Information Retrieval (Models)

Fabio Aiolli

http://www.math.unipd.it/~aiolli

Dipartimento di Matematica Pura ed Applicata
Università di Padova

Anno Accademico 2008/2009

The Boolean Model ~1955

The *boolean model* is the first, most criticized, and (until a few years ago) commercially more widespread, model of IR. Its functionalities can often be found in the *Advanced Search* windows of many search engines.

- A document is represented by means of an *and* of index terms;
  d_1=(and jazz saxophone Coltrane)
- A query is represented by a combination, obtained through {and,or,not} of index terms belonging to a controlled vocabulary;
  q_1=(and jazz (or saxophone clarinet) (or Parker Coltrane))
- The matching function is $RSV(d_1,q_1)=1$ if q_1 is a *logical consequence* of d_1 according to Boolean logic.

Boolean queries: Exact match

- Boolean Queries are queries using *AND, OR* and *NOT* together with query terms
  - Views each document as a set of words
  - Is precise: document matches condition or not.
- Primary commercial retrieval tool for 3 decades.
- Professional searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you're getting.

Closed Word Assumption

There is an implicit *Closed World Assumption* in the interpretation of the document representations:

The absence of term t in the representation of d is equivalent to the presence of (not t) in the same representation.

Given document

d_1 = (and jazz saxophone Coltrane)

and query

q_2 = (and jazz (not Parker))

then $RSV(d_1,q_2)=1$ even if $(d_1 \not\models q_2)$

This means that the true definition of $RSV(d_i,q_j) \in \{0,1\}$ is

$RSV(d_i,t) = 1$, if $t \in IREP(d_i)$, 0 otherwise

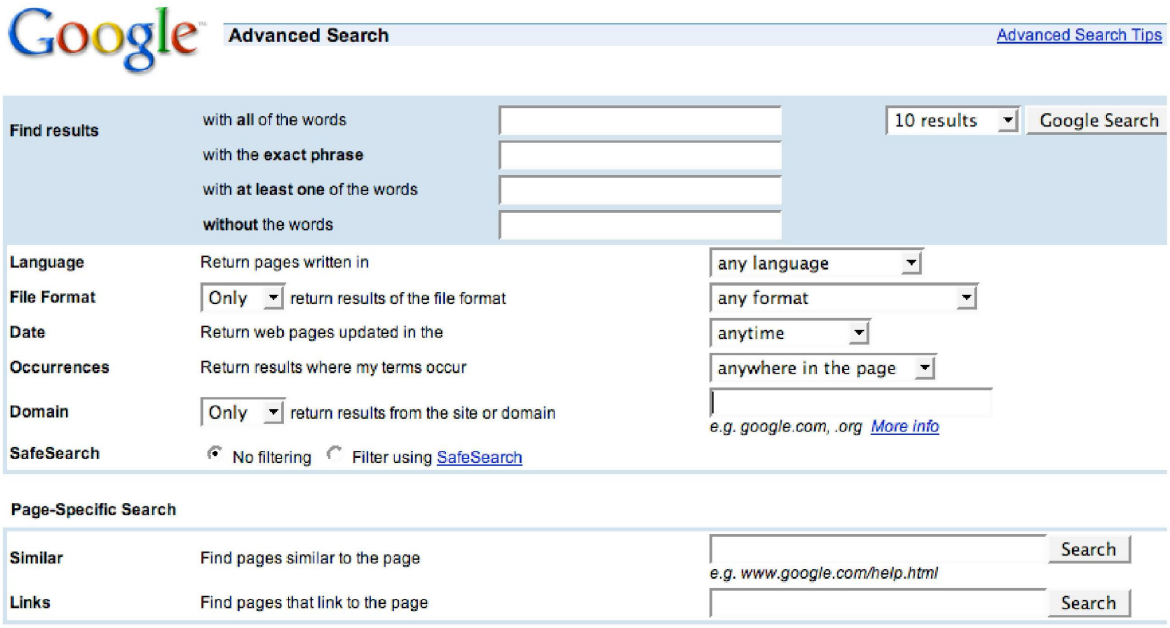
$RSV(d_i,(\textbf{not}\ q_j)) = 1 - RSV(d_i,q_j)$

$RSV(d_i,(\textbf{and}\ q_1 \ldots q_n)) = \min\{RSV(d_i,q_1),\ldots,RSV(d_i,q_n)\}$

$RSV(d_i,(\textbf{or}\ q_1 \ldots q_n)) = \max\{RSV(d_i,q_1),\ldots,RSV(d_i,q_n)\}$

- In queries, expert users tend to use only faceted queries, i.e. disjunctions of quasi-synonyms (*facets*) conjoined through **and**:
  (**and** (**or** jazz classical) (**or** sax* clarinet flute) (**or** Parker Coltrane))

- Extensions:
  - Truncation of the query term; e.g.
    q_3 = (**and** (**or** sax* clarinet) (**or** Parker Coltrane))
  - Information on adjacency, distance and word order invertibility; e.g.
    q_4 = (**and** jazz electric **prox[0,F]** guitar)
  - Possibility to restrict the search to given subfields such as title, author, abstract, publication date, etc. (fielded search).



