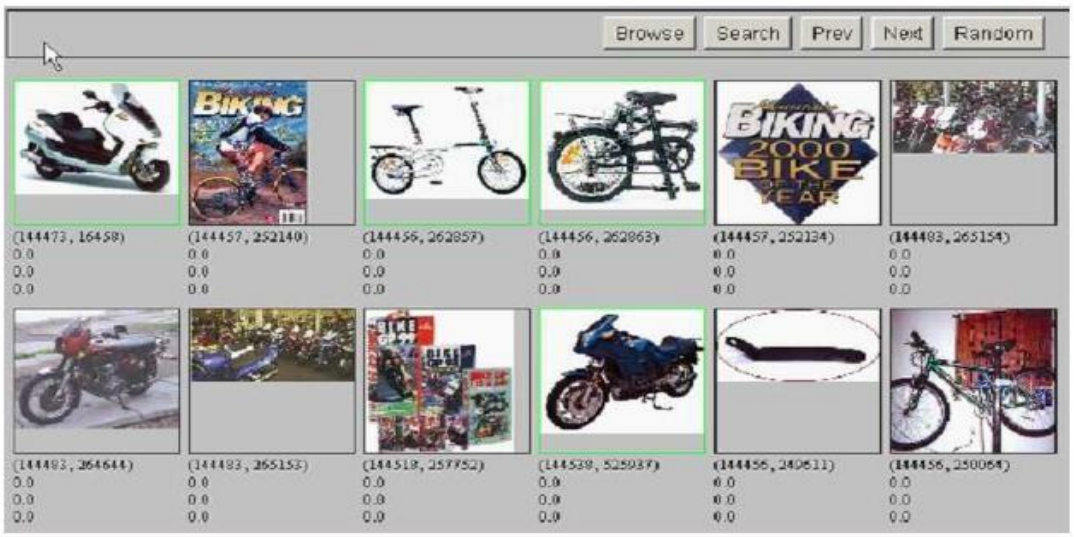
Example1 : Image search engine http://nayana.ece.ucsb.edu/imsearch/imsearch.html



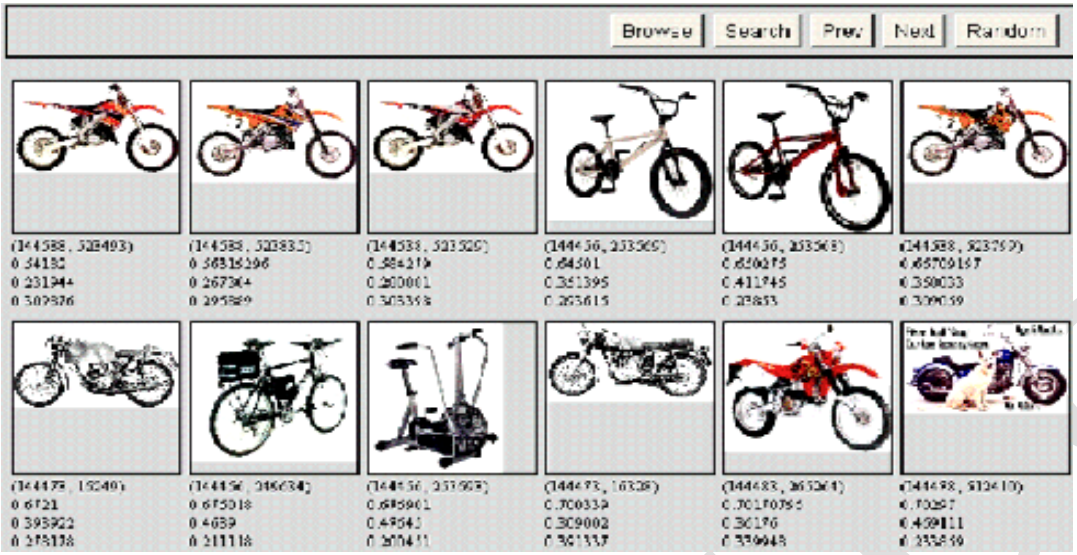
Result of initial Query



User feedback: Select what is relevant



After Relevance Feedback



Example 2: A real (non-image) example - shows a textual IR example where the user wishes to find out about new applications of space satellites.

