Digital Libraries: Information Retrieval

RAFFAELLA BERNARDI

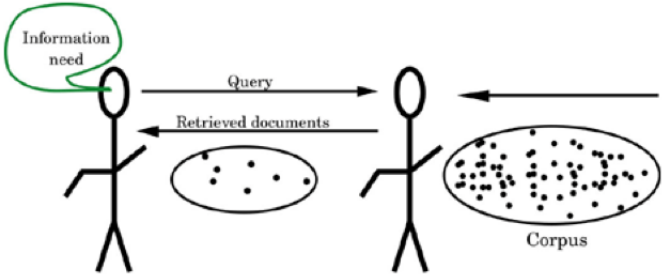
UNIVERSITÀ DEGLI STUDI DI TRENTO

P.ZZA VENEZIA, ROOM: 2.05, E-MAIL: BERNARDI@DISI.UNITN.IT

Contents

1. Finding out about

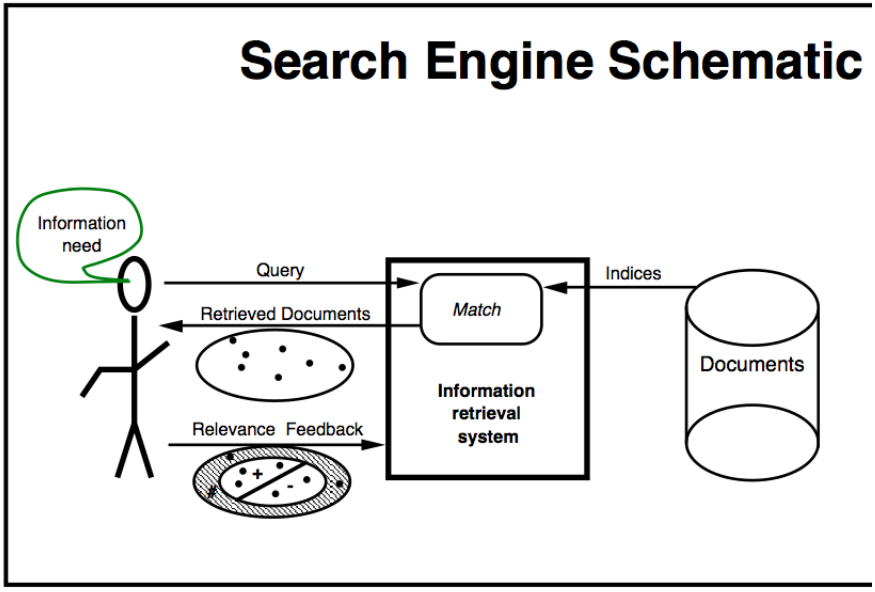
The construction of an answer



- Forming a good answer is hard!
- *Assume*: answer = {predefined text passages}

© R. K. Belew 1996-2001

1.1. Information Retrieval in a picture



Finding Out About

© R. K. Belew 1996-2001

1.2. History of IR

1960-70's *Small* text retrieval systems; basic boolean and vector-space retrieval models.

1980's *Large* document database system, many run by companies

1990's Searching FTPable documents on the *Internet*; Searching the World Wide Web (Yahoo, Altavista)

2000's Link analysis for web search (Google); QA (TREC QA track); Multimedia IR; Cross-Language IR; Document Summarization; Recommender systems, automated text categorization and clustering (iTunes, Amazon, Flick)

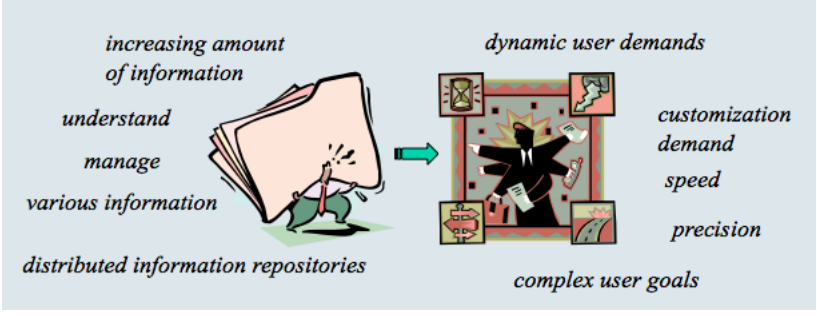
1.3. Information Retrieval

> Information Retrieval (IR) is finding material (usually documents) of an *unstructured* nature (usually text) that satisfies an information need from within *large* collections (usually stored on computers). [Maggining, Raghavan, Schütze 2009]