Query Example

- Which plays of Shakespeare contain the words Brutus *AND* Caesar but *NOT* Calpurnia?

Term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Brutus** AND **Caesar** but NOT **Calpurnia

1 if play contains word, 0 otherwise

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for Brutus, Caesar and Calpurnia (complemented) ➡ bitwise *AND*

110100 *AND* 110111 *AND* 101111 = 100100

Bigger corpora

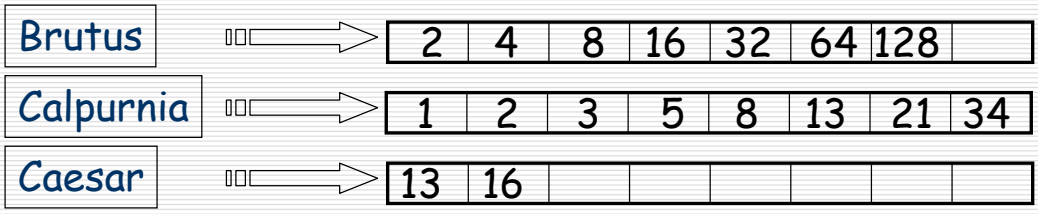
- Consider *n* = 1M documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data in the documents.
- Say there are *m* = 500K distinct terms among these.

Can't build the matrix

- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's.
  - matrix is extremely sparse. Why?
- What's a better representation?
  - We only record the 1 positions.

Inverted index

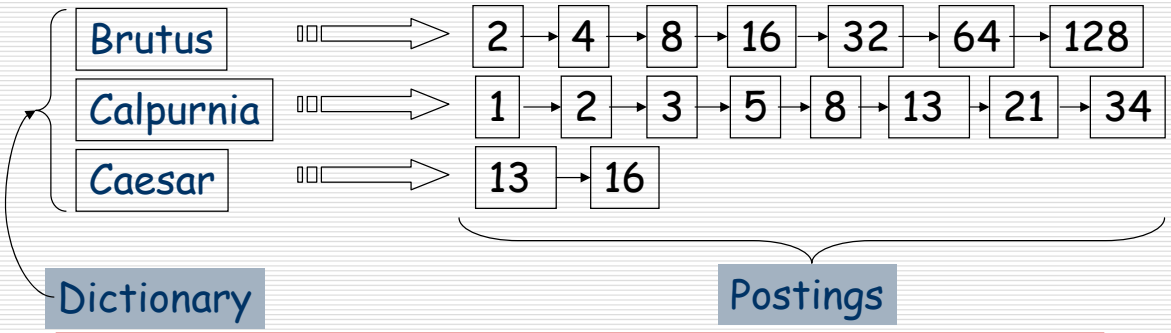
- For each term *t*, we must store a list of all documents that contain t.
- Do we use an array or a list for this?



What happens if the word Caesar is added to document 14?

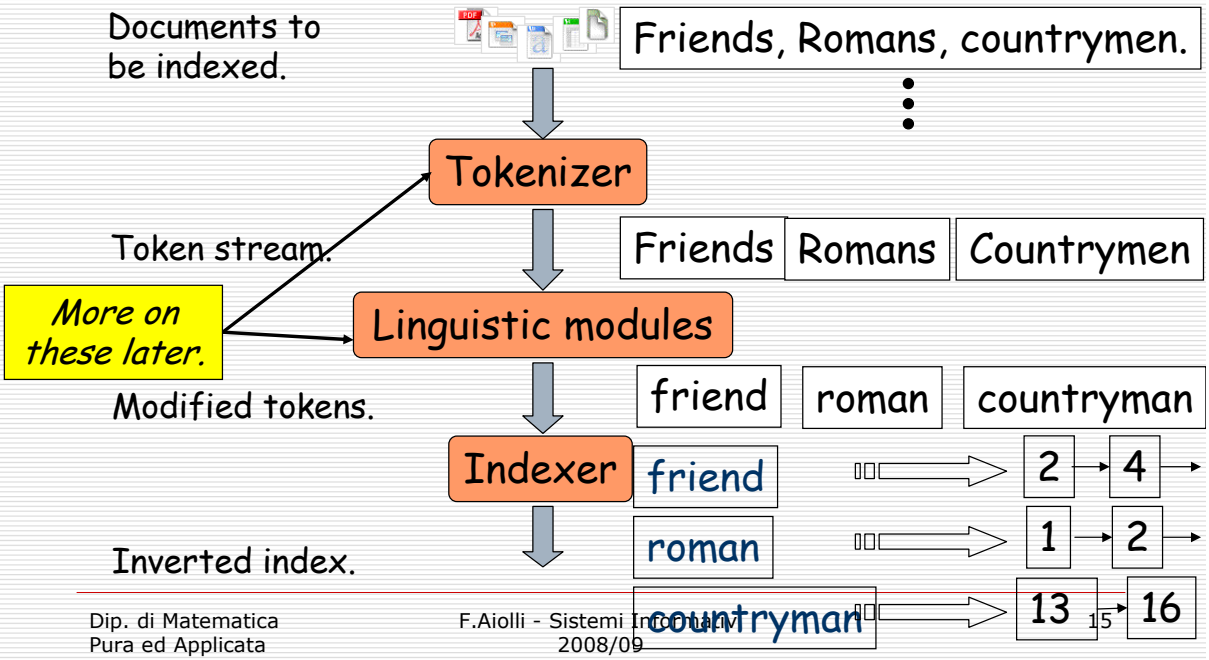
Inverted index

- Linked lists generally preferred to arrays
  - Pro: Dynamic space allocation
  - Pro: Insertion of terms into documents easy
  - Cons: Space overhead of pointers



Sorted by docID (more later on why).

