Information Retrieval
Notes

Andrea Mansi UniUD

II° Semestre 2020/2021
Big Data Analytics

Contents

Infos about the notes

Questi appunti (**non ufficiali**) si pongono l'obiettivo di riassumere i principali concetti teorici trattati nel corso di Information Retrieval (Prof. Stefano Mizzaro - Università degli studi di Udine). Gli appunti sono basati sulle lezioni e sulle slide del corso. Alcune parti potrebbero essere trascurate, pertanto tali appunti non sono da considerarsi un'alternativa alle lezioni, bensì un utile dispensa per un veloce ripasso.

These (**unofficial**) notes aim to summarize the main theoretical concepts covered in the Information Retrieval course (Prof. Stefano Mizzaro - Università degli studi di Udine). These notes are based on lectures and course slides. Some parts could be incomplete, therefore they are not to be considered an alternative to the lessons, but a useful tool for a quick review.

Last update: February 20, 2022

1 Introduction

Let's start with a simple definition of **Information Retrieval (IR)**:

> Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfy an information need from within large collections (usually on local computer servers or on the internet).

Information Retrieval is not that simple... there are some problems, for example, consider the following paradox:

Meno's Paradox:

> ...and how will you inquire into a thing when you are wholly ignorant of what it is? Even if you happen to bump right into it, how will you know it is the thing you didn't know?

A more formal reformulation of this statement has been formulated by Socrates:

> (A) man cannot search either for what he knows or for what he does not know. He cannot search for what he knows–since he knows it, there is no need to search–nor for what he does not know, for he does not know what to look for.

IR systems usually work on lots of data (for example the web which has more than 10 billion pages), using query languages and giving ranked results, according to some **relevance** metric.

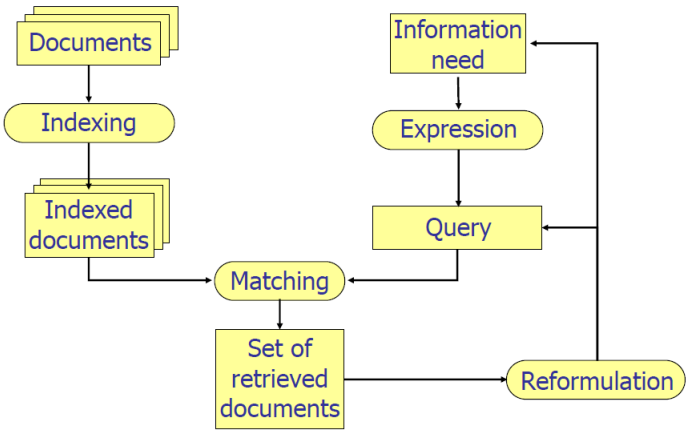
It's worth noting that IR is very similar but at the same time different in one key aspect from the database field: in databases data is heavily structured.

We can summarize differences between the concept of **data retrieval** and **information retrieval** with this table:

	Data Retrieval	**Information Ret.**
Match	Exact	Partial, best
Model	Deterministic	Probabilistic
Query language	Artificial	Natural
Query	Complete	Incomplete
Searched items	Those that "match"	The "relevant" ones
Results	Set	Ranked set
Need	Precise	Confused

1.1 What is an IR system?

We have an IR System (**IRS**) and a user with an **information need**. The user submits a **query** in order to gather the needed informations. The IRS **retrieves** some documents (usually ranked by decrasing relevance) from a **repository** of documents (information items). The user analyzes the IRS output. He can reformulate the query and modify the information need. Those steps can be summarized with the **classical IR model:**



What's not IR: Natural language processing, question answering (IR returns documents, not answers), database (recall of information vs data).

1.2 Characteristics of an Information need

An information need is composed by three main components:

- **Topic:** what the documents should be about (the ones I'm searching);
- **Task:** what I have to do with the informations from the documents;
- **Context:** everything else, like what I already know, how much time I have, etc.

Those three components are also valid for documents and informations. We can also include features like high recall/precision; well or badly defined; real or perceived/expressed.

1.3 Two levels of Information Retrieval

IR can be performed at "two level": we can work at the cognitive or syntactic level:

- **Cognitive, Semantic**: ideal, but difficult: IRS works on understanding the user's need. We don't know how to do that...
- **Syntactic**: feasible, easier to implement: IRS works on term counting/crunching. Working on the given available "evidence".

2 Search user interfaces

The job of the **search user interface** is to aid users in the expression of their information needs, in the formulation of their queries, in the understanding of their search results, and in keeping track of the progress of their information seeking efforts. However, the typical search interface today is of the form: type-keywords-in-entry-form, view-results-in-a-vertical-list.

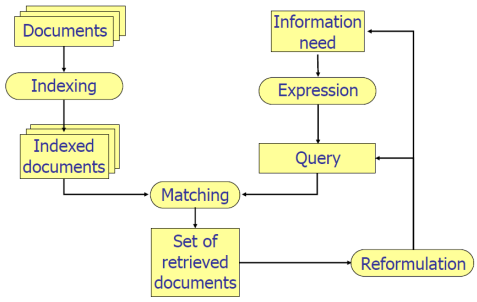
Some important reasons for the relative simplicity and unchanging nature of the standard Web search interface are:

- Search is a means towards some other end, rather than a goal in itself. When a person is looking for information, they are usually engaged in some larger task, and do not want their flow of thought interrupted by an intrusive interface.
- Related to the first point, search is a mentally intensive task. When a person reads text, they are focused on that task; it is not possible to read and to think about something else at the same time. Thus, the fewer distractions while reading, the more usable is the interface.
- Since nearly everyone who uses the Web uses search, the interface design must be understandable and appealing to a wide variety of users of all ages, cultures and backgrounds, applied to an enormous variety of information needs.

2.1 Models of the Information Seeking Process

In order to design successful search user interfaces, it is necessary to understand the human information seeking process, including the strategies people employ when engaged in search.

Classic IR model: Many accounts of the information seeking process assume an interaction cycle consisting of identifying an information need, followed by the activities of query specification, examination of retrieval results, and if needed, reformulation of the query, repeating the cycle until a satisfactory result set is found.



This is the **classic IR model** already seen in the intro. The classic model cycle can be summarized as composed by the following 4 steps:

- Problem identification,
- Articulation of information need(s),

- Query formulation, and
- Results evaluation.

The standard model of the information seeking process contains an underlying assumption that the user's information need is static and the information seeking process is one of successively refining a query until all and only those documents relevant to the original information need have been retrieved. However, observational studies of the information seeking process find that searchers' information needs change as they interact with the search system. Searchers learn about the topic as they scan retrieval results and term suggestions, and formulate new subquestions as previously posed subquestions are answered. In contrast, a dynamic model is formulated: the **Berry-Picking Model** [Bates].

The model is based on two main points:

- The first is that, in the process of reading and learning from the information encountered throughout the search process, the searchers' information needs, and consequently their queries, continually shift. Information encountered at one point in a search may lead in a new, unanticipated direction. The original goal may become partly fulfilled, thus lowering the priority of one goal in favor of another.
- The second point is that searchers' information needs are not satisfied by a single, final retrieved set of documents, but rather by a series of selections and bits of information found along the way. This is in contrast to the assumption that the main goal of the search process is to hone down the set of retrieved documents into a perfect match of the original information need.

The underlying metaphor is: berries do not come all together - they are scattered in bushes and need to be picked up one at a time.

2.2 Query Specification

In the query specification part of the information access process, the searcher expresses an information need by converting their internalized, abstract concepts into language, and then converting that expression of language into a query format that the search system can make use of. Queries can be grouped into three main groups:

1. **Keyword based query:** single words, contextual, boolean, natural language;
2. **Pattern matching:** prefix, suffix, substring, range, with errors;
3. **Structural:** based on the structure of text.

Keyword queries consist of a list of one or more words or phrases whose intention is to find documents containing those words that are likely to be relevant to the user's information need. The main issues with this type of queries is that it is difficult to understand what is a word or a separator and that statistics on word occurences influence the results.

- Single words: documents that contain at least one of the query terms;
- Contextual: to find words in a context, i.e., near other words. Those type of queries are mainly grouped into 2 kinds: search phrases, which means "sequence of n words" and to search by proximity, which means sequences of words or phrases within a maximum distance.
- Boolean: words with boolean operators. If pure boolean, it is difficult for the end user: there is no ranking and it is difficult to understand.
- Natural language: it is an extension of boolean, but understanding a NL query is very difficult for the system. Eventually it needs to be converted in terms and weights. It is more natural for the user.

Pattern matching queries examples:

- Prefix: infor* = information, informative, ...
- Suffix: *ter = computer, painter, smarter, ...
- Substring: *trie* = retrieval, retrieving, trie, ...
- Range: from A to B (example: dates)
- With errors: query misspellings, DNA sequences, collection (OCR), ...
- Regular expressions

Structural queries are based on text structure, for example:

- Fixed: search on specific fields (email, google scholar advanced, etc.)
- Hierchical: XML retrieval, etc.
- Hypterextual: Graph, links, etc.

Other types of query can be QBE queries (query by example), spoken queries, long queries; much richer than the simple query box.

3 IR Models

An Information Retrieval model specifies three main aspects:

- **Retrieve**: how to select the retrieved documents;
- **Rank**: how to rank the retrieved documents;
- **Query**: how to specifiy the query.

Basic IR classification of IR models:

- **Classic**: Boolean, Vector space, Probabilistic;
- **Evolutions**: Set based (Extended Boolean, Fuzzy), Algebraic (Generalized vector space, Latent Semantic Indexing, Neural networks), Probabilistic (BM25, Language models, Bayesian models).

We now introduce some common definitions ("base-line terminology") for further formulas and models:

- d_j = j-th document;
- N = number of documents in the collection;
- *Index term*: if it is or not in the index;
- t = number of terms in the index term;
- k_i = i-th term in the index (keyword);
- $K = k_1, k_2, ..., k_t$ = set of index terms;
- $w_{ij} \geq 0$ = weight ok k_i in d_j.

Let's see a basic **index** implementation:

		k_1	k_2	k_3	…	k_t	
d_1	(	1	0	0	…	1	)
d_2	(	1	1	0	…		)
d_3	(	0	0	1	…	0	)
…		…	…	…	…	…	
d_N	(	1	0	0	…	1	)

query

q	(	1	0	0	…	1	)

1 means the term is in the document and 0 means the opposite. The problem with this basic implementation is that the matrix is huge and very sparse (a lot of zeros). It is just a conceptual description, not feasible in practice.