In reality, almost no data is "unstructured" (e.g. linguistic structures, or document metadata). Hence, IR is the process of applying algorithms over *unstructured, semi-structured or structured* data in order to satisfy a given information query. It needs to be efficient w.r.t. algorithms, query building, data organization.

Structured data and their query/answering methods Databases are the typical structured data. They are queried with query languages that exploit the structure of the data and might give as answer "I am sorry, I can only look up your order if you can give me your order ID". They need exact info (as in the database schema) and they return exact answers.

1.4. IR challenges



1.5. Retrieval: Example

Take a book of one million words, for instance “Shakespeare’s Collected Works”. You want to know which plays contain the words “Brutus” and “Caesar” and do not contain “Calpurnia”.

An easy way of achieving this task is “to read” through the text. There is a simple unix command for this called “grep”. But what about if:

- the text collection is much *bigger*? Billions or trillions of words.
- you want to search for the word “Romans” when occur somewhere *near to* “countrymen”
- You want to have a *ranked* retrieval of the document to know which are the “best” answers.

Instead of grepping the text, IR systems *index* it.

2. Index

An index is usually the most common way to find content in a book or a journal:

- Table of contents (to know where a topic is in book)
- Catalog (to know where a book is in the library)
- Concordances (to know where a word is in a book)

IR uses something like "concordances". What is needed is an index of the words contained in the whole collection.

A collection is a set of "documents" each described by a set of representative terms.

Need to define the granularity of "documents" (files or chapters or pages etc.) and "representative terms".

The index will contain a list of the different terms that appear in the whole collection; (b) for each term, the list of documents where the term appears; (c) additional term-related information.

2.1. Manual and Automatic indexing

Manual indexing means that people, domain experts, have selected appropriate keywords for the documents. Automatic indexing refers to algorithmic procedures for accomplishing this same result.

Indexing is a core problem in IR, it's the fundamental connection between the users' expressions of information need and the documents that can satisfy them.

Automatic and manual indexing need not be viewed as competing alternatives. There are ways to train machine learning indexing systems.

Exercise Build a manual index (assigned keywords from a controlled vocabulary) for the texts given.

2.2. Index: example

The matrix below represent whether a certain word occurs (1) or does not occur (0) in a given document.

	d1	d2	d3	d4	d5	d6	...
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	
...							

Hence, the documents that contain “Brutus” and “Caesar” but do not contain “Calpurnia” are:

$$110100 \text{ and } 110111 \text{ and } 101111 = 100100$$

in words, d1, d4.

2.3. Problems with index

Usually IR is done from a very large document collection (or "corpus"). For instance, assume we have:

- 1 million documents,
- each document is about 1,000 words (2-3 book pages),
- each word is about 6 bytes.
- Then, the document collection is about 6 gigabytes (GB) size.
- With around 500,000 distinct terms

The term-document matrix would be too big: $500K \times 1M$ has half-a-trillion 0's and 1's. They would not fit in a computer's memory.

The 0's could be many (sparsa data). It might be better to record only the things that do occur, that is, the 1's. This is the idea behind "inverted index".

2.4. Inverted Index

Dictionary	*Postings*							
Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				
...								

Terminology Note The dictionary is also called "vocabulary" or "lexicon". Each item in the list of the documents in which the word occur is called "posting". The list is called "postings list" (or "inverted list").