Inverted index construction



Indexer steps

Sequence of (Modified token, Document ID) pairs.

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

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

Sort by terms.

Core indexing step

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

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

- Multiple term entries in a single document are merged.
- Frequency information is added.

Why frequency? Will discuss later

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

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

The result is split into a *Dictionary* file and a *Postings* file.

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

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Where do we pay in storage?

Terms

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

Pointers

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

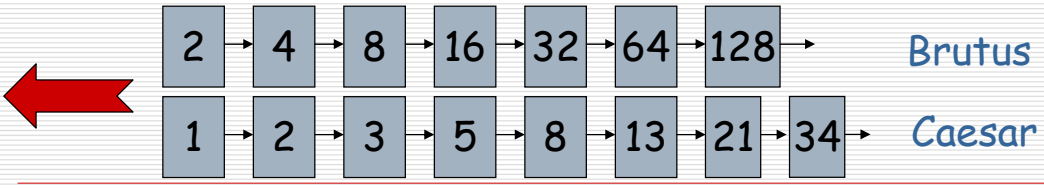
Exercise

Draw the Inverted Index for the following document collection:

- Doc1 new home sales top forecast
- Doc2 home sales rise in july
- Doc3 increase in home sales in july
- Doc4 july new home sales rise

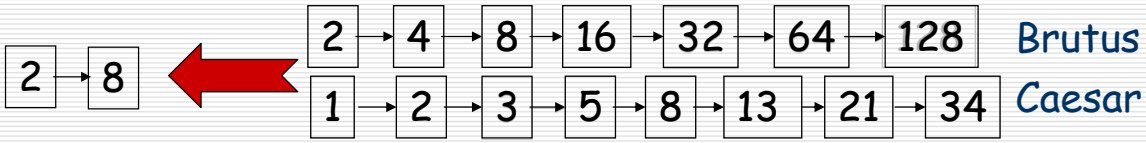
Query processing

- Consider processing the query:
  Brutus *AND* Caesar
  - Locate Brutus in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - *"Merge"* the two postings:



The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *x* and *y*, the merge takes *O(x+y)* operations.
Crucial: postings sorted by docID.

Merge procedure

```
INTERSECT(p1,p2)
    answer ← <>
    while p1 ≠ NIL and p2 ≠ NIL do
        if docID(p1)=docID(p2)
                then Add(answer,docID(p1))
                     p1 ← next(p1)
                     p2 ← next(p2)
                else if docID(p1)<docID(p2)
                     then p1 ← next(p1)
                     else p2 ← next(p2)
    return answer
```

More general merges

Exercise: Adapt the merge for the queries:

Brutus *AND NOT* ***Caesar***

Brutus *OR NOT* ***Caesar***

Can we still run through the merge in time $O(x+y)$?

Query Optimization

What about an arbitrary Boolean formula?

(Brutus *OR* ***Caesar)*** *AND NOT*

(Antony *OR* ***Cleopatra)***

- Can we always merge in "linear" time?
- Consider ***(Brutus*** *AND* ***Caesar) AND Calpurnia***
  - Can we do better?

Exercise

- Recommend a query processing order for
- (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

Term	Posting Size
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Beyond term search

- What about phrases?
- Proximity: Find **Gates** NEAR **Microsoft**.
  - Need index to capture position information in docs.
- Zones in documents: Find documents with (author = **Ullman**) **AND** (text_contains(**automata**)).

Evidence accumulation

- 1 vs. 0 occurrence of a search term
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
- Need term frequency information in docs

Ranking search results

- Boolean queries give inclusion or exclusion of docs.
- Need to measure proximity from query to each doc.

Advantages of the BM

- Possibility to formulate structured queries. E.g. to distinguish 'synonymy' (**or** t_1 t_2) from noun phrases (**and** t_1 **prox[0,F]** t_2)
- In the case of expert users, intuitivity. To those users familiar with Boolean Logic, it is immediately clear why a doc has been or not been retrieved following a given query. This allows query refinement and query reformulation on the part of the user.
- Efficiency obtained through the use of inverted files (Ifs) the data structures in secondary storage in which document representations are physically stored.

Disadvantages of the BM

1. Intimidating (in general, there is not automatic query acquisition). The proponent of the BM where mathematicians; but "lay users" find the language of the BM unnatural.
2. Lack of output magnitude control after a given query, unless the user knows well the distribution of the topics in the collection. This is a consequence of the fact that RSV takes binary values.
3. Output obtained as a result of a given query is not ranked with respect to the estimated degree (probability) of relevance.

4. No importance factors can be attached to the index terms in the IREPs of documents and query.

5. "Flattened", hence unintuitive, results (lack of output discrimination)

Boolean systems have thus mainly been used by expert intermediaries and are widely considered unfit the needs of 'casual' (non professional) information seekers.