The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would need to scan every document in the corpus, which would require considerable time and computing power.

3.1 The Boolean model

The boolean model is based on set theory and boolean algebra. In this model the query can contain logical connectives such as AND, OR, NOT etc. A document is retrieved if it "satisfies" the query. Example:

d_1: "Shipment of **gold** damaged in a fire"
d_2: "Delivery of **silver** arrived in a **silver truck**"
d_3: "Shipment of **gold** arrived in a **truck**"

q_1: "gold AND silver AND truck" $\rightarrow \varnothing$
- ($sim(d_j,q)$ is 0 for the three docs.)

q_2: "(gold OR silver) AND truck" $\rightarrow d_2\, d_3$
- ($sim(d_j,q)$ is 0 for d_1 and 1 for d_2 and d_3)

q_3: "gold AND truck" $\rightarrow d_3$

q_4: "gold AND truck AND NOT silver" $\rightarrow d_3$

Pros:

- Simple, crystal-clear;
- Binary decision criterion: the collection is divided into 2 parts (retrieved and not retrieved).

Cons:

- Difficult for the average end-user. Beginner users sometimes use AND where OR is required (or viceversa). Example: in the following query: "I'm interested in chess and boxing" OR should be used.
- No ranking of the retrieved documents: The user is not informed about more/less relevant docs; tipically too many/few docs. This is because a document is only relevant or non-relevant (binary decision criterion).
- Exact match: documents partially satisfying the query are not retrieved.

A model in which weights are not binary, would allow partial matches, query-document degree of similarity and the construction of a ranked set of retrieved documents. Vector space model allows that. Let's see it.

3.2 Vector Space Model

The basic idea consist of having a t-dimensional vector space. One dimension corresponds to one term (assumed alphabetially sorted). A document is a vector in the vector space.

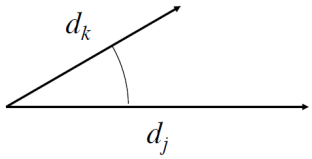
$$d_j = (w_{1,j}, w_{2,j}, w_{3,j}, \ldots w_{t,j})$$

$$w_{i,j} = \begin{cases} 0 \; iff \; k_i \notin d_j \\ 1 \; iff \; k_i \in d_j \end{cases}$$

		k_1	k_2	k_3	…	k_t	
d_1	(	1	0	0	…	1	)
d_2	(	1	1	0	…		)
d_3	(	0	0	1	…	0	)
…		…	…	…	…	…	
d_N	(	1	0	0	…	1	)

This model is based on some assumptions: Vectors (associated to terms) are an orthonormal basis of the vector space (i.e. they are **independent** and terms occur in docs. independently from each other) - which is obviously not true (no proof that it is a practical problem).

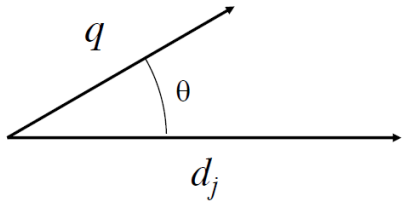
With the vector space model we can derive the similarity between two vectors. Similar vectors means similar terms which means similar documents. We can use the cosine similarity.



The query is a vector as well so given a query, documents can be ranked in decreasing similarity order. In formulas:

$$d_j = (w_{1,j}, w_{2,j}, \ldots w_{t,j})$$

$$q = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$$



$$sim(d_j, q) = \frac{\sum_{i=1}^{t} \left(w_{i,j} \cdot w_{i,q} \right)}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Cosine Sim

$$\frac{\sum_{k=1}^{n} A(k)B(k)}{\sqrt{\sum_{k=1}^{n} A(k)^2} \sqrt{\sum_{k=1}^{n} B(k)^2}}$$

Normalization

Let's analyze the *sim* similarity function. It goes from 0 to 1. A document can be retrieved even if it does not contain all the query terms: no more exact match and more and less similar documents are retrieved too. Denominator:

- Divide by vector norms to normalize, otherwhise longer docs. are always more relevant.
- By normalizing, all vectors length are the same.

Weights can be binary but in some docs. the weight of a term might be higher than in other docs. (and with binary weights this information is lost). But the model still works if weights are from 0 to 1. But how to choose those weights? There are two intuitions/criteria:

1. If a term occurs many times in a document, it will have a higher weight than a term occuring just a few times;
2. If a term occurs in many documents, it will have a lower weight than a term occuring in just a few documents, because it's a common terms along the collection.

If a term k_j occurs in many docs., the fact that it occurs in a gived doc. d_j is not so important. k_j is not very important to discriminate d_j from the other docs. To the limit a term occuring in all docs. is useless and whereas a term occurring in a single document is very useful. Thus in the first criteria weights will be proportional to the frequency of the term in a document. In the second criteria the weights will be inversely proportional to the number of documents containing that term.

We can combine the two criteria: **tf.idf**

3.2.1 TF-IDF:

let's see the formulas:

Criterion 1:
Term frequency (*tf*)

$$tf(i,j) = \frac{freq(i,j)}{\max_{k_l \in d_j} freq(l,j)}$$

Criterion 2:
Inverse Document Frequency (*idf*)

$$idf(i) = \log \frac{N}{n_i}$$

Where:

- $freq(i,j)$ = # of occurrences of k_i in d_j
- N = # of docs. in the collection
- n_i = # docs. containing k_i

$$w_{i,j} = tf(i,j) \cdot idf(i) = \frac{freq(i,j)}{\max_{k_l \in d_j} freq(l,j)} \cdot \log \frac{N}{n_i}$$

Log is in base 10 (the base is not very important). It weakens the IDF effect. TF varies over docs. while IDF is the same for all docs. (it depends on the entire collection).

Weights for query terms are little different. They are similar to those for docs. but with some normalization/optimization. If $n_i = 0$ for a term (new term is present in the query, new because there are no docs. already containing it), then the value would be infinite, but that term would not occur in the collection, and therfore in the vector q. The problem is fixed with the following formula version:

$$w_{i,q} = \left(0.5 + \frac{0.5 \cdot freq(i,q)}{\max\limits_{k_l \in q} freq(l,q)} \right) \cdot \log \frac{N}{n_i}$$

Pros:

- Non binary weights: better topic representation;
- Partial match: can retrieve docs. that match the query only partially;
- Ranking of the retrieved docs;
- Better effectiveness than Boolean.

Cons:

- Theoretical vectors/term indipendence assumption (we don't know if it has effects in practical use);
- Curse of dimensionality.

In general it is a good model which is still in use.

3.3 Probabilistic model (BIM)

Given a fixed query q the probabilistic model estimates, for each document in the collection, the probability that it is relevant on the basis of the available evidence: terms in the docs. (and in all the docs.), terms in the query, other data, application of bayes rule, etc.

Here, that probability is a mean, not the end probability, and it is used to rank the docs.

The ranking is based on the probability ranking principle: If the aim is to sort the documents then the ranking by decreasing relevance probability is the best one. But any monotic transformation of probability does not affect the ranking (they preserve the ranking). There is an hidden assumption: relevance is binary (either a doc. is relevant or it is not). Given a fixed query we can define:

Generic probability of relevance:

- The probability that a document is relevant;
- Without any further knowledge; let's call it $p(R)$.

Probability of relevance of a given document d_j:

- The probability that a specific doc. d_j is relevant;
- Let's call it $p(R|d_j)$;
- What we would like to know: if we have it for any doc. then we should be able to rank them.

If I draw a random document from the collection, what is the probability that I find a specific document d_j? It is $p(d_j)$ and it is easy to estimate but not much interesting.

If I draw a random document from the relevant ones, what is the probability that I find a specific document d_j? It is $p(d_j|R)$ which is also estimable with some assumptions.

We can define a similarity measure. Given a query q we can define sim as:

$$sim(d_j,q) = \frac{p(R \mid d_j)}{p(\overline{R} \mid d_j)}$$

We prefer docs. with high probability of relevance and low probability of non-relevance. We should remark that similarity is a wrong name, we are interested in the ranking. The standard term is RSV (Retrieval Status Value). So we want to compute sim given a query for all documents. Of course we don't know $P(R|d_j)$ and $p(\bar{R}|d_j)$ but we can apply the Bayes theorem:

$$p(C \mid I) = \frac{p(C) \cdot p(I \mid C)}{p(I)}$$

By applying the Bayes's theorem we get:

$$p(R \mid d_j) = \frac{p(R) \cdot p(d_j \mid R)}{p(d_j)} \qquad p(\overline{R} \mid d_j) = \frac{p(\overline{R}) \cdot p(d_j \mid \overline{R})}{p(d_j)}$$

$$sim(d_j,q) = \frac{p(R \mid d_j)}{p(\overline{R} \mid d_j)} = \frac{\dfrac{p(R) \cdot p(d_j \mid R)}{p(d_j)}}{\dfrac{p(\overline{R}) \cdot p(d_j \mid \overline{R})}{p(d_j)}} = \frac{p(R)}{p(\overline{R})} \cdot \frac{p(d_j \mid R)}{p(d_j \mid \overline{R})}$$

Odds allow us to simplify the fraction: this is why $p(d_j)$ is not important.

Thus we have:

$$sim(d_j,q) = \frac{p(R)}{p(\overline{R})} \cdot \frac{p(d_j \mid R)}{p(d_j \mid \overline{R})}$$

which makes sense, at least intuitively:

- $p(R)$ is the probability that a doc. is relevant, with no further info;
- $p(d_j|R)$ is the probability that drawing a random doc. from the relevant ones I find exactly d_j;
- The same for $p(\bar{R})$ and $p(d_j|\bar{R})$.

But $p(R)$ and $p(\bar{R})$ are independent from the single doc. (given a query q they are constant for all the documents) so we can simplify them. Then we get:

$$sim(d_j,q) \cong \frac{p(d_j \mid R)}{p(d_j \mid \overline{R})}$$

We still have to understand how to estimate $p(d_j|R)$ and $p(d_j|\bar{R})$. If we had R it would be easy but it is not known. It is what we're trying to find. Moreover, even if we knew that a given doc. d_j is in R, this is not helpful to know if $\forall x, d_x$ in R.