Initial query: [new space satellite applications] Results for initial query: (*r* = rank)

	r		
+	1	0.539	NASA Hasn‟t Scrapped Imaging Spectrometer
+	2	0.533	NASA Scratches Environment Gear From Satellite Plan
	3	0.528	Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes
	4	0.526	NASA Satellite Project Accomplishes Incredible Feat: Staying Within Budget
	5	0.525	Scientist Who Exposed Global Warming Proposes Satellites for Climate Research
	6	0.524	Report Provides Support for the Critics Of Using Big Satellites to Study Climate
	7	0.516	Arianespace Receives Satellite Launch Pact From Telesat Canada
+	8	0.509	Telecommunications Tale of Two Companies User then

marks relevant documents with “+”.

Expanded query after relevance feedback

query: [new space satellite applications]

2.074	new	15.106	space
30.816	satellite	5.660	application
5.991	nasa	5.196	eos
4.196	launch	3.972	aster
3.516	instrument	3.446	arianespace
3.004	bundespost	2.806	ss
2.790	rocket	2.053	scientist
2.003	broadcast	1.172	earth
0.836	oil	0.646	measure

Results for expanded query

	r		
*	1	0.513	NASA Scratches Environment Gear From Satellite Plan
*	2	0.500	NASA Hasn‟t Scrapped Imaging Spectrometer
	3	0.493	When the Pentagon Launches a Secret Satellite, Space Sleuths Do Some Spy Work of Their Own
	4	0.493	NASA Uses „Warm‟ Superconductors For Fast Circuit
*	5	0.492	Telecommunications Tale of Two Companies
	6	0.491	Soviets May Adapt Parts of SS-20 Missile For Commercial Use
	7	0.490	Gaping Gap: Pentagon Lags in Race To Match the Soviets In Rocket Launchers
	8	0.490	Rescue of Satellite By Space Agency To Cost $90 Million

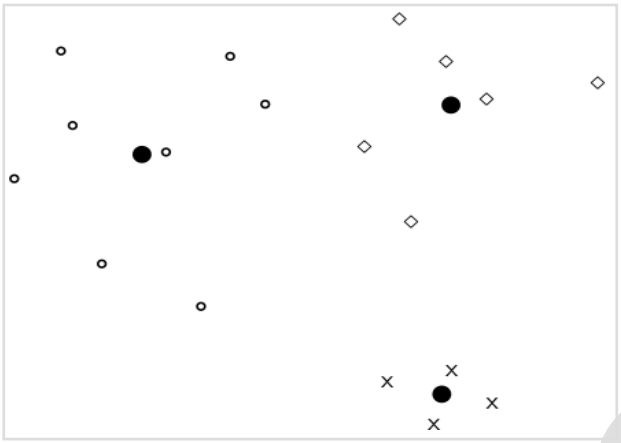
Key concept for relevance feedback: Centroid

The centroid is the center of mass of a set of points. Recall that we represent documents as points in a high-dimensional space. Thus: we can compute centroids of documents.
Definition:

$$\vec{\mu}(D) = \frac{1}{|D|} \sum_{d \in D} \vec{v}(d)$$

where D is a set of documents and $\vec{v}(d) = \vec{d}$ is the vector we use to represent document *d*.

Example:



The Rocchio algorithm for relevance feedback

- The Rocchio‟ algorithm implements relevance feedback in the vector space model.
- Rocchio‟ choosesthequery $\vec{q}_{opt}$ that maximizes

$$\vec{q}_{opt} = \arg\max_{\vec{q}}[\text{sim}(\vec{q}, \mu(D_r)) - \text{sim}(\vec{q}, \mu(D_{nr}))]$$