The Fuzzy Set Theory

The Fuzzy Set Theory deals with the representation of classes whose boundaries are not well defined

A fuzzy subset of a u.o.d. U is characterized by a membership function $\mu_A : U \rightarrow [0,1]$ which associates with each element $u \in U$ a number $\mu_A(u)$ in the interval [0,1]

- 0 means no membership
- 1 means full membership
- 0<x<1 something in between

Commonly used operations on fuzzy sets are:

- the complement $\mu_{\bar{A}}(u) = 1 - \mu_A(u)$
- the union $\mu_{A \cup B}(u) = \max(\mu_A(u), \mu_B(u))$
- the intersection $\mu_{A \cap B}(u) = \min(\mu_A(u), \mu_B(u))$

The Fuzzy Logic Model (FLM) [Tahani,1976]

The Fuzzy Logic Model (FLM), proposed by Tahani (1976) is an extension, in a quantitative direction, of the boolean model:

- A document is represented by an and of weighted index terms;
  e.g. d_1 = (and <jazz,0.4> <sax,0.3> <Coltrane,0.5>);
- A query q_j is represented as in the Boolean Model
- The matching function RSV computes the degree to which document d_i satisfies q_j
  - $RSV(d_i,t_k) = w_{ki}$
  - $RSV(d_i,(\text{not } q_j)) = 1 - RSV(d_i,q_j)$
  - $RSV(d_i,(\text{and } q_1 .. q_n)) = \min\{RSV(d_1,q_1),..,RSV(d_i,q_n))$
  - $RSV(d_i,(\text{or } q_1 .. q_n)) = \max\{RSV(d_1,q_1),..,RSV(d_i,q_n))$

Here too there is an implicit closed word assumption: the absence of a term t in the representation of d is only a notational abbreviation for the presence of <t,0> in the same representation

Ogawa-Morita-Kobayashi implementation [1991]

- The basic idea is to expand the set of index terms in the query with related terms such that additional relevant documents can be retrieved by the user query
- A thesaurus is constructed by defining a term-term correlation matrix (keyword connection matrix)

$$c_{il} = \frac{n_{il}}{n_i + n_l - n_{il}}$$

n_i :#docs containing term k_i n_l :#docs containing term k_l

n_{il} :#docs containing both of them

- We can use this matrix to define a fuzzy set associated to each index term k_i
- In this fuzzy set, a document d_j has degree of membership

$$\mu_{ij} = 1 - \prod_{k_l \in d_j} (1 - c_{il})$$

- Given a query as a DNF $q = cc_1 \vee cc_2 \ldots \vee cc_p$

$$\mu_{qj} = \mu_{(\sum_i cc_i)j} = 1 - \prod_{i=1}^{p} (1 - \mu_{cc_i j}) = 1 - \prod_i (1 - \prod_k \mu_{cc_i^k j})$$

N.B. We have used algebraic sum (implemented as the complement of negated algebraic product) in place of the max function, and algebraic product in place of the min function, respectively.

Advantages of FLM

1. To a user familiar with Boolean systems, a fuzzy system is different only in that it returns ranked output
2. If the weights $w_{ki} \in \{0,1\}$, the FLM behaves like the BM; a FLM-based IR system is thus compatible also with document representations produced for Boolean systems
3. Possibility to associate weights to index terms in the IREPs of documents
4. Output magnitude control:
   - No more under-dimensioned output, as the terms that represent a document will be many more than in a Boolean representation, even if they have weights 0. As a consequence, almost always at least one document satisfies a query with degree >0
   - No more over-dimensioned output as, thanks to ranked retrieval, the user may scan the ranked list until desired

Disadvantages of FLM

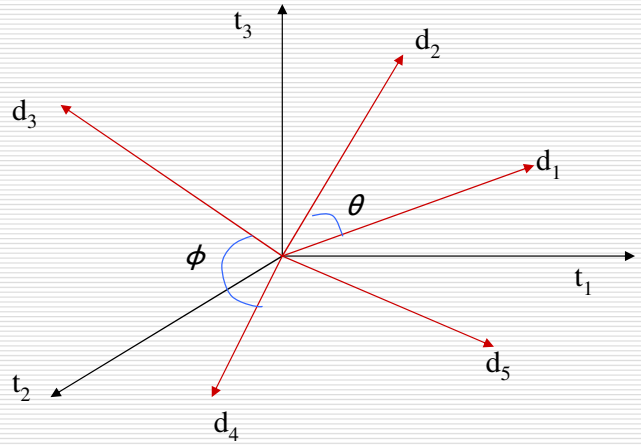
1. As in the BM, no automatic query acquisition;
2. As in the BM, no possibility to associate weights to index terms in the IREPs of queries. Extensions of the FLM to allow this are mathematically complicated
3. As in the BM, flattened, hence unintuitive, results (lack of output discrimination)! The problem derives from the fact that documents are ranked based only on the weight of a single term
   - Following query (and jazz flute Coltrane) the same RSV of 0.1 is given to document
     (and <jazz,0.1><flute,0.1><Coltrane,0.1><Dolphy,0.4>) and to document
     (and <jazz,0.1><flute,0.9>,<Coltrane,0.9><Parker,0.3>) alike;
   - Following query (or jazz flute Coltrane Dolphy) the same RSV of 0.8 is given to document
     (and <jazz,0.5><flute,0.5><Coltrane,0.5>,<Dolphy,0.8>) and to document (and <classical,0.1><flute,0.8><Gazzelloni,0.1>) alike.