3.3.1 What is a document? A document d_j has some particular features. For example, its terms (but they could be anything: colors, length of a song, etc.):

$$d_j = \{k_1, k_2, k_3, \ldots\}$$
$$p(d_j \mid R) = p(\{k_1,k_2,k_3,\ldots\} \mid R)$$
$$p(d_j \mid \overline{R}) = p(\{k_1,k_2,k_3,\ldots\} \mid \overline{R})$$

If I draw from the relevants, to find d_j is equal to find a document with the features of d_j. The probabilistic model is based on **two assumptions**: features are binary and independent.

$$\text{Binary: } k_i = \begin{cases} 0 \text{ iff } k_i \notin d_j \\ 1 \text{ iff } k_i \in d_j \end{cases}$$

Either a document has a feature or it has not. Independent: k_i and k_x with $i \neq x$ are independent.

Independence allows to break down the probability into a product of probabilities:

$$\begin{aligned} p(d_j \mid R) &= p(\{k_1,k_2,k_3,\ldots\} \mid R) = \\ &= p(k_1 \mid R) \cdot p(k_2 \mid R) \cdot p(k_3 \mid R) \cdot \ldots = \\ &= \prod_i p(k_i \mid R) \end{aligned}$$

Which means that:

- $p(k_i|R)$: if I draw a document from the relevant, what is the probability that it has feature k_i?
- In place of computing $p(d_j|R)$ we can compute several $p(k_i|R)$ in place of a single $p(d_j|R)$.
- $p(d_j|\bar{R}) \rightarrow p(k_i|\bar{R})$.

Let's now talk about the estimation of $p(k_i|R)$. If k_i is one of d_j features (i.e. k_i is part of d_j), we can ask: what is the probability $p(k_i|R)$ that a relevant document has the k_i feature (e.g. the k_i term)? This is as easy as $p(d_j|R)$: we simply count among the relevant docs. Whereas $p(k_i|R)$ can be used for all the documents having the feature k_i.

If we now come back to $sim(d_j|q)$ we have:

$$sim(d_j,q) \cong \frac{p(d_j \mid R)}{p(d_j \mid \overline{R})} = \frac{\prod_i p(k_i \mid R)}{\prod_i p(k_i \mid \overline{R})} = \prod_i \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})} = \prod_{k_i \in q} \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})} \cdot \cancel{\prod_{k_i \notin q} \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})}} = \ldots.$$

In which we can split the π by distinguishing $k_i \in q$ and those $\notin q$.

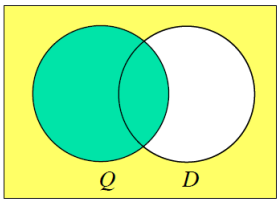
If a term does not occur in the query we assume that the probabilities that it occurs in relevant and nonrelevant docs. are equal for all the documents $d_j : p(k_i|R) = p(k_i|\bar{R})$. So we take into account only the query terms.

If we take into account also d_j we will have:

$$\ldots. = \prod_{k_i \in q} \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})} = \prod_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})} \cdot \prod_{\substack{k_i \in q \wedge \\ k_i \notin d_j}} \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})} = \ldots.$$

By splitting again the π by distinguishing terms in the doc. and not in the doc. ($\bar{k}_i$)) we have:

$$.... = \prod_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})} \cdot \prod_{\substack{k_i \in q \wedge \\ k_i \notin d_j}} \frac{p(\overline{k}_i \mid R)}{p(\overline{k}_i \mid \overline{R})} = ...$$



If now we multiply and divide:

$$= \prod_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \frac{p(k_i \mid R)}{p(k_i \mid \overline{R})} \cdot \prod_{\substack{k_i \in q \wedge \\ k_i \notin d_j}} \frac{p(\overline{k}_i \mid R)}{p(\overline{k}_i \mid \overline{R})} \cdot \prod_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \frac{p(\overline{k}_i \mid \overline{R})}{p(\overline{k}_i \mid R)} \cdot \prod_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \frac{p(\overline{k}_i \mid R)}{p(\overline{k}_i \mid \overline{R})} =$$

$$= \prod_{k_i \in q \cap d_j} \frac{p(k_i \mid R) \cdot p(\overline{k}_i \mid \overline{R})}{p(k_i \mid \overline{R}) \cdot p(\overline{k}_i \mid R)} \cdot \prod_{k_i \in q} \frac{p(\overline{k}_i \mid R)}{p(\overline{k}_i \mid \overline{R})} =$$

Now we simplify:

$$.... = \prod_{k_i \in q \cap d_j} \frac{p(k_i \mid R) \cdot p(\overline{k_i} \mid \overline{R})}{p(k_i \mid \overline{R}) \cdot p(\overline{k_i} \mid R)} \cdot \cancel{\prod_{k_i \in q} \frac{p(\overline{k_i} \mid R)}{p(\overline{k_i} \mid \overline{R})}} = ...$$

The last fraction is independent from the specific document, we can neglect it. What we get is the final version of the *sim* function (without transformations):

$$sim(d_j, q) \cong \prod_{k_i \in q \cap d_j} \frac{p(k_i \mid R) \cdot p(\overline{k_i} \mid \overline{R})}{p(k_i \mid \overline{R}) \cdot p(\overline{k_i} \mid R)}$$

Actually, we can apply one last step: logarithm transformation (useless logically, needed only for implemenation reasons: precision of floating point operations). We need the ranking only and $log(x)$ is monotic. We take the log of the *sim* function.

$$\begin{aligned} sim(d_j, q) &\cong \log \prod_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \frac{p(k_i \mid R) \cdot p(\overline{k_i} \mid \overline{R})}{p(k_i \mid \overline{R}) \cdot p(\overline{k_i} \mid R)} = \\ &= \sum_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \log \frac{p(k_i \mid R) \cdot p(\overline{k_i} \mid \overline{R})}{p(k_i \mid \overline{R}) \cdot p(\overline{k_i} \mid R)} \end{aligned}$$

The real **final version** of the *sim* function is:

$$sim(d_j, q) \cong \sum_{\substack{k_i \in q \wedge \\ k_i \in d_j}} \log \frac{p(k_i \mid R) \cdot p(\overline{k_i} \mid \overline{R})}{p(k_i \mid \overline{R}) \cdot p(\overline{k_i} \mid R)}$$

Intuitively it makes sense. It $\uparrow$ if:

- numerator $\uparrow$ if:
  - $p(k_i|R) \uparrow$ (it is probable that term is in docs. R)
  - $p(\bar{k}_i|\bar{R}) \uparrow$ (it is probable that term is not in docs. $\bar{R}$)
- denominator $\downarrow$ if:
  - $p(k_i|\bar{R}) \downarrow$ (it is probable that term is in docs. $\bar{R}$)
  - $p(\bar{k}_i|R) \downarrow$ (it is probable that term is not in docs. R)

It is based on the terms/features in query and docs. Remarks: It is not a proper similarity, we use it to rank the documents and it is derived formally, it has a mathemathical foundation.

But how do we compute the 4 probabilities? which actually are two:

$$p(\overline{k_i} \mid R) = 1 - p(k_i \mid R)$$
$$p(\overline{k_i} \mid \overline{R}) = 1 - p(k_i \mid \overline{R})$$

Two options: at start (with no info about relevance) or after a relevance feedback.

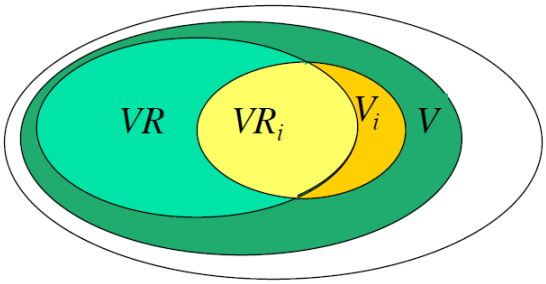
At start means right after query q specification. The probability that a relevant document contains k_i is $p(k_i|R)$ is unknown but the simplest and reasonable approximation is $p(k_i|R) = 0.5$. The probability that a nonrelevant doc. contains k_i is also unknown and approximated as $p(k_i|\bar{R} = \frac{n_i}{N})$ with n_i = number of docs. containing k_i. Using those approximations we have a starting point for the ranking. Then user can communicate the relevance of some documents (relevance feedback). This means new evidence: we can revise the two estimations. We're using user's info to revice/improve estimations. There are several possibilities. The most simplest one is the following:

User evaluates (first) V documents, some as relevant, some as nonrelevant.

- V = number of docs. evaluated
- VR = docs. evaluated as relevant
- V_i = docs. evaluated containing k_i
- VR_i = docs. evaluated as relevant and containing k_i

Then we have:

$$p(k_i \mid R) = \frac{VR_i}{VR}$$
$$p(k_i \mid \overline{R}) = \frac{V_i - VR_i}{V - VR}$$



We have a problem if $VR = 0$ or $V - VR = 0$. We can fix it with some math tricks.

$$p(k_i \mid R) = \frac{VR_i}{VR}$$
$$p(k_i \mid \overline{R}) = \frac{V_i - VR_i}{V - VR}$$

$$\Rightarrow$$

$$p(k_i \mid R) = \frac{VR_i + 0.5}{VR + 1}$$
$$p(k_i \mid \overline{R}) = \frac{V_i - VR_i + 0.5}{V - VR + 1}$$

Actually, user relevance feedback is not strictly necessary. In fact, we can use pseudo relevance feedback: we simply assume that the first n docs. are relevant. This might improve the estimation with no user action. Now pros and cons about the BIM.

Pros:

- Based on a solid mathematical theory (probability theory)
- Includes user interaction (relevance feedback)
- Good effectiveness

Contra:

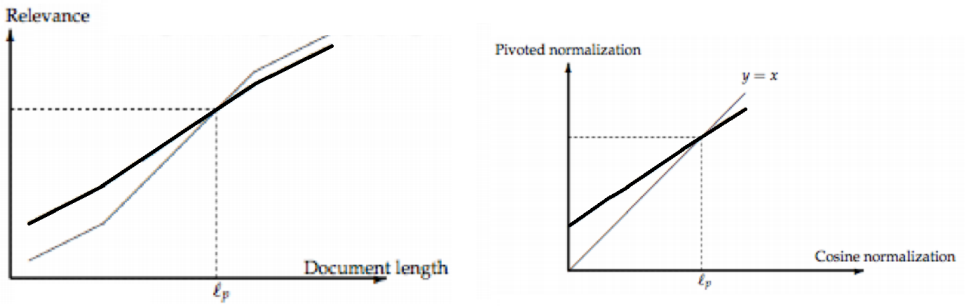
- Difficult initial probabilities estimation: tf can be used
- Independence assumption: but it is not certain it is a real problem

3.4 Document length normalization

Even if models take document length into account to some extent (e.g. normalizations in vector space) there are some undesirable effects related to length of documents. In other terms, models do take into account the length of documents but it can be done better.