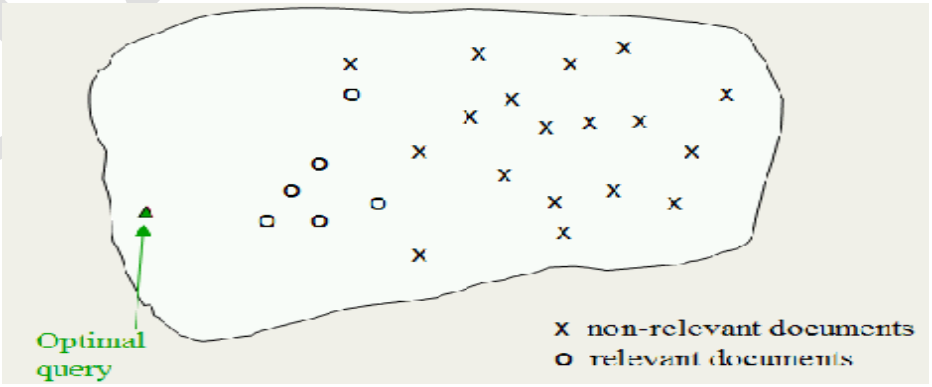
- D_r : set of relevant docs; D_{nr} : set of nonrelevantdocs
- Intent: ~qopt is the vector that separates relevant and nonrelevant docs maximally.
- Making some additional assumptions, we can rewrite as:

$$\vec{q}_{opt} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$$

- The optimal query vector is:

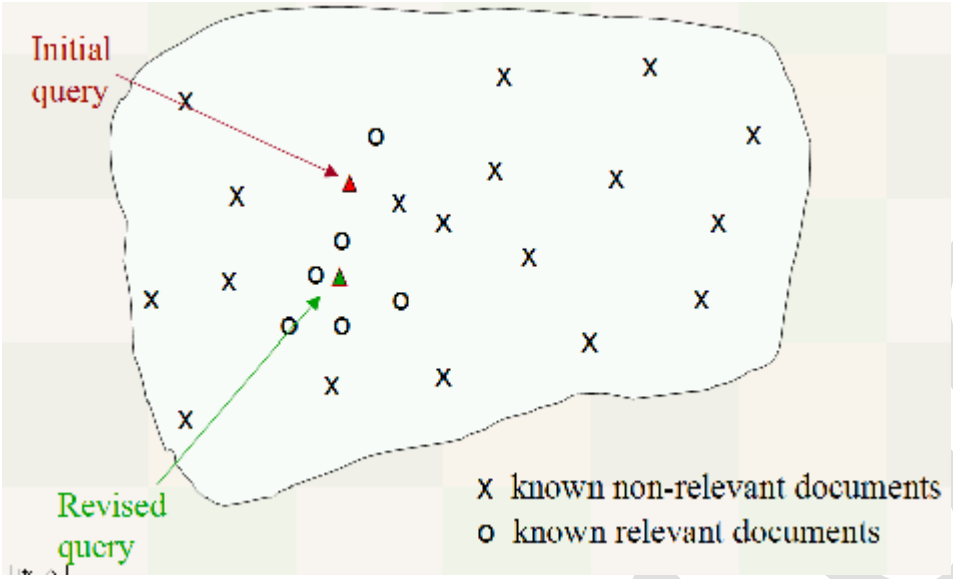
$$\begin{aligned}\vec{q}_{opt} &= \mu(D_r) + [\mu(D_r) - \mu(D_{nr})] \\ &= \frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j + [\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j]\end{aligned}$$

- **The Problem is we don't know the truly relevant docs.** We move the centroid of the relevant documents by the difference between the two centroids.



The Rocchio optimal query for separating relevant and nonrelevant documents.

Rocchio 1971 algorithm



Relevance feedback on initial query

- We can modify the query based on relevance feedback and apply standard vector space model. Use only the docs that were marked. Relevance feedback can improve recall and precision.Relevance feedback is most useful for increasing *recall* in situations where recall is important. Users can be expected to review results and to take time to iterate

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\mu(D_r) - \gamma\mu(D_{nr})$$
$$= \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j$$

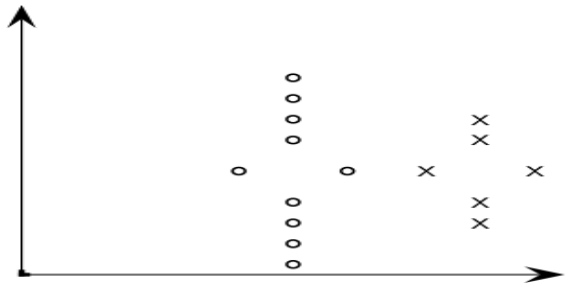
qm: modified query vector; *q0*: original query vector; *Dr* and *Dnr* : sets of known relevant and nonrelevant documents respectively; α, β, and γ: weights

- New query moves towards relevant documents and away from nonrelevant documents.
- Tradeoff α vs. β/γ: If we have a lot of judged documents, we want a higher β/γ.
- Set negative term weights to 0.
- "Negative weight" for a term doesn"t make sense in the vector spacemodel.
- Positive feedback is more valuable than negative feedback.