The Vector Space Model
Salton 1965

- Document d_i and query q_j are viewed as vectors in a n-dimensional space, where n is the dictionary size
- Why turn docs into vectors?

  First application: Query-by-example
  - Given a doc D, find others "like" it.
  - Now that D is a vector, find vectors (docs) "near" it.

Intuition



Postulate: Documents that are "close together" in the vector space talk about the same things.

The vector space model

Query as vector:

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

Desiderata for proximity

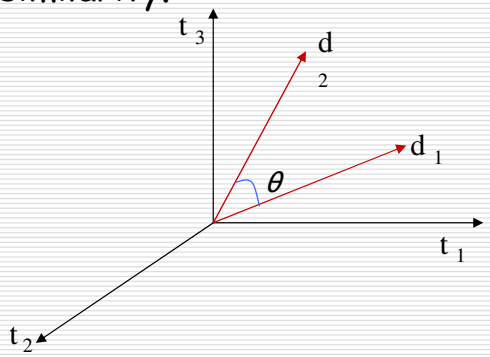
- If d_1 is near d_2, then d_2 is near d_1.
- If d_1 near d_2, and d_2 near d_3, then d_1 is not far from d_3.
- No doc is closer to d than d itself.

First cut

- Distance between d_1 and d_2 is the length of the vector $|d_1 - d_2|$.
  - Euclidean distance
- Why is this not a great idea?
- We still haven't dealt with the issue of length normalization
  - Long documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at *angles* instead

Cosine similarity

- Distance between vectors d_1 and d_2 *captured* by the cosine of the angle x between them.
- Note – this is *similarity*, not distance
  - No triangle inequality for similarity.



Cosine similarity

- A vector can be *normalized* (given a length of 1) by dividing each of its components by its length – here we use the L_2 norm $\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$

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

- This maps vectors onto the unit sphere: longer documents don't get more weight

RSV of the Vector Space Model

- The matching function *RSV* is the cosine of the angle between the two vectors

$$RSV(d_i, q_j) = cos(\alpha) = \frac{\sum_{k=1}^{n} w_{ki} w_{kj}}{||d_i||_2 ||q_j||_2}$$

$$= \sum_{k=1}^{n} \frac{w_{ki} w_{kj}}{\sqrt{\sum_{k=1}^{n} w_{ki}^2} \sqrt{\sum_{k=1}^{n} w_{kj}^2}}$$

Note that

- Given two vectors (either docs or queries), their similarity is determined by their *direction* and *verse* but not their *distance*! You can think of them as lying on the sphere of unit radius
- The following three option returns the same value
  - Cosine of two generic vectors $\mathbf{x}_1$ and $\mathbf{x}_2$
  - Cosine of two vectors $v(\mathbf{x}_1)$ and $v(\mathbf{x}_2)$, where $v(x)$ is a length-normalized vector of x
  - The inner product of vectors $v(\mathbf{x}_1)$ and $v(\mathbf{x}_2)$

Normalized vectors

- For normalized vectors, the cosine is simply the dot product:

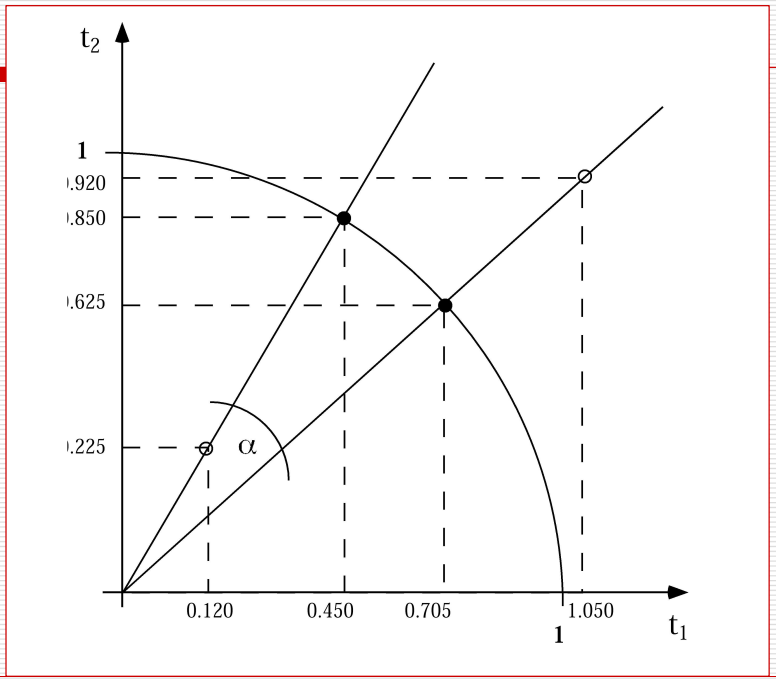
$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$

Exercise

- Euclidean distance between vectors:

$$\left| d_j - d_k \right| = \sqrt{\sum_{i=1}^{n} \left(d_{i,j} - d_{i,k} \right)^2}$$