3.4.1 Pivoted document length normalization

Let's analyze those two images:



To the left we have:

- **thick line**: cosine similarity as function of doc. length
- thin line: true relevance as function of doc. length
- Those are experimental/empirical results
- Problem: relevance of short/long docs. is over/under estimated

To the right we have:

- A "pivoted" (rotated) sim function
- The effectivness is increased and this is proved by experimental results.

This concept is on the base of the BM25 model (Best Match 25, 1994)

3.5 BM25 (Best Match 25 - 1994)

Let's start with the BM25 formula of similarity:

$$sim(Q,D) = \sum_{k_i \in Q} \left(\log \frac{p(k_i \mid R) \cdot (1 - p(k_i \mid \overline{R}))}{p(k_i \mid \overline{R}) \cdot (1 - p(k_i \mid R))} \cdot \frac{(\alpha + 1) f_i}{\gamma + f_i} \cdot \frac{(\beta + 1) qf_i}{\beta + qf_i} \right)$$

f_i	Frequency of term k_i in the document
qf_i	Frequency of term k_i in the query
α	Determines the contribute of tf component when varying f_i
β	Similar to α, but works on qf_i frequencies
γ	$\alpha\left((1-b) + b \cdot \frac{dl}{adl}\right)$ $b = 0 \rightarrow$ No document length normalization $b = 1 \rightarrow$ "Total" normalization
dl / adl	Doc. length / Average doc. length

Tipical paramenter values: α=1.2; β=100; b=0.75

Meaning of BM25: BIM + TF (document and query) + DL normalization.

$$sim(Q,D) = \sum_{k_i \in Q} \log \frac{p(k_i \mid R) \cdot (1 - p(k_i \mid \overline{R}))}{p(k_i \mid \overline{R}) \cdot (1 - p(k_i \mid R))} \cdot \frac{(\alpha + 1) f_i}{\gamma + f_i} \cdot \frac{(\beta + 1) qf_i}{\beta + qf_i}$$

TF factor and DL

TF factor in the qry

The first factor is the basic BIM formula. Then we have the TF/DL and query TF factors.

If $b = 0$ we have:

- TF factor:
  - Increases as f_i increases
  - Limited increase, log
  - $\alpha >>$ then TF importance $>>$
- TF factor in the query: about the same

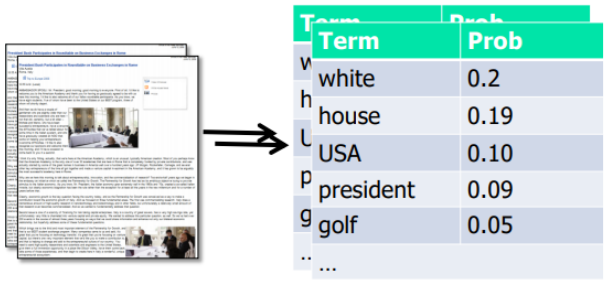
If $b > 0$ we have:

- Also doc. length normalization
- dl/adl: max for longest doc. then decreasing

3.6 Language models

A language model is a function that assigns probability values to term sequences.

The simplest: start from p(single term), then product. We have one LM per document.



Term	Prob
white	0.2
house	0.19
USA	0.10
president	0.09
golf	0.05
...	

How can we compute *sim* in LM? there are several possibilities:

1. **Generation based:**
   - Generate a set of terms, randomly, according to a language model
   - 1.a. Probability that the document LM generates the query
   - 1.b. Probability that the query LM generates the document
2. **Distance based:**
   - Difference/distance between document LM and query LM

3.6.1 1.a. Query likelihood ranking

The formula:

$$sim(Q,D) = P(Q \mid D) = \prod_{q_i \in Q} P(q_i \mid D) \approx \sum_{q_i \in Q} \log(P(q_i \mid D))$$

$$P(q_i \mid D) = \frac{freq(q_i, D)}{length(D)}$$

$P(q_i|D)$ is the probability that the LM given by document D assigns to the term q_i. Problem: If a query term q_i does not occur in document D $P(q_i|D) = 0$ and so is the π (and log is minus infinite). Even if the other query terms occur in D. The solution is called "smoothing". We compute $P(q_i|D)$ by combining:

- The probability given by its frequence in D;
- The probability given by its frequence in the whole collection C
- So it is never 0

$$P(q_i \mid D) = (1-\alpha)\cdot\left(\frac{freq(q_i \mid D)}{length(D)}\right) + \alpha\cdot\left(\frac{freq(q_i \mid C)}{length(C)}\right)$$

- α = constant (Jelinek-Mercer smoothing)
- α = f(|Q|) (0.1 for short queries, 0.9 for long ones)
- $\alpha = \frac{\mu}{\mu + |D|}$ (Dirichlet smoothing, μ=1000-2000)

There are also more sophisticate criteria.

3.6.2 2. Difference/Distance between LMs Difference/distance between document LM and query LM (distance between the two probability distributions). For example using the **Kullback-Leiber** divergence.

$$KL(P|Q) = \sum_x P(x) log(\frac{P(x)}{Q(x)})$$

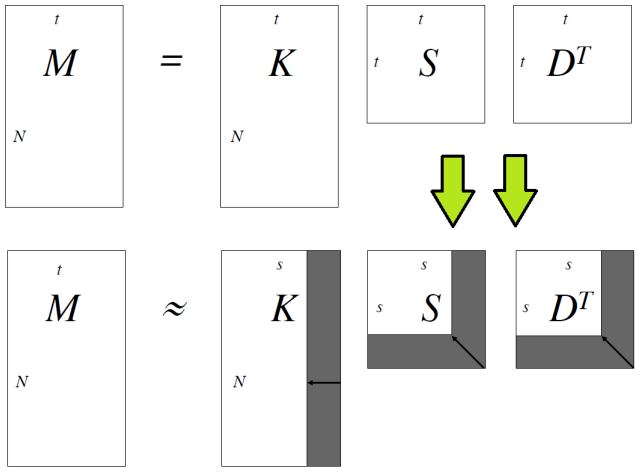
3.7 SVD (Singular Value Decomposition)

We have seen how to build (conceptually) the index: a matrix of weights which is very large and inefficient. However, from a theorem of linear algebra, we have:

$$M = K * S * D^T$$

where:

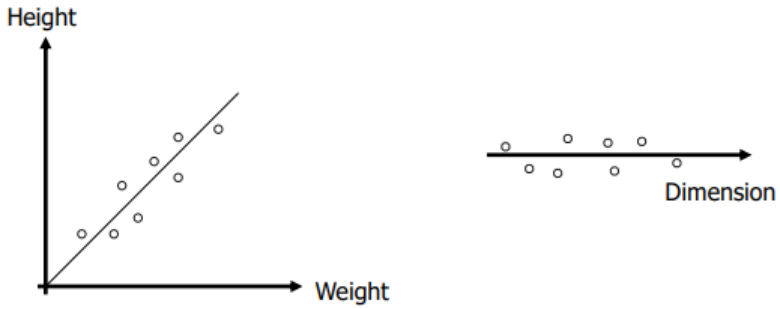
- D^T is the traspose matrix of D
- S is a diagonal matrix, $r * r$ with decrasing values and r is $\min(t, N)$
- K and D are the matrices of eigenvectors of $M * M^T$ and $M^T * M$ respectively



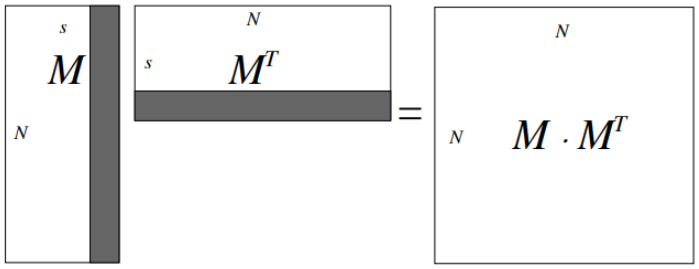
We can approximate M by removing the smallest elements in S keeping only the largest s ones.

3.7.1 LSI - Latent Semantic Indexing

By using LSI (to use M_s - the approximated matrix - instead of M) we have the certain advantages of reduced dimensionality, higher efficency (storage, faster operations, ...) and the hope of reduced space that is more conceptual and less syntatic; resolves or at least relieves the problems of synonymi and polysemy; concepts in place of terms. M_s has fewer columns: in place of the t terms of M we have s "concepts", for example:



So, with $M_s * M_s^T$ we have concepts similarity (document similarity). As usual we can consider the query as a document; for example with document 0, the first row of the matrix $M_s * M_s^T$ gives us the similarity of each doc. with the query.



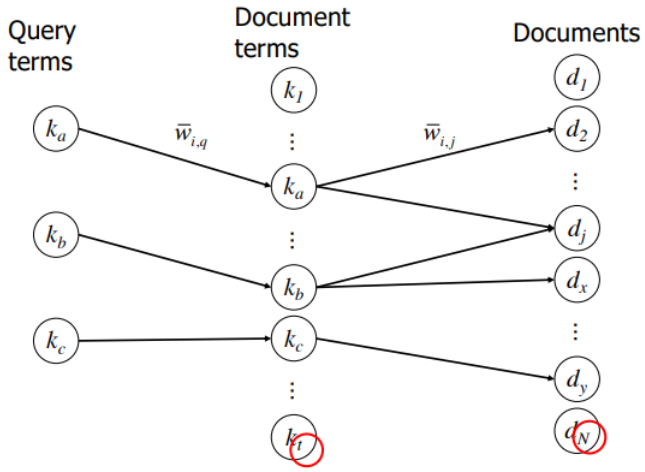
As for performance of the LSI, it is not yet clear if is more effective than a classical approach (apparently it is). The problem is how to choose s.