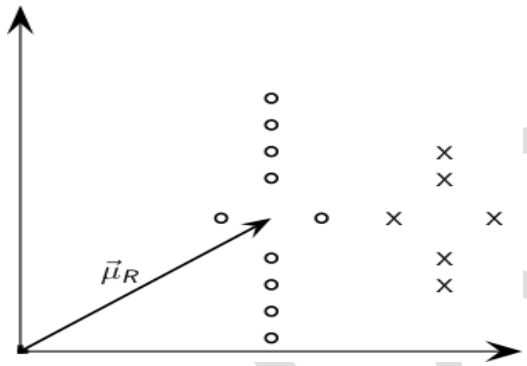
  For example, set $\beta = 0.75$, $\gamma = 0.25$ to give higher weight to positive feedback. Many systems only allow positive feedback.

To Compute Rocchio' vector

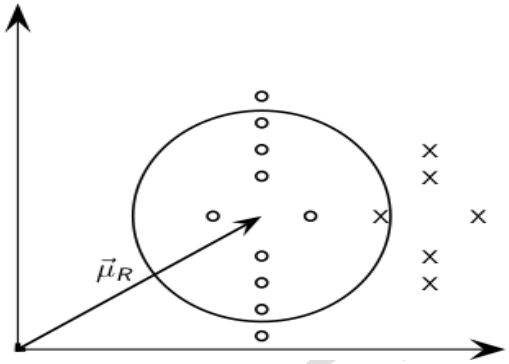
1) circles: relevant documents, Xs: nonrelevant documents



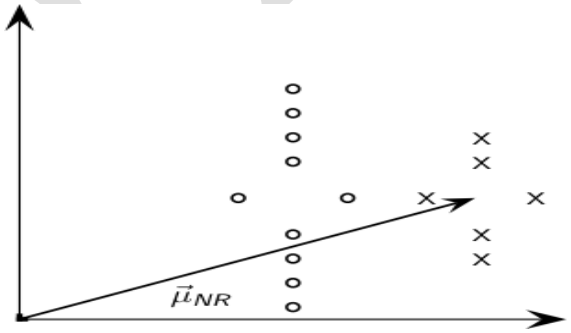
2) $\vec{\mu}_R$ centroid of relevant documents



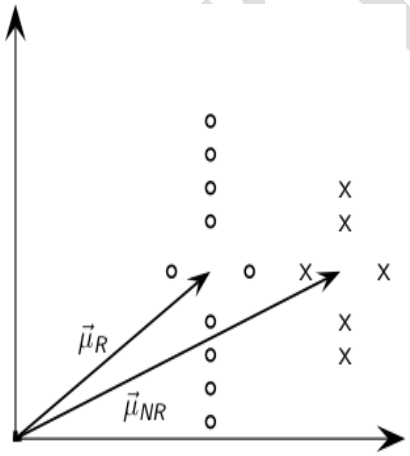
3) $\vec{\mu}_R$ does not separate relevant / nonrelevant.



4) $\vec{\mu}_{NR}$: centroid of nonrelevant documents.