2.5. Indexing and Inverted Index

Formally, *indexing* is the process of associating one or more keywords with each document they are about. The vocabulary used can either be controlled or uncontrolled (closed or open.)

$$\texttt{Index} : doc_i \rightarrow \{kw_j\}$$

The *inverse mapping* captures, for each keyword, the documents it describes:

$$\texttt{Index}^{-1} : kw_i \rightarrow \{doc_j\}$$

Keywords are linguistic atoms – typically words, pieces of words, or phrases – used to characterize the content of a document. They must bridge the gap between the users' characterization of information need (i.e. their queries) and the characterization of the documents'topical focus against which these will be matched.

2.6. Indexer: step 1

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

- Sequence of token, Document ID pairs.
- Tokens are modified: lower case

Term	DocID
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

2.7. Indexer: step 2

- Sorted alfabetically 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	DocID
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

2.8. Indexer: step 3

- **Multiple term entries in a single document** are merged: **word types** only are stored in the inverted index
- **Frequency** information is added
- The intuition if that frequency contributes to the **relevance** of a term for a document

Term	DocID
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

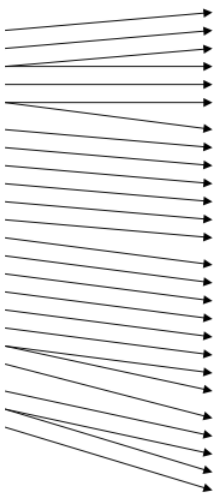
Term	DocID	Term freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

2.9. Indexer: step 4

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

Term	DocID	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	Coll 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



DocID	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

2.10. Index and inverted index: Exercise

Consider these documents

Doc 1 breakthrough drug for schizophrenia

Doc 2 new schizophrenia drug

Doc 3 new approach for treatment of schizophrenia

Doc 4 new hopes for schizophrenia patients

(a) Draw the term-document matrix for this document collection.

(b) Draw the inverted index representation of this collection.

(c) what are the returned results for the query: schizophrenia AND drug;

2.11. Index size and space

A document level index needs a value for each ⟨ term, document ⟩ pair. A word-level index needs a value for each word in the collection. We could end up having an inverted index that takes as much space as the text itself.

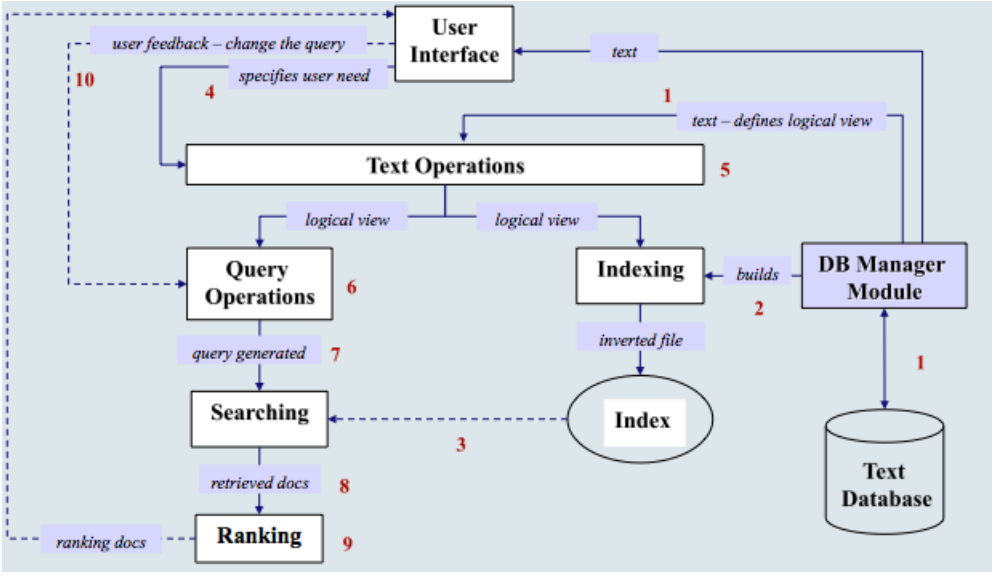
The elimination of *stop words* helps reducing the space.

Several compression methods have been proposed.