3.8 Neural Networks

Neural networks are good in pattern matching/classification. They learn to recognize a certain pattern. In IR what is needed is to match documents and query, and documents have to be classified as relevant or irrelevant.

The idea is that:

- There is a link from a **query term node** to a **document term node** only if they are equal.
- There is a link from a **document term node** to a **document node** only if the document contains that term.
- The **query term nodes** fire, activating the **documents term nodes**.
- The **document term nodes** fire, activating the **documents nodes**.
- The **documents nodes** that are activated are the retrieved documents.



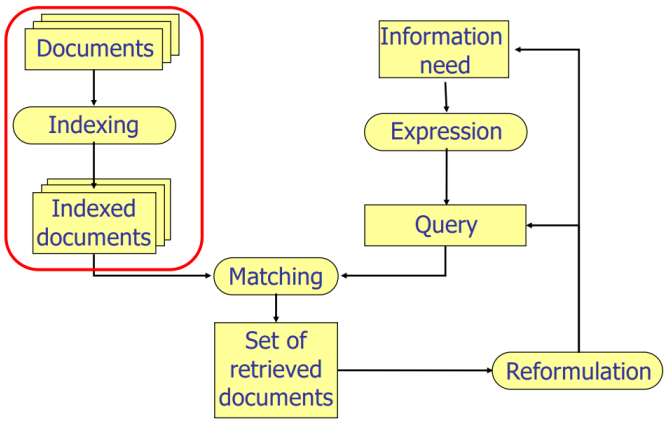
With the weights, a document (node) might be not activated even if it contains the query terms (with some values for weights and thresholds). In this way, the document nodes are activated with a weight equal to tf.idf.

Thus, so far, the model is equivalent to vector space model, but there are bidirectional links, therefore it is possible that among the activated nodes there are terms not in the query and documents that do not contain the query terms.

It is similar to pseudo relevance feedback. This model includes relevance feedback in a natural way: when a document is judged as relevant (by the user), the weights of the outgoing links can be increased, thus, possibly, activating other terms.

4 From documents to the index, preprocessing

We will now cover some properties of natural language text, basis for index implementations and justifications for some choices in IR models. Considering the classical IR conceptual model, the highlated area is the one of interest for those topics.



4.1 Properties of natural language text

First of all we can say that a text is a sequence of letters with non-uniform distribution. Some letters are more frequent then others (e.g. vowels). We can also notice that different languages have different distributions.

Each text can be seen as generated by a model (generative models):

- **Binomial:** each single letter is generated with a probability that corresponds to its frequence;
- **Markovian:** the probability of a letter depends on previous letters (Markov models).

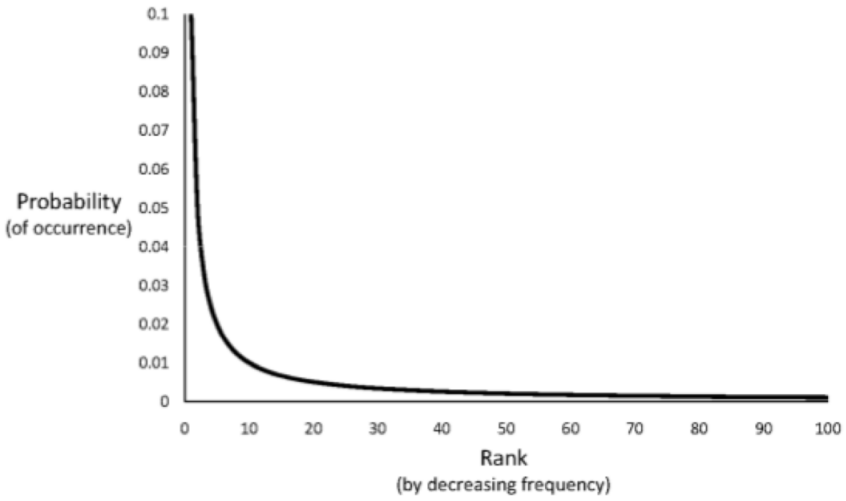
However, letters are not much interesting to us. We can use the markov models at world level: the probability of a certain word given the k previous words can be considered. Both models are still based on statistics and they don't consider semantic.

4.1.1 Zipf's law

The word frequency distribution follow the Zipf's law which is a power-law. Ranking the words from most frequent to less frequent, frequency of each word is given by:

$$F(r) = \frac{C}{r^{\alpha}}, \qquad \alpha \approx 1,\ C \approx 0.1$$

- r is the rank (position in the ranking)
- α and C are constants



The frequency of the r-th word is proportional to about $1/r$. A few words occur often; many words occur just a few times.

We can plot the zipf's law on log-log scale in order to transform it in a linear fashion.

It has been experimentally proven that Zipf's law holds for different texts, author, languages, both when using terms and when using stems, on the web, for population dynamics, etc.

4.1.2 Vocabulary size - Heaps'law

The vocabulary is the set of unique terms in a given collection. How does vocabulary size increase with collection size? If new documents are added to the collection, how many new terms are added to the vocabulary? Intuitive answer: at start, many terms; then fewer. More precise answer: Heaps'law.

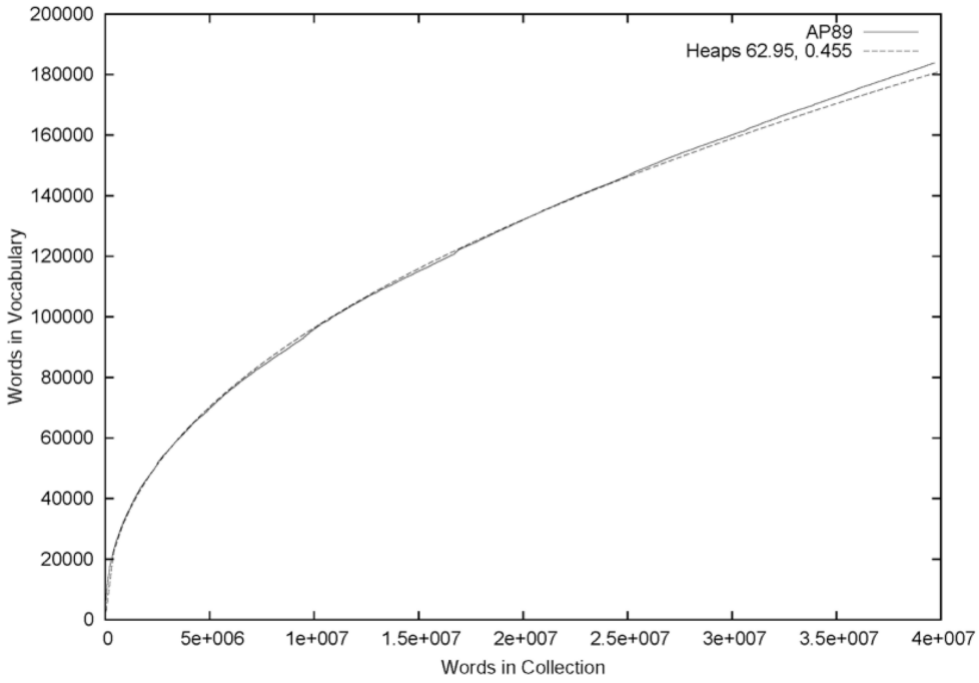
$$V = Kn^{\beta}, \qquad K \approx 10-100,\ 0<\beta<1$$

- V: vocabulary size, number of new/unique terms
- n: text size/length (number of terms,counting repetitions)
- V = f(n)
- Typical β values: 0.4 – 0.6

Then, the number of new terms in an n words text increases proportionally to $\sqrt{n}$.

Heaps is very accurate, also for very large collections, even if is an empirical law obtained by experiments.

Example: Heap's law on AP89 collection:



4.1.3 Words upper limit

The numer of terms in a language is pretty constant, then there must be an upper limit. It is almost true but neologisms, misspellings, numbers, dates, email addresses, urls, etc. It is not very clear, but experimentally no upper limit is found.

4.1.4 Average word length

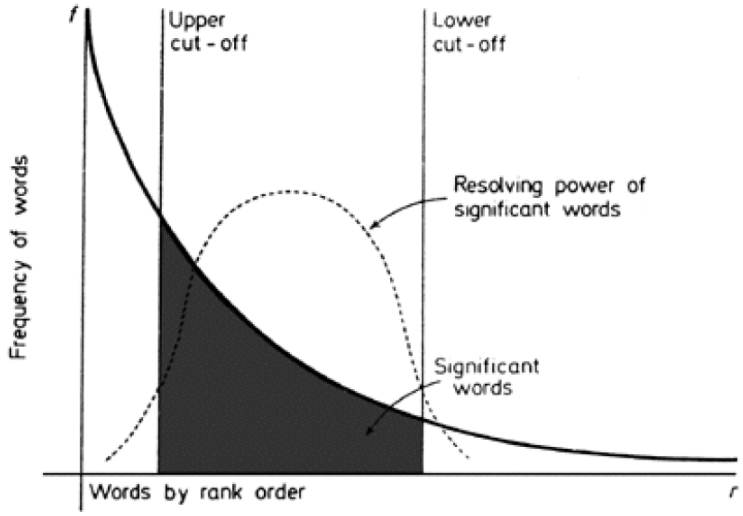
Given the number of terms n, the size of the vocabulary in words V can be computed with Heaps. But what about the number of chars? The average length of words is about 5 (4.8-5.3). Without stopwords it's about 6-7.

4.2 Consequences on index construction

Generally speaking, not all the terms need to be in the index because some terms are more meaningful. Some terms just make noise and some terms are more/less important than others. We should also consider that resources might be scarce. The index might contain also terms that are not in the documents: anchor text, tags, metadata.

Not only it is important how much a term "describes well" a document but it is also important how much a term discriminates a document from the other ones. Too frequent terms (extreme case: in all docs.) are not much useful for retrieval, they do not discriminate.

Also terms with an extremely low frequency (extreme: 0-1 docs) are not much useful: they might be errors and it is unlikely that they are used in the query.



4.3 Before building the index

Before proper index building, some steps are required (some are optional):

1. Lexical analysis (tokenization)
2. Stopwords removal
3. Stemming
4. Index terms selections
5. Thesaura construction

4.3.1 Lexical analysis

The tokenization (lexical analysis) process transforms a character stream (text in documents) into a word stream (the candidate index terms).