- Show that, for normalized vectors, Euclidean distance gives the same proximity ordering as the cosine measure



The effect of normalization

- From the p.o.v. of ranking, the three previous options are also equivalent to computing the (inverse of) distance between $v(\mathbf{x}_1)$ and $v(\mathbf{x}_2)$
- Normalization implies that the RSV computes a *qualitative* similarity (what kind of information they contain) instead of *quantitative* (how much information they contain). It has experimentally be proven to yield improved results
- We can verify that in the VSM, ***RSV(d,d)=1*** (ideal document property). Boolean, Fuzzy Logic, and other models also have this property

Example

- Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights*

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

	SaS	PaP	WH
affection	0,996	0,993	0,847
jealous	0,087	0,120	0,466
gossip	0,017	0,000	0,254

cos(SAS, PAP) = .996 x .993 + .087 x .120 + .017 x 0.0 = 0.999
cos(SAS, WH) = .996 x .847 + .087 x .466 + .017 x .254 = 0.889

Advantages of the VSM

1. *Flexibility*. The *most decisive* factor in imposing VSM. The same intuitive geometric interpretation has been re-applied, apart from relevance feedback, in different contexts

   - Automatic document categorization
   - Automatic document filtering
   - Document clustering
   - Term-term similarity computation (terms are indexed by documents, dual)

2. Possibility to attribute weights to the IREPs of both documents and queries thus allowing *automatically indexed document bases*
3. Ranked output and output magnitude control
4. Automatic query acquisition
5. No output "flattening"; each query term contributes to the ranking in an equal way, depending on its weights
6. Much better effectiveness than the BM and the FLM

Disadvantages of the VSM

- Impossibility of formulating "structured" queries: there are no operators
  - Or (synonyms or quasi-synonyms)
  - And (compulsory occurrence of index terms)
  - Not (compulsory absence of index terms)
  - Prox (specification of noun phrases)
- The VSM is based on the hypothesis that the terms are pairwise stochastically independent (binary independence hypothesis). A more recent extension of the VSM relaxes this hypothesis, allowing the Cartesian axes to be non-orthogonal

On Similarity Measures

Any two arbitrary objects are equally similar unless we use domain knowledge!

- Inner Product
- Jaccard and Dice Similarity
- Overlap Coefficient
- Conversion from a distance
- Kernel Functions (beyond vectors)

	Binary case	Non-binary case
Inner Product	$\lvert d_i \cap q_j \rvert$	$d_i \cdot q_j$
Dice	$\frac{2\lvert d_i \cap q_j \rvert}{\lvert d_i \rvert + \lvert q_j \rvert}$	$\frac{2 d_i q_j}{\lvert\lvert d_i \rvert\rvert^2 + \lvert\lvert q_j \rvert\rvert^2}$
Jaccard	$\frac{\lvert d_i \cap q_j \rvert}{\lvert d_i \cup q_j \rvert}$	$\frac{d_i q_j}{\lvert\lvert d_i \rvert\rvert^2 + \lvert\lvert q_j \rvert\rvert^2 - d_i \cdot q_j}$
Overlap Coef.	$\frac{\lvert d_i \cap q_j \rvert}{min(\lvert d_i \rvert, \lvert q_j \rvert)}$	$\frac{d_i q_j}{min(\lvert\lvert d_i \rvert\rvert^2, \lvert\lvert q_j \rvert\rvert^2)}$

Conversion from a distance

Minkowsky Distances

$$L_p(x,z) = \left(\sum_i^n |x_i - z_i|^p\right)^{\frac{1}{p}}$$

When p = ∞ , $L_\infty = \max_i(|x_i - z_i|)$

A similarity measure taking values in [0,1] can always be defined as

$$s_{p,\lambda}(x,z) = e^{-\lambda L_p(x,z)}$$

Where $\lambda \in (0,+\infty)$ is a constant parameter

Kernel functions

A kernel function K(x,z) is a (generally non-linear) function which corresponds to an inner product in some expanded feature space,

i.e. $K(x,z) = \phi(x) \cdot \phi(z)$

Example: For 2-dimensional spaces $x=(x_1,x_2)$

$$K(x,z) = (1 + x \cdot z)^2$$

is a kernel where

$$\phi(x) = (1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2)$$

- Polynomial Kernels:

$$K(x,z) = (x \cdot z + u)^p$$

- RBF Kernels

$$K(x,z) = e^{-\lambda||x-z||^2}$$

- Other Kernels

  Also, there are kernels for structured data: strings, sequences, trees, graphs, an so on

- Polynomial kernels allows to model features which are conjunctions (up to the order of the polynomial)
- Radial Basis Function kernels is equivalent to map into an infinite dimensional Hilbert space
- A string kernel allows to have features that are subsequences of words

The document ranking problem

- We have a collection of documents
- User issues a query
- A list of documents needs to be returned
- **Ranking method is core of an IR system:**
  - **In what order do we present documents to the user?**
  - We want the "best" document to be first, second best second, etc….
- **Idea: Rank by probability of relevance of the document w.r.t. information need**
  - P(relevant|document$_i$, query)

Recall a few probability basics

Bayes' Rule: For events *a* and *b*,

$$p(a,b) = p(a \cap b) = p(a \mid b)\,p(b) = p(b \mid a)\,p(a)$$

$$p(\overline{a} \mid b)\,p(b) = p(b \mid \overline{a})\,p(\overline{a})$$

$$p(a \mid b) = \frac{p(b \mid a)\,p(a)}{p(b)} = \frac{p(b \mid a)\,p(a)}{\sum_{x=a,\overline{a}} p(b \mid x)\,p(x)}$$

Prior ←

Posterior ↑

Odds: $$O(a) = \frac{p(a)}{p(\overline{a})} = \frac{p(a)}{1 - p(a)}$$

The Probability Ranking Principle

"If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, 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."