2.12. Compression Methods

Method	Memory (Mbytes)	Disk (Mbytes)	Time (hours)
Linked lists (memory)	4,000	0	6
Linked lists (disk)	30	4,000	1,100
Sort-based	40	8,000	20
Sort-based compressed	40	680	26
Sort-based multiway merge	40	540	11
Sort-based multiway in-place	40	150	11
In-memory compressed	420	1	12
Lexicon-based, no extra disk	40	0	79
Lexicon-based, extra disk	40	4,000	12
Text-based partition	40	35	15

2.13. Retrieval Process



3. Query Languages

The vocabulary used can either be controlled or uncontrolled (closed or open vocabularies.)

The execution (transformation into a formal language) of a query depend on the underling "data model":

- Structured data (data in tables.)
- Semi-structured data (document's structure)
- Unstructured data (free text.)

It also depends on whether we want an exact match between the query terms and the documents (boolean query) or a "full-text retrieval" (non boolean query). Ranking of the result is important in both cases.

3.1. Query processing

Boolean Query mechanics Find X: return all documents containing the term X (X=single words or phrases); complex queries built with boolean operators ('and", "or" and also "but_not").

It's precise: document matches the query or does not match it.

Pattern Matching a set of syntactic features that occur in a text segment; segments that fulfils the pattern specifications –pattern match. Retrieve pieces of text that have some property. Eg. by means of *regular expressions*: general pattern build up by simple strings and operators, e.g. "pro(blem|tein)(s|ε)(0|1|2)*" will match "problem02" and "proteins"

3.2. Example: queries

An example of commercial boolean query.

Information need: Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company

Query: “trade secret” /s disclos! /s prevent /s employe!

Information need: Requirements for disabled people to be able to access a workplace

Query: disab! /p access! /s work-site work-place (employment /3 place)

Information need: Cases about a host’s responsibility for drunk guests

Query: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Legenda space disjunction; /s, /p, /k stand for matches in the same sentence, paragraph, or within k words, resp. ! wildcard, eg. liab! matches all words starting with “liab”.

3.3. Exercise with Boolean Model

D1= “computer information retrieval”
D2= “computer retrieval”
D3= “information”
D4= “computer information”

Q1= “information AND retrieval”
Q2 = “information BUT NOT Computer”

Q1= “information AND retrieval” D1
Q2 = “information BUT NOT Computer” D3

3.4. Query processing: Boolean Model

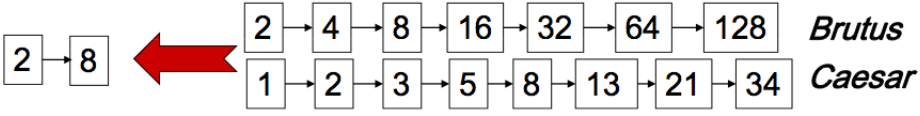
- Consider processing the query:

  Brutus *AND* ***Caesar***

  – Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.

  – Locate *Caesar* in the Dictionary;
    - Retrieve its postings.

  – "Merge" the two postings:



3.5. Context Queries

Complement single-word queries with search for "context" –words which are near to other words.

- Phrase context query: sequence of single-word queries.
- Proximity context query: more relaxed version of phrase query; sequence of single-word queries with a max allowed distance (in characters or words) between them.

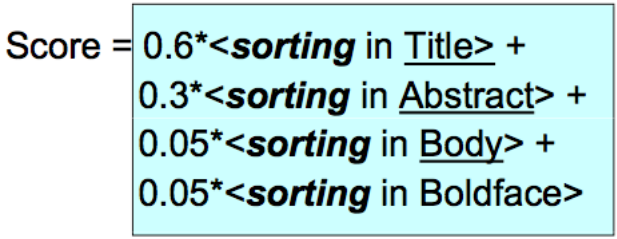
Example: "information retrieval" vs. "information about retrieval" (1 w.) vs. "information with respect to the retrieval" (4 w.)

4. Ranking

- In Boolean queries a document either matches or does not match the query.
- The order of the returned documents is not specified (often reflects the internal organization of the index.).
- In large collections, the number of (unordered) returned documents is far too big for "human consumption"
- Need to rank the matching documents according to the (estimated) relevance of a document to a query (assigning a score to a (document, query) pair.
- In actual Digital Libraries, ranking is essential for both: Boolean queries and Full-text (non-boolean) queries.