Usually numbers are not good index terms because they are vague without context. There are also several small problems: collection with N documents, each one with a numeric ID: add N terms to the index? Search of ranges/intervals, sometimes mixed with letters in a world. Digits and numbers can also be important for some collections: physicists working on finding values of constants.

Often numbers are not selected as index terms but there are exceptions, maybe specified by regular expressions or maybe normalizing dates, indexing parts of numbers (instead of whole numbers), etc.

"-" is often replaced by space by paying attention to particular cases, within a word. Dots and commas are used for the end of sentence, but not only! Sometimes they can be ignored but it must be payed attention to numbers (decimals) or software collections.

Capitalization is usually converted in lowercased. Even here there are minor problems for particular cases to be take care of.

4.3.2 Stopwords removal

Stopwords: articles, prepositions, conjunctions that occur in every document but carry just "little meaning". Words occurring in 80-90% of documents are useless for retrieval ("resolving power").

Usually are not index terms. NB: a stopword is not a term occuring in every document but a term that occur in every document of every collection (in a given language). The advantages of doing stopwords removal is that it is an effective technique for index compression. For example the word computer which appears in every document of a collection of tech articles is not a stopword!

4.3.3 Stemming

Stemming means to reduce a word to its stem, removing suffixes and prefixes. The stem is the part of the word that remains after suffix/prefix removal. Stemming is useful because the query might contain one term and the document the stemmed one. By conflating to a common stem we save space/storage in the index. There are also disadvantages of practicing stemming: as usual, the end user might not understand. Increases recall, ok for small collections, not so ok for large collections. As usual, web SEs tend not to stem (or rather to do a light stemming).

There are 3 kinds of algorithms to execute stemming:

- Tabular search: a table word to stem; of course expensive for the storage and difficult to populate;
- Morphological, lingustic: to exploit knowledge from computational linguistics, it is complex;
- Suffix (and prefix) removal: rewriting rules to remove suffixes, stepwise. It is intuitive, simple, efficient and more or less effective compared to the previous ones (for retrieval).

Porter algorithm

The idea: set of rewriting rules to remove suffixes. For example:

- $s \rightarrow \varepsilon$, $sses \rightarrow ss$, ...: from plural to singular
- $ed \rightarrow \varepsilon$, $ing \rightarrow \varepsilon$, ...: present/past participle, past, ...
- $ization \rightarrow ize$, ...

On effectivness of stemming: controversial, different studies obtained different results (it depends on the collection, information need, user preferences). The efficiency gain is however certain: index size reduced.

4.3.4 Index terms selections

There are two options: full-text and selection which can be done manually, or automatically.

4.3.5 Thesaura construction

It is a collection (list) of multi-terms plus relationships between terms (synonyms but not only). Example:

- **Main Entry**: expert
- **Part of Speech**: noun
- **Definition**: specialist
- **Synonyms**: ace, adept, artist, artiste, authority, buff, connoisseur, doyen, gnome, graduate, guru, hot shot, master, maven, old hand, old pro, phenom, pro, professional, proficient, shark, virtuoso, whiz, wizard
- **Antonyms**: amateur, apprentice, beginner, novice, tyro

The thesaurus can be build automatically. We can build a graph in which nodes are terms and arcs are relationships. There are different kinds of arcs: synonymy, narrower/broader terms, related terms etc. Starting from the collection, typically during the indexing phase.

The thesaura, in IR, is used for query formulation: the end user looks for adequate terms in the thesaurus or for query reformulations: the end user and/or the system search in the thesaurus for terms that are related to query terms.

5 Inverted Index

We have already seen how the index is build - conceptually: a matrix of weights (e.g. tf.idf), which is very large (not efficient).

$N \sim 10^6 - 10^{10}, t \sim 10^5, N >> t$

$$\mathbf{M} = \begin{array}{c|cccccc} & d_1 & d_2 & \dots & d_j & \dots & d_N \\ k_1 & w_{11} & w_{12} & \dots & \dots & \dots & \dots \\ k_2 & w_{21} & w_{22} & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ k_i & \dots & \dots & \dots & w_{ij} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ k_t & \dots & \dots & \dots & \dots & \dots & w_{tN} \end{array}$$

The matrix is conceptually ok, but it is bad for implementation. If $N = 10^{10}, t = 10^5$, each w_{ij} 4 byte (float) we have 10^{15} byte = 1000 Terabyte. Anyway, it's almost all zeros, it is a very sparse matrix (actually about 99.8% are zeros). There's waste of space.

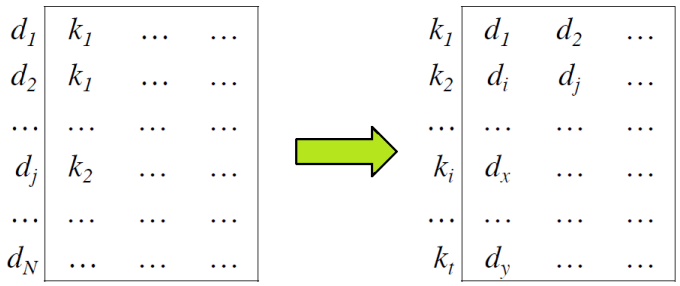
Notation:

- N = number of docs;
- t = number of words;
- n = size in bytes of the collection;
- m = length in bytes of the query;
- M = size of the main memory;
- n' = size in byte of the collection update (adding/removing/etc. docs).

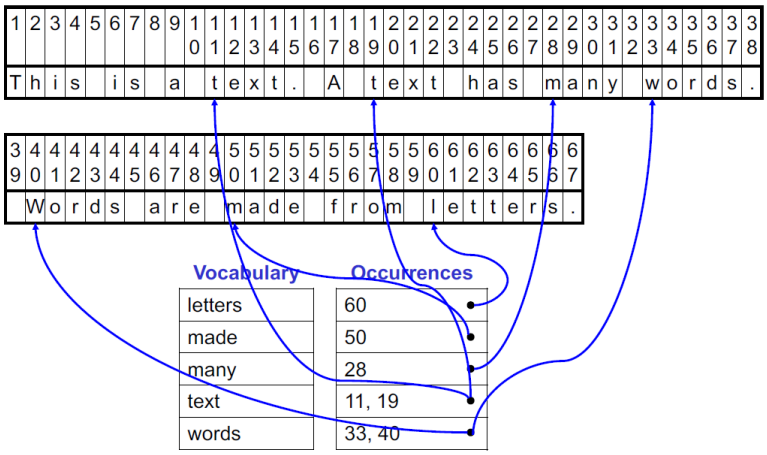
Let's now talk about the **inverted index** which is composed of two elements:

- **Vocabulary:** set of all the words to be indexed ("index terms");
- **Occurences:** position in the text where the index terms occur:
  - As number of chars: ok for direct access;
  - As number of words: ok for contextual search. Smaller.

The index is called inverted because we start from a representation of what are the terms in each document to a representation of what are the documents that contain each term.

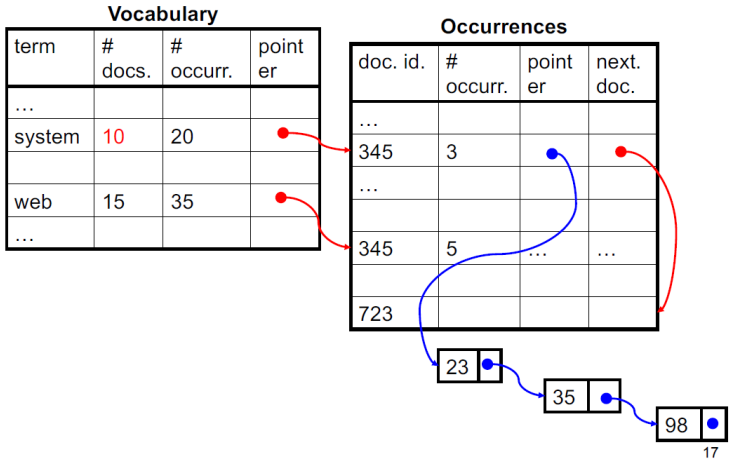


A first basic representation of an index:



In the vocabulary we have: terms, number of docs. where it occurs (df), number of total occurences, point to occurences list.

The occurences are represented as a lists in which we can find the document ID, the number of occurences in the doc. (tf), a pointer to a list with occurences position and a pointer to next document containing the term.



There are several optimizations for the inverted index:

- Keep vocubulary and occurences in two different files: Vocabulary is small and usually it fits in main memory. Occurences optimizations do not change the vocabulary.

- In the occurences, a list of arrays in place of a simple list: consecutive memory addresses, when it is full we "jump" to another set of consecutive memory addresses. Periodical reorganizations.
- Absolute frequencies are useful and convenient. To compute the "weight" and to update the index when the collection changes (no need to rebuild it from scratch).

Let's analyze the inverted index space requirements.

- Space of the vocabulary: small (remember heaps: $O(\sqrt{n})$)
- Space of the occurences: much larger. For every word in the text there is an occurence ($O(n)$). Without stopwords, occurences space is 60-70% less.
- Empirically the inverted index uses 25-50% of the entire collection.

5.0.1 Block addressing

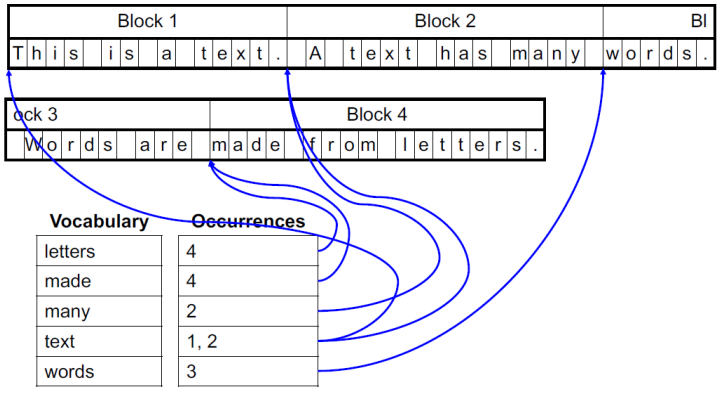
Block addressing is a compression technique used to save memory space. Instead of storing the exact position of a term we divide the text into blocks and we store the block index/link in which the term occurs.

Pro:

- Smaller pointers (there are fewer blocks than chars/words);
- Fewer occurences (more occurences in the same block are merged);
- Space efficiency: from 30-40% to around 5% (with a typical block size of 256byte - 64Kbyte).