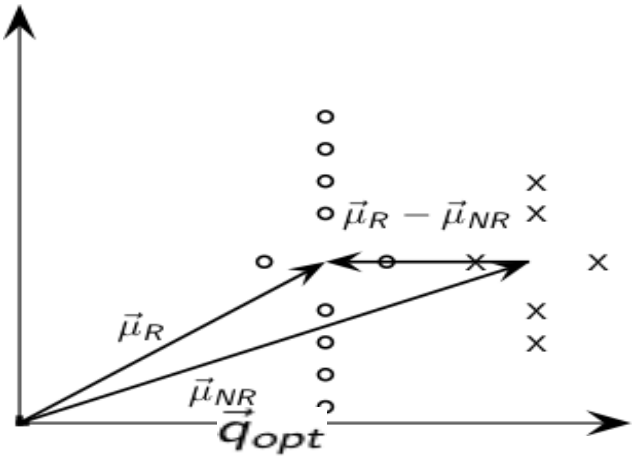


5)



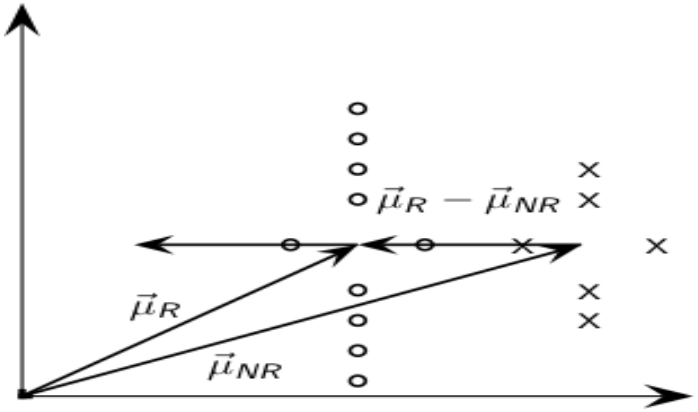
6) difference vector

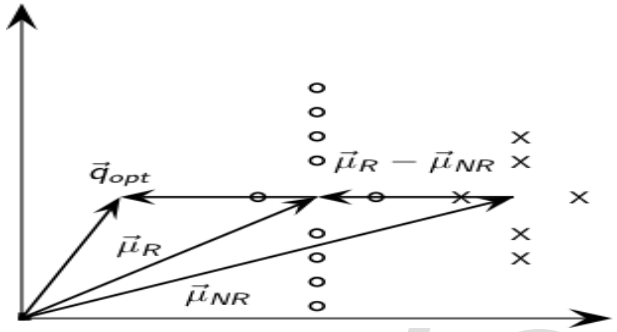
$\vec{\mu}_{NR}$:- $\vec{\mu}_R$



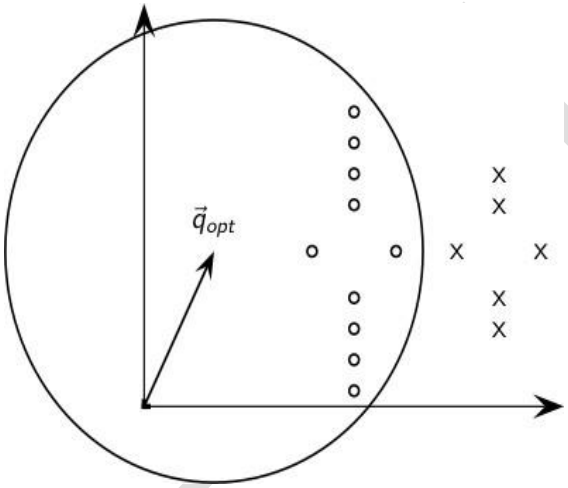
7) Add difference vector to $\vec{\mu}_R$

8) to get





9) $\vec{q}_{opt}$ separates relevant / nonrelevant perfectly.



Disadvantages of Relevance feedback

- Relevance feedback is expensive.
  - Relevance feedback creates long modified queries.
  - Long queries are expensive to process.
- Users are reluctant to provide explicit feedback.
- It‟s often hard to understand why a particular document was retrieved after applying relevance feedback.
- The search engine Excite had full relevance feedback at one point, but abandoned it later.

Pseudo relevance feedback / blind relevance feedback

Pseudo-relevance feedback automates the “manual” part of true relevance feedback.

Pseudo-relevance algorithm:

1) Retrieve a ranked list of hits for the user‟s query
2) Assume that the top *k* documents are relevant.
3) Do relevance feedback (e.g., Rocchio)

Works very well on average, But can go horribly wrong for some queries. Several iterations can cause query drift.