[1960s/1970s] S. Robertson, W.S. Cooper, M.E. Maron; van Rijsbergen (1979:113); Manning & Schütze (1999:538)

Probability Ranking Principle

Let *x* be a document in the collection.
Let *R* represent **relevance** of a document w.r.t. given (fixed) query and let *NR* represent **non-relevance**.

R={0,1} vs. NR/R

Need to find p(*R*/*x*) - probability that a document *x* is **relevant**.

$$p(R \mid x) = \frac{p(x \mid R)\,p(R)}{p(x)}$$

$$p(NR \mid x) = \frac{p(x \mid NR)\,p(NR)}{p(x)}$$

p(R),p(NR) - prior probability of retrieving a (non) relevant document

$$p(R \mid x) + p(NR \mid x) = 1$$

p(x|R), p(x|NR) - probability that if a relevant (non-relevant) document is retrieved, it is *x*.

Probability Ranking Principle (PRP)

- Simple case: no selection costs or other utility concerns that would differentially weight errors
- ***Bayes' Optimal Decision Rule***
  - ***x*** is **relevant** iff $p(R|x) > p(NR|x)$
- PRP in action: Rank all documents by $p(R|x)$
- Theorem:
  - Using the PRP is optimal, in that it minimizes the loss (Bayes risk) under 1/0 loss
  - Provable if all probabilities correct, etc. [e.g., Ripley 1996]

Probability Ranking Principle

- More complex case: retrieval costs.
  - Let *d* be a document
  - *C* - cost of retrieval of relevant document
  - *C'* - cost of retrieval of non-relevant document
- Probability Ranking Principle: if

$$C \cdot p(R|d) + C' \cdot (1 - p(R|d)) \leq C \cdot p(R|d') + C' \cdot (1 - p(R|d'))$$

for all *d' not yet retrieved*, then *d* **is the next document to be retrieved**

- **We won't further consider loss/utility from now on**

Probability Ranking Principle

- How do we compute all those probabilities?
  - Do not know exact probabilities, have to use estimates
  - Binary Independence Retrieval (BIR) is the simplest model
- Questionable assumptions
  - "Relevance" of each document is independent of relevance of other documents.
    - Really, it's bad to keep on returning **duplicates**
  - Boolean model of relevance
  - That one has a single step information need
    - Seeing a range of results might let user refine query

Probabilistic Retrieval Strategy

- Estimate how terms contribute to relevance
  - How do things like tf, df, and length influence your judgments about document relevance?
- Combine to find document relevance probability
- Order documents by decreasing probability

Probabilistic Ranking

Basic concept:

"For a given query, if we know some documents that are relevant, terms that occur in those documents should be given greater weighting in searching for other relevant documents.

By making assumptions about the distribution of terms and applying Bayes Theorem, it is possible to derive weights theoretically."

Van Rijsbergen

PM are based on the hypothesis that the *distribution* of term in relevant document is *different* from the one in irrelevant documents

Then,

- A greater importance should be given to terms that occur in many relevant documents and are absent in many irrelevant documents
- A smaller importance should be given to terms that occur in many irrelevant documents and are absent in many relevant documents

Binary Independence Model Robertson & Spark Jones (1976)

- Traditionally used in conjunction with PRP
- "**Binary**" = **Boolean**: documents are represented as binary incidence vectors of terms (cf. lecture 1):
  - $\vec{x} = (x_1, \ldots, x_n)$
  - $x_i = 1$ iff term i is present in document *x*.
- "**Independence**": terms occur in documents independently
- Different documents can be modeled as same vector
- Bernoulli Naive Bayes model (cf. text categorization!)

Binary Independence Model

- Queries: binary term incidence vectors
- Given query ***q***,
  - for each document ***d*** need to compute ***p(R|q,d)***.
  - replace with computing ***p(R|q,x)*** where ***x*** is binary term incidence vector representing ***d*** Interested only in ranking
- Will use odds and Bayes' Rule:

$$O(R \mid q, \vec{x}) = \frac{p(R \mid q, \vec{x})}{p(NR \mid q, \vec{x})} = \frac{\frac{p(R \mid q)\, p(\vec{x} \mid R, q)}{p(\vec{x} \mid q)}}{\frac{p(NR \mid q)\, p(\vec{x} \mid NR, q)}{p(\vec{x} \mid q)}}$$

Binary Independence Model

$$O(R \mid q,\vec{x}) = \frac{p(R \mid q,\vec{x})}{p(NR \mid q,\vec{x})} = \frac{p(R \mid q)}{p(NR \mid q)} \cdot \frac{p(\vec{x} \mid R,q)}{p(\vec{x} \mid NR,q)}$$