Cons:

- If the exact position is needed (e.g. contextual query) then sequential access to the text is needed; because it must be computed at run-time.
- The collection has to be available.



5.1 Using the index to answer a query

There are three main steps to follow in order to use the index to answer a query:

1. Search in the vocabulary words and patterns;
2. Retrieve the occurences lists;
3. Scan/process the lists (union, intersection, etc.) for boolean/contextual/structural/etc. operators.

5.1.1 Search in the vocabulary

Search starts from the vocabulary and it is the reason why it is convenient to have it in main memory, in a separate file. There are data structure that can speed up the search:

- Vocabulary lexicographically ordered and binary search: $O(log\ n)$
- B-Tree, Red Black Tree, Splay Tree, ... : $O(log\ n)$
- Hash, Trie, ... : $O(m)$
- Prefix and interval queries: ok but with hash

5.1.2 Scan/process the occurences lists

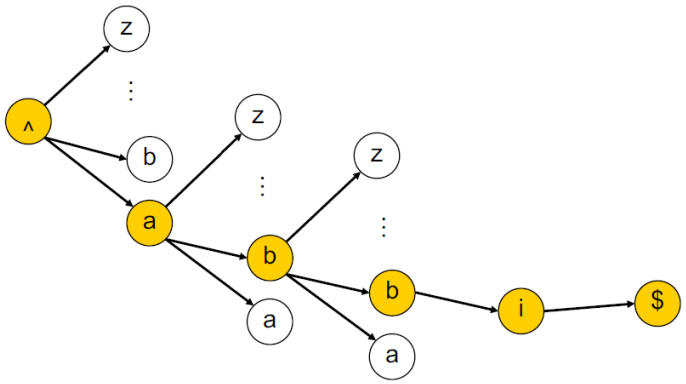
This phase has a higher temporal complexity. We should evaluate boolean queries with some regards to which operand is going to discard more terms and use it first. Contextual search is more complex. The scan must be intelligent and not performed naively.

5.2 Index construction

Let's start with a prerequisite notion: the trie data structure.

5.2.1 Trie

In the trie data structure the search operation has time complexity $O(m)$.



We can build an index with a trie and the occurence lists. For each term k in the collection we search it in the trie: if not found, we insert it with empty occurence list. We add the occurence (position) at the end of the list. At the end, we save the trie and lists on a file (vocabulary and occurences in two different files).

The time complexity for index construction is $O(n)$ in the worst case. For each character in the text, we have $O(1)$ operations in the trie. The insert at the end of the occurence list has cost $O(1)$ (by keeping a pointer, of course).

But if the collection is large, the data structure (trie + occurences) does not fit in main memory. Even with virtual memory and caching/paging the time effectiveness decreases. An alternative are the partial indexes plus merging.

In this approach we proceed as above until main memory is full. We save the partial index I_i in a file then we free the main memory (only from occurences, the vocabulary stays in memory).

Then we repeat with the remaining text and repeat. At the end, partial indexes are on a disk and are merged together hierarchically.

The partial index construction has cost $O(n)$ as above. The number of partial indexes is $O(n/M)$ with M = main memory size. $O(log_2(n/M))$ merge levels are needed. The total cost in time is: $O(n * log_2(n/M))$.

We perform indexing updating when the collection is modified (usually, when new documents arrive). The so called "Incremental indexing" is an increasing more important/fundamental requirement.

It is implemented in a natural and efficient way with the previous algorithm. When new documents are added we create a new partial index and we perform merging.

5.3 Index compression

Three things can be compressed about the index:

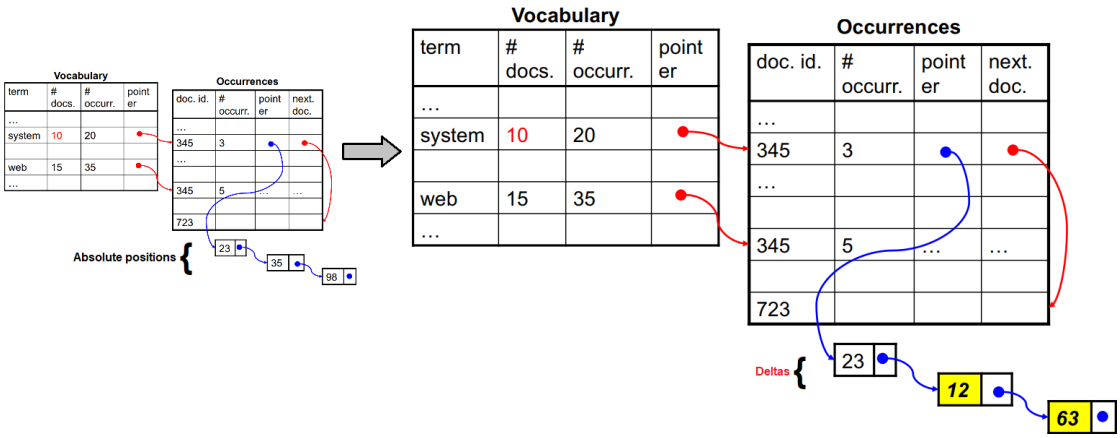
- Collection
- Index's vocabulary
- Index's occurences list

However it is very difficult to search on a compressed text. Up to some years ago only what searched rarely was compressed (and decompressed before searching). This saved space but costed time. Up to some years ago compression was not effective because decompression time was very high. Today with more efficient hardware and algorithms compress and decompress operations are more efficient. Temporal efficiency can even be improved because fewer data are transferred from secondary storage to main memory, so the total operation time complexity is minor.

Type of compressions:

- **Collection compression**: decompress only to show the document.
- **Vocabulary compression**: if using the same code used in the collection, the compressed query term is searched.

- **Occurence lists compression**: positions are in increasing order: represent deltas in place of absolute values. Lists are scanned sequentially, so almost nothing is lost in time efficiency.



Collection and vocabulary compression can be made with the **Huffman algorithm**. The algorithm is based on this principle: to assign:

- a longer code to less frequent symbols (having lower occurence probability), and
- a shorter code to symbols that are more frequent (higher probability).

(Whereas ASCII/Unicode/... assign codes having the same length to all symbols/letters)

Actually we don't use a code for each letter but a code for each word. Words are searched, not letters. Words are used in the index so we already have them (with their frequencies).

A field is added in the vocabulary (the code). It is important to save the original term and the term-code association. Anyway, the collection is compressed and this will save much more space. Shorter codes to more frequent terms.

5.3.1 Huffman algorithm

A trie is constructed. Starting from leaves, listed in increasing frequency order, the two least frequent nodes/leaves are selected and a new parent node is created, with frequency equal to the sum of the two frequencies. The two edges are labeled as 0 and 1. Repeat. The obtained trie gives the code for each symbol (on the paths).

6 Query reformulation

With "reformulation" we identify techniques and models to modify the query (term add/remove/replace/weight change). There are three kinds of reformulation. Manual reformulation, which is done by the user by manually changing the query; Automatic reformulations (Automatic Query Expansion - e.g. with pseudo relevance feedback) and Semi-automatic (Interactive QE) in which the user changes the query manually, but the system provides suggestions.

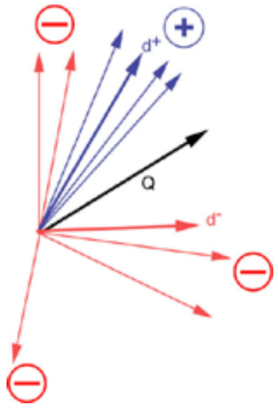
The are three main reformulation techniques:

- **Relevance feedback**
- **Local analysis:** use the retrieved documents to find new query terms
  - Clusters: Association, Metric, Scalar
  - Local Context analysis
- **Global analysis:** use the whole collection to perform reformulation
  - Similarity Thesaurus
  - Statistic Thesaurus

6.1 Relevance feedback

After the user submits a query, it examines the (first?) retrieved documents providing feedback on their relevance (positive or negative, for example). The system can exploit that information (of which documents has been opened or manually marked as relevant) to improve the query by adding or removing terms, by changing term weights, etc.

Visual representation of the relevance feedback in the vector space model:



6.1.1 The Cluster Hypothesis

The Cluster Hypothesis claims that: "closely associated documents tend to be relevant to the same requests". The relevant documents for a given information-need tend to be in the same part of the vector space and they have similar vectors. If it is not true then relevance feedback doesn't work. This Hypothesis seems reasonable and empirically/experimentally true. However, it could be false in different ways: more than one cluster, one cluster but very sparse, etc. (multi-cluster hypothesis).

6.1.2 Deriving the new query

How to derive the new query (from q to q')? We try to move q towards the cluster of relevant documents and far away from irrelevant ones (by working with/on centroids).

There are three classical formulas:

- Rocchio: $q' = \alpha\, q + \frac{\beta}{|D_r|} \sum_{\forall d_j \in D_r} d_j - \frac{\gamma}{|D_n|} \sum_{\forall d_j \in D_n} d_j$
- Ide: $q' = \alpha\, q + \beta \sum_{\forall d_j \in D_r} d_j - \gamma \sum_{\forall d_j \in D_n} d_j$
- Ide_Dec_Hi: $q' = \alpha\, q + \frac{\beta}{|D_r|} \sum_{\forall d_j \in D_r} d_j - \gamma \; first\,(d_j)_{\forall d_j \in D_n}$
  - D_r (and D_n): retrieved and judged (non) relevant
  - α, β, γ: constant
  - *first*: the first of the not relevant and retrieved

Those three formulas have similar empirical effectiveness. But how is effectiveness evaluated? Evaluation should be performed on the residual collection, without the already judged docs. Effectiveness is going to decrease but it allows to compare different techniques. Examples: what would you prefer?

- An IR system that moves to earlier rank positions docs. that you have already judged as relevant.
- An IR system that does not show you anymore docs. that you have judged as relevant.
- And IR system that saves into a folder docs. that you have already judged as relevant then finds other docs, maybe not showing those that you have already seen.

6.1.3 Relevance feedback and probabilistic model

As already seen relevance feedback is at the basis of the prob. model. It does not include query expansion but only weight changes. It is less effective than vector space but some extensions have been proposed: for example to add terms separated from weight change.

Some remarks on relevance feedback: it is simple for the user...