4.1. Ranking: boolean query

A simple scoring method is to use linear combination of boolean values, by assigning a weight to each field. E.g. the query contains “sorting”:



the document score is obtained by *adding* up the wighted contributions of the fields. The *weights* are evaluated with *Machine Learning* methods, based on a test corpus, a suite of test queries and a set of relevance judgments.

4.2. Full text queries (non boolean)

The query is a sequence of query terms, and it's not practical to consider them as an AND nor as an OR query.

Need to define a method to compute a *similarity measure* between the query and the document. Results will be ranked according to the similarity measures.

We need to represent the document and the query in some mathematical form so to compute their similarity.

The most used format is a vector.

4.3. Document and Query as a binary vector

queries: “eat”; “hot porridge”.

d	Document vectors $\langle w_{d,t} \rangle$									
	col	*day*	*eat*	*hot*	*lot*	*nin*	*old*	*pea*	*por*	*pot*
1	1	0	0	1	0	0	0	1	1	0
2	0	0	0	0	0	0	0	1	1	1
3	0	1	0	0	0	1	1	0	0	0
4	1	0	0	1	0	0	0	0	0	1
5	0	0	0	0	0	0	0	1	1	0
6	0	0	1	0	1	0	0	0	0	0
eat	0	0	1	0	0	0	0	0	0	0
hot porridge	0	0	0	1	0	0	0	0	1	0

4.4. Similarity Measure

Given the pair (“hot porridge”,d1) there similarity measure can be obtained as their *inner product*

$$(0,0,0,1,0,0,0,0,0,1,0) \bullet (1,0,0,1,0,0,0,0,1,1,0) = 2$$

Drawbacks :

- *No* account of *term frequency* in the document (i.e. how many times a term appears in the document)
- *No* account of *term scarcity* (in how many documents the term appears)
- *Long documents* with many terms are favoured

We come back to this problem tomorrow.

5. IR type of evaluation

- Assistance in formulating queries
- Speed of retrieval
- Resources required
- Presentation of documents
- Appealing to users
- Evaluation generally comparative
- Cost-benefit analysis possible

We are interested in retrieval *performance evaluation*

5.1. IR Evaluation

Effectiveness the ability of IR system to retrieve relevant documents and suppress non-relevant documents; it's related to relevancy of retrieved items.

Relevancy typically it's *subjective*, situational (user's current needs).

Collection *Real collections*: never know full set of relevant documents. Compare retrieval performance with a *Test Collection*: set of documents, set of queries, set of relevance judgments (which docs relevant to each query.)

Method Compare performance of two techniques: each technique used to evaluate test queries; results (set or ranked list) compared using some performance measure.

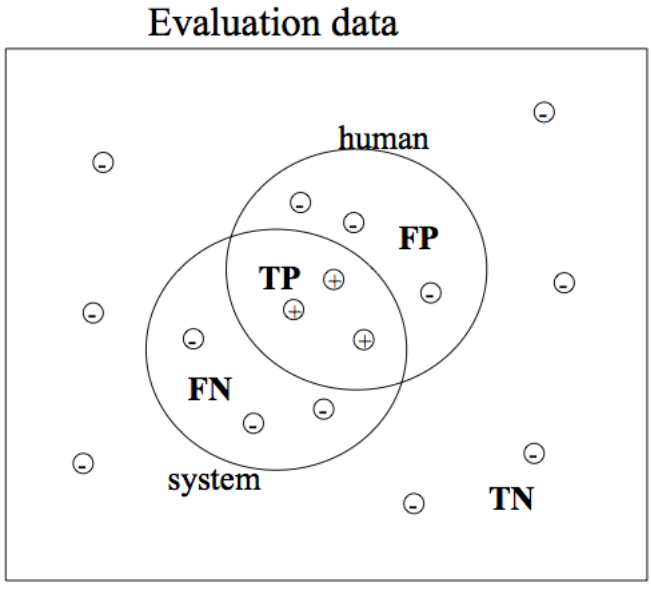
Measures most common measures: Precision and Recall. Usually, use multiple measures to get different views of performance. Usually, test with multiple collections, performance is collection dependent.