Pseudo-relevance feedback at TREC4

Cornell SMART system given query results showing number of relevant documents out of top 100 for 50 queries (so total number of documents is 5000):

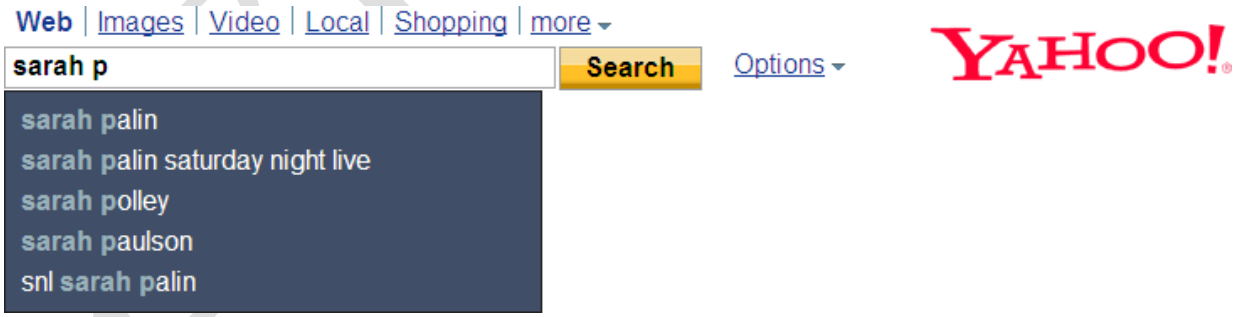
method	number of relevant documents
lnc.ltc	3210
lnc.ltc-PsRF	3634
Lnu.ltu	3709
Lnu.ltu-PsRF	4350

Results contrast two length normalization schemes (L vs. l) and pseudo-relevance feedback (PsRF). The pseudo-relevance feedback method used added only 20 terms to the query. (Rocchio will add many more.) This demonstrates that pseudo-relevance feedback is effective on average.

Query Expansion

- Query expansion is another method for increasing recall. We use "global query expansion" to refer to "global methods for query reformulation". In global query expansion, the query is modified based on some global resource, i.e. a resource that is not query-dependent. Main information we use: (near-)synonymy. A publication or database that collects (near-)synonyms is called a thesaurus.
- There are two types of thesauri:
  - manually created
  - automatically created.

Example



Types of user feedback

There are two types of feedback

1) Feedback on documents - More common in relevance feedback
2) Feedback on words or phrases. - More common in query expansion

Types of query expansion

1) Manual thesaurus (maintained by editors, e.g., PubMed)
2) Automatically derived thesaurus (e.g., based on co-occurrence statistics)
3) Query-equivalence based on query log mining (common on the web as in the "palm" example)

Thesaurus-based query expansion

For each term t in the query, expand the query with words the thesaurus lists as semantically related with *t*.