Constant for a given query

Needs estimation

- Using **Independence** Assumption:

$$\frac{p(\vec{x} \mid R,q)}{p(\vec{x} \mid NR,q)} = \prod_{i=1}^{n} \frac{p(x_i \mid R,q)}{p(x_i \mid NR,q)}$$

- So: $O(R \mid q,d) = O(R \mid q) \cdot \prod_{i=1}^{n} \frac{p(x_i \mid R,q)}{p(x_i \mid NR,q)}$

Binary Independence Model

$$O(R \mid q,d) = O(R \mid q) \cdot \prod_{i=1}^{n} \frac{p(x_i \mid R,q)}{p(x_i \mid NR,q)}$$

- Since x_i is either 0 or 1:

$$O(R \mid q,d) = O(R \mid q) \cdot \prod_{x_i=1} \frac{p(x_i=1 \mid R,q)}{p(x_i=1 \mid NR,q)} \cdot \prod_{x_i=0} \frac{p(x_i=0 \mid R,q)}{p(x_i=0 \mid NR,q)}$$

- Let $p_i = p(x_i=1 \mid R,q); \quad r_i = p(x_i=1 \mid NR,q);$

- Assume, for all terms not occurring in the query ($q_i=0$) $p_i = r_i$

Then...

This can be changed (e.g., in relevance feedback)

Binary Independence Model

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{x_i = q_i = 1} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i = 0 \\ q_i = 1}} \frac{1 - p_i}{1 - r_i}$$

All matching terms

Non-matching query terms

$$= O(R \mid q) \cdot \prod_{x_i = q_i = 1} \frac{p_i (1 - r_i)}{r_i (1 - p_i)} \cdot \prod_{q_i = 1} \frac{1 - p_i}{1 - r_i}$$

All matching terms

All query terms

Binary Independence Model

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{x_i = q_i = 1} \frac{p_i (1 - r_i)}{r_i (1 - p_i)} \cdot \prod_{q_i = 1} \frac{1 - p_i}{1 - r_i}$$

Constant for each query

Only quantity to be estimated for rankings

- Retrieval Status Value:

$$RSV = \log \prod_{x_i = q_i = 1} \frac{p_i (1 - r_i)}{r_i (1 - p_i)} = \sum_{x_i = q_i = 1} \log \frac{p_i (1 - r_i)}{r_i (1 - p_i)}$$

Binary Independence Model

- All boils down to computing RSV.

$$RSV = \log \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} = \sum_{x_i=q_i=1} \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

$$RSV = \sum_{x_i=q_i=1} c_i; \qquad c_i = \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

So, how do we compute c_i's from our data ?

Binary Independence Model

- Estimating RSV coefficients.
- For each term *i* look at this table of document counts:

Documents	Relevant	Non-Relevant	Total
$X_i=1$	s	$n-s$	n
$X_i=0$	$S-s$	$N-n-S+s$	$N-n$
Total	S	$N-S$	N

- Estimates: $p_i \approx \frac{s}{S} \qquad r_i \approx \frac{(n-s)}{(N-S)}$

$$c_i \approx K(N,n,S,s) = \log \frac{s/(S-s)}{(n-s)/(N-n-S+s)}$$

For now, assume no zero terms.

Estimation – key challenge

- If non-relevant documents are approximated by the whole collection, then r_i (prob. of occurrence in non-relevant documents for query) *is* n/N and
  - log $(1- r_i)/r_i$ = log $(N- n)/n \approx$ log N/n = IDF!
- p_i (probability of occurrence in relevant documents) can be estimated in various ways:
  - from relevant documents if know some
    - Relevance weighting can be used in feedback loop
  - constant (Croft and Harper combination match) – then just get idf weighting of terms
  - proportional to prob. of occurrence in collection
    - more accurately, to log of this (Greiff, SIGIR 1998)

Iteratively estimating p_i

1. Assume that p_i constant over all x_i in query
   - p_i = 0.5 (even odds) for any given doc
2. Determine guess of relevant document set:
   - V is fixed size set of highest ranked documents on this model (note: now a bit like tf.idf!)
3. We need to improve our guesses for p_i and r_i, so
   - Use distribution of x_i in docs in V. Let V_i be set of documents containing x_i
     - $p_i = |V_i| / |V|$
   - Assume if not retrieved then not relevant
     - $r_i = (n_i - |V_i|) / (N - |V|)$
4. Go to 2. until converges then return ranking

Advantges and Disadvantages

- Advantage
  - Documents are ranked in decreasing order of probability of being relevant
- Disadvantages
  - The need to guess the initial separation of documents into relevant and irrelevant
  - It does not take into account the frequency with which a term occurs inside a document
  - The adoption of independence of index terms

PM: Other directions

- Just one of many types of "Naïve Bayes" IR models. Important research directions are:
  - Introducing non-binary document weights
  - Introducing document length normalization
  - Relaxing the independence assumption

Bayesian Networks
Jensen and Jensen [2001]

- Probabilistic Graphical Model for information retrieval
- Use of directed graph to describe dependencies between variables (e.g. terms)
- Algorithms for propagating probabilities (infering) by using the Bayes rule