5.2. Contingency Matrix

A contingency matrix allows to compare the results automatically obtained by an IR system with the correct judgments manually provided by a human on the same dataset.

	Relevant	Not Relevant
Retrieved	True Positive (TP)	False Positive (FP)
Not retrieved	False Negative (FN)	True Negative (TN)

5.3. Evaluation Setting



⊕ Relevant document

⊖ Not relevant document

TP: True Positive
TN: True-Negative
FP: False-Positive
FN: False-Negative

5.4. Evaluation Measure

Accuracy Percentage of documents correctly classified by the system.

$$\frac{\mathbf{TP + TN}}{TP+TN+FP+FN}$$

Error Rate Inverse of accuracy. Percentage of documents wrongly classified by the system

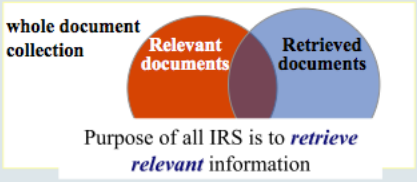
$$\frac{\mathbf{FP+FN}}{TP+TN+FP+FN}$$

But: accuracy is not a good measure for IR.

In IR the number of TN (true negative) is usually much bigger than the number of true positive, false positive and false negative.

Hence, we need a measure that does not consider TNs.

5.5. Precision



The ability of the search to find ***all*** of the relevant documents in the corpus

The ability of the search to retrieve ***top-ranked*** documents that are mostly relevant

Retrieved documents that are relevant

$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

5.6. Precision, Recall and F-Measure

- **Precision**: percentage of relevant documents correctly retrieved by the system (TP) with respect to all documents *retrieved by the system* (TP + FP). (how many of the retrieved books are relevant?)

$$P = \frac{TP}{TP + \mathbf{FP}}$$

- **Recall**: percentage of relevant documents correctly retrieved by the system (TP) with respect to all documents *relevant for the human* (TP + FN). (how many of the relevant books have been retrieved?)

$$R = \frac{TP}{TP + \mathbf{FN}}$$

- **F-Measure**: Combine in a single measure Precision (P) and Recall (R) giving a *global estimation of the performance* of an IR system

$$F = \frac{2PR}{R + P}$$

5.7. Exercise

	Relevant	Not-relevant
Retrieved	TP=10	FP =30
Not retrieved	FN= 5	FN=55

Accuracy = (10+55)/100 = 65/100 = 0,65
Error rate = (5+30)/100 = 35/100 = 0,35
Precision = 10/10+30 = 0,25
Recall = 10/10+5 = 0,66
F-Measure = (2 * 0,25 * 0,66)/(0,25+0,66) = 0,38

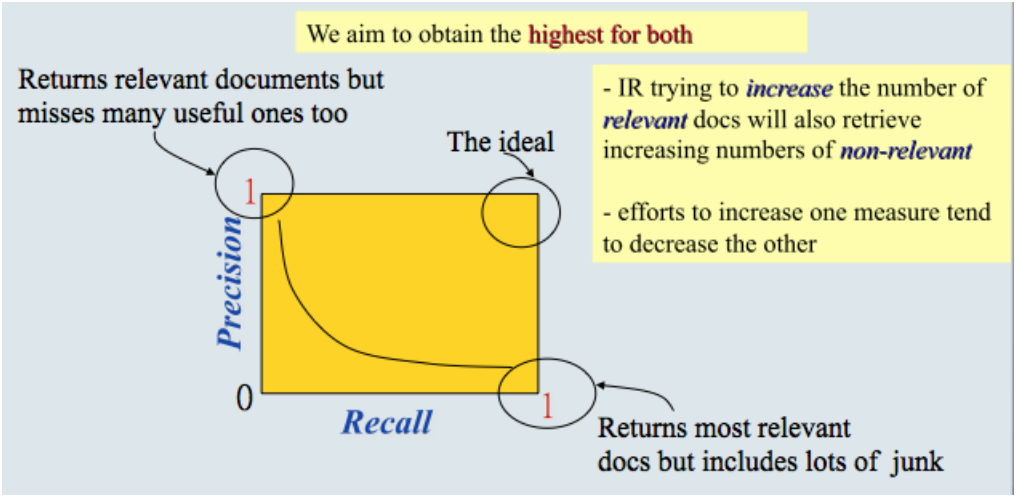
5.8. Problem with Recall

Determining Recall is difficult, the total number of relevant items is sometimes not available.