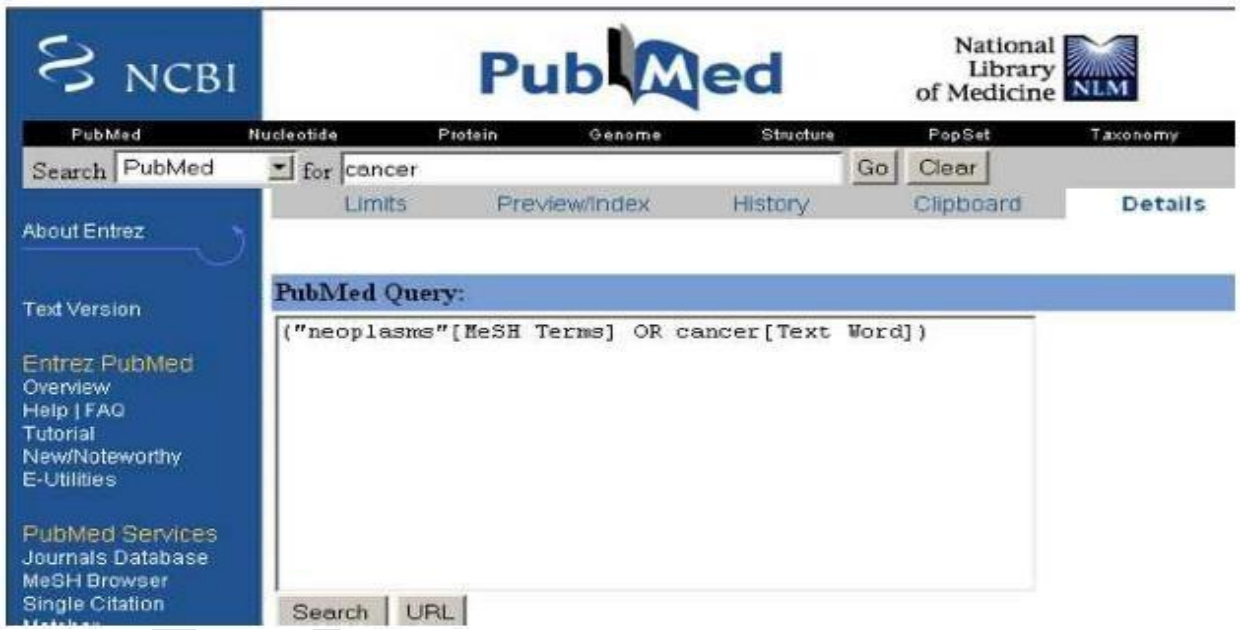
Example : HOSPITAL → MEDICAL

Generally increases recall , May significantly decrease precision, particularly with ambiguous terms

INTEREST RATE → INTEREST RATE FASCINATE

Widely used in specialized search engines for science and engineering, It‟s very expensive to create a manual thesaurus and to maintain it over time. A manual thesaurus has an effect roughly equivalent to annotation with a controlled vocabulary.

Example for manual thesaurus: PubMed



Automatic thesaurus generation

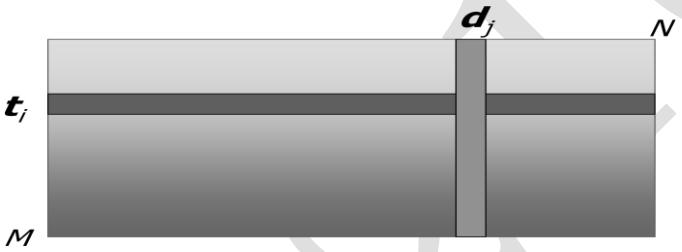
- Attempt to generate a thesaurus automatically by analyzing the distribution of words in documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are similar if they co-occur with similar words.
  - "car" ≈ "motorcycle" because both occur with "road", "gas" and "license", so they must be similar.

- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
  - You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Quality of associations is usually a problem. Term ambiguity may introduce irrelevant statistically correlated terms.
  - "Apple computer" □ "Apple red fruit computer"
- Problems:
  - False positives: Words deemed similar that are not
  - False negatives: Words deemed dissimilar that are similar
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.
- Co-occurrence is more robust, grammatical relations are more accurate.

Co-occurence-based thesaurus: Examples

The simplest way to compute a co-occurrence thesaurus is based on term-term similarities.

in $C = AA^T$ where A is term-document matrix , $w_{i,j}$ = (normalized) weight for (t_i ,d_j)



For each ti, pick terms with high values in C

Word	Nearest neighbors
Absolutely	absurd whatsoever totally exactly nothing
bottomed	dip copper drops topped slide trimmed
captivating	shimmer stunningly superbly plucky witty
doghouse	dog porch crawling beside downstairs
makeup	repellent lotion glossy sunscreen skin gel
mediating	reconciliation negotiate case conciliation
keeping	hoping bring wiping could some would
lithographs	drawings Picasso Dali sculptures Gauguin
pathogens	toxins bacteria organisms bacterial parasite
senses	grasp psyche truly clumsy naive innate

WordSpace demo on web

Query expansion at Search Engines

- Main source of query expansion at search engines: query logs
- Example 1: After issuing the query [herbs], users frequently search for [herbal remedies].
  - → "herbal remedies" is potential expansion of "herb".
- Example 2: Users searching for [flower pix] frequently click on the URL photobucket.com/flower. Users searching for [flower clipart] frequently click on the same URL.
  - → "flower clipart" and "flower pix" are potential expansions of each other.