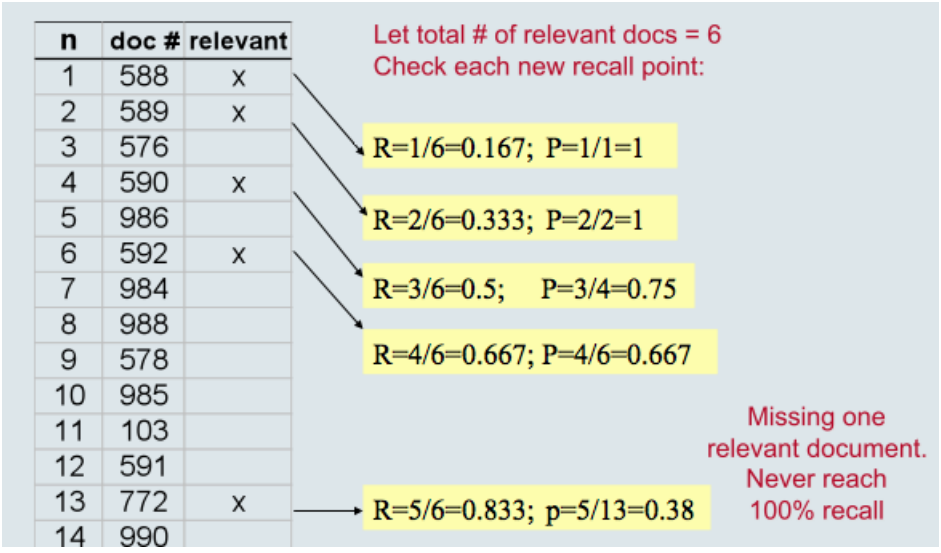
Sample across the database and perform relevance judgment on these items.

Apply different retrieval *algorithms* to the same database for the same query. The aggregate of relevant items is taken as the total relevant set.

5.9. Trade-off between Recall & Precision



5.10. Precision/Recall: at position



5.11. Solutions for Low Recall

- suffix removal allows word variants to match
- word roots often precede modifiers
- Boolean system often allow manual truncation
- stemming does automatic truncation
- use for thesaurus-based query expansion (to detect synonym)
- word sense disambiguation: indexing word meaning rather than words; context provide clues to word meaning.

6. Conclusions

The basic issues and technologies presented are good and still at work, but we would need more flexible search and to better grasp the user information need.

More flexible search

- We would like to better determine the set of terms in the dictionary and to provide retrieval that is tolerant to *spelling mistakes* and inconsistent *spelling choices*.
- It is often useful to search for compound or phrases (e.g. “operating system”, and to handle *proximity queries*.
- A Boolean model only records term presence or absence, but often we would kike to accumulate evidence giving more weight to documents that have a term *several times* as opposed to one that contain it only once.
- Boolean queries just retrieve a set of matching documents, but often we wish to have a method to order (or *rank*) the returned results, viz. a method to decide how good a document is for a query.

Vocabulary is not meaning An *information need* is the topic about which the user desires to know more. It is different from a query, which is what the user conveys to the computer in the attempt to communicate the information need.

A document is relevant if it is one that the user perceives as containing information of value with respect to their personal information need. In short:

Computers match words (character strings) not meanings.

vs.

Retrieval performance (relevance) is judged according to *meaning*.

Hence, need of *knowledge*.

7. Administrativa

Check the calendar: are there any days in which you know you cannot come. I am inviting expert on specific topics to give talks.