- That has only to judge document relevance;
- No need to manually modify the query terms or weights;
- No need to see reformulated query;
- Indeed, (empirically) better if the reformulated query is not shown to the user, that might find it strange.

Relevance feedback works on the documents (and only implicity on the terms).

6.2 Local Analysis

Let's start with some prerequisite notions. We have the usual matrix M:

	d_1	d_2	…	d_j	…	d_N
k_1	w_{11}	w_{12}	…	…	…	…
k_2	w_{21}	w_{22}	…	…	…	…
…	…	…	…	…	…	…
k_i	…	…	…	w_{ij}	…	…
…	…	…	…	…	…	…
k_t	…	…	…	…	…	w_{tN}

What is $M * M^T$?

$d_1 = (k_1, k_2)$ $\quad$ $d_3 = (k_1, k_2)$

$d_2 = (k_1, k_2, k_3)$ $\quad$ $d_4 = (k_2, k_3)$

$M \cdot M^T = ?$

	d_1	d_2	d_3	d_4
k_1	1	1	1	0
k_2	1	1	1	1
k_3	0	1	0	1

$\cdot$

	k_1	k_2	k_3
d_1	1	1	0
d_2	1	1	1
d_3	1	1	0
d_4	0	1	1

$=$

	k_1	k_2	k_3
k_1	3	3	1
k_2	3	4	2
k_3	1	2	2

This multiplication:

- Add 0 when (at least) one of the two is zero;
- Add 1 when both are one: when the terms co-occur in the doc.

So this multiplication is counting the number of **co-occurences**:

$$\left(M \cdot M^T\right)_{u,v} = \sum_{d_j \in D} freq(k_u, d_j) \cdot freq(k_v, d_j)$$

$M * M^T$ tells us how many times the terms co-occur in a doc. (term/term co-occurence matrix):

- diagonal: df, how many docs. contain the terms
- (sum of "1"s over one row of M)

If we have the usual weights in place of 0s and 1s, the result is similar:

- 0 if they don't co-occur,
- >0 depending on "how much" they co-occur and how much important they are.

Therefore $M * M^T$ it's a **term/term co-occurence matrix**.

Hypothesis: terms that co-occur are "associated" semantically. For example if Java and C++ co-occur, that should mean something. If Microsoft and Office co-occur it should mean something too. If Hamburgers and Lions doesn't co-occur...

We now have an automatic method to measure the association between terms. Let's go back to automatic query expansion with local clustering (association clusters).

6.2.1 Local and global analysis

Measuring the association between terms, for example counting the number of co-occurences. We use the terms more "associated" to (the terms of) the query q to do the Query Expansion (QE). If performed on the basis of the whole collection, it does not seem effective:

- Most likely because the "context" determinated by the q is different from the whole collection. For example: polysemic term in the collection could be clearly disambiguated in the query: Java Eiffel/C++ (collection = Web, for ex.).

Let's go back to local analysis. In local analysis the retrieved docs. are examinated at query time and are used to choose the terms for query expansion. This is done with no user intervention, completely automatic ("pseudo relevance feedback"). There are two main techniques:

- Local clustering
- Local context analysis

6.2.2 QE with local clustering

The basic idea is to construct a cluster of terms related to the single query terms (a cluster for each term) and then add to the query q the terms in the clusters.

There are 3 approaches to cluster construction:

- Association clusters
- Metric clusters
- Scalar clusters

Association clusters

We don't use the whole M, let's restrict to the "local context". Given a query q we define:

- D_l: the set of retrieved docs
- V_l: the set of distinct terms (vocabulary) in D_l

and then we build a matrix M_l with V_l rows and D_l columns which is similar to M but restricted to the local context: $M_{ij} = freq(k_i, d_j)$. Only the documents in D_l and the terms in V_l are taken into account:

- The columns j such that $d_j \in D_l$
- The rows that have at least one value greater than 0

We now compute the co-occurence matrix $C = M_l * M_l^T$

$$C_{u,v} = \sum_{d_j \in D_l} freq(k_u, d_j) \cdot freq(k_v, d_j)$$

- $C_{u,v}$ is the absolute frequency of co-occurrences of k_u and k_v in the docs in D_l
- Let us normalize C: $CN_{u,v} = \dfrac{C_{u,v}}{C_{u,u} + C_{v,v} - C_{u,v}}$

  The denominator is:
  # occurrences (~doc.) of k_u
  + # occurrences (~doc.) of k_v
  − # occurrences (~doc.) of both =
  "number of occurrences (~doc.) of either one"

Now that we have the association matrix CN, it tells us how much the terms from q co-occur in retrieved documents.

How do we build the clusters? for each term $k_u \in q$:

- In CN we select the row corresponding to the term k_u (u-th row) and consider the n (threshold parameter) highest values (excluding the u-th column). The terms that "co-occur most with k_u".

- This is a cluster of terms to be added to q.

CN matrix has to be computed at query time but:

- Only the rows in CN corresponding to the terms in q are needed;
- The best clusters are the small ones;
- Thus it is not too slow.

Effectiveness improves experimentally, but it can be improved.

Metric clusters

The idea is that association clusters are based on co-occurences in the document but two terms that co-occur "near" are more related than two term that co-occur in a doc. "far away". Metric clusters consider the distance of the co-occurence: if two terms co-occur "near to each other", then they "co-occure more". We define:

- $r(k_u, k_v)$ as the distance (number of words between the two occurences + 1). If k_u and k_v do not co-occur then the distance is infinite.

Now C is defined as follow:

$$C_{u,v} = \sum_{k_u \in V_l} \sum_{k_v \in V_l} \frac{1}{r(k_u, k_v)}$$

$C_{u,v}$ is how much k_v and k_u co-occur and it increase if r decrease. Now, the matrix is used exactly as before: the obtained clusters are metric.

Scalar clusters

The idea is that terms with similar "neighbors" are similar if k_u and k_v co-occur with the same terms, k_u and k_v are similar even if they don't co-occur (e.g. they might be synonyms).

A matrix C is created (association or metric, normalized or not). This matrix is used to create an association matrix C':

- Consider the rows of k_u and k_v
- Choose the n largest values in each row
- Build 2 vectors with those values
- Compare the two vectors (with scalar product), and this is the association matrix that is used as before:

$$C'_{u,v} = \frac{k_u \cdot k_v}{|k_u| \cdot |k_v|}$$

Effectivness: The terms in the cluster of a $k_u \in q$ that are added to the query, sometimes are synonyms, often they are not. The resulting query retrieves docs/concepts different from the original ones. Normalized and not normalized matrices/clusters give different results, that can be fruitfully merged. Metric clusters seem more effective than the association ones because correlation between two terms seems to depend on their distance. In general:

- Local clustering seems effective.
- Even if local techniques are effective, the same ones but at the global level (on the whole collection) are not.

It's worth mentioning that actually, stems are usually used and not terms.

6.2.3 Local context analysis

It still is local analysis: the retrieved documents are analyzed to do query expansion. It works on "concepts", not on terms. The are 4 steps:

1. The retrieved docs. are decomposed into passages of fixed length (e.g. 300 words);
2. The passages are sorted as if they were docs, and the first n are selected;
3. For each concept c in the first n passages, $sim(q, c)$ is computed (tf-idf with some changes). Concept: not a term but a noun phrase.
4. The first m concepts are selected (the most similar to the query q) and added to q, with decreasing weights $(1 - 0.9 * i/m))$ and smaller than original terms.

It has good effectiveness but tailored to one collection. To adapt it to different collections changes have to be made to the formulas, constants, etc. Distance between terms is not taken into account explicitly (compared to associative and metric clusters), but anyway passages are rather short (300 words).

6.3 Global analysis

Global analysis works not only with information from retrieved documents, but also from the whole collection ("global"). At indexing time a structure to relate terms is built: it is similar to a thesaurus: similarity between terms, co-occurence in the same docs, etc.

At query time, the structure is used for Query Expansion by adding new terms and by doing re-weighting. The structure is not so related to the query:

- It is important to exploit the similarity with the whole query.
- Similarity with single terms (as in the local analysis) is not enough, there are alternatives, let's see 3 of them.

6.3.1 QE with a global co-occurence thesaurus

As for association/metric/scalar clusters but built on the whole collection, not only on retrieved docs. Does not seems effective, other approaches but more complex are possible.

6.3.2 QE with similarity thesaurus

The thesaurus is based on similarity, not on co-occurence. A vector space is built:

- A space for terms, not for documents.
- Each dot/vector in this space is a term, not a document.
- The dimensions are "concepts" (each doc. is a concept).

It is a "dual" space: dimensions are docs, not terms. docs. and terms roles are swapped.

M^*:

	d_1	d_2	...	d_j	...	d_N
k_1	w^*_{11}	w^*_{12}	...	...	...	...
k_2	w^*_{21}	w^*_{22}	...	...	...	...
...	...	...	...	...	...	...
k_i	...	...	...	w^*_{ij}	...	...
...	...	...	...	...	...	...
k_t	...	...	...	...	...	w^*_{tN}

Values in M^* are different from the usual tf.idf, they are **tf.itf**.

Weight more shortest docs:

- If 2 terms co-occur in a short document, they are more similar than 2 terms that co-occur in a longer document.
- If a term occurs in a short document, it is more important to express that concept (to describe that doc.).

Weights are not occurence measure anymore:

- They are measures of how much a term is important to express a concept.

Construction and use of the global similarity thesaurus

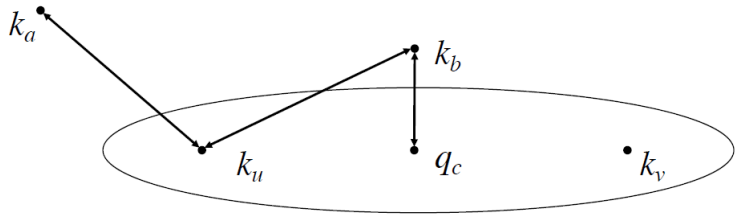
As above, but start from the M^* matrix. Then $C = M^* * M^{*T}$. The query is represented in the concept space. The terms that occur in a query are weighted with tf.itf. The most similar terms (to the whole query) are selected.

Similarity with q and with single terms in q

$q = \{k_u, k_v\}$

The single term k_u has the same distance from (has the same similarity with) k_a and k_b but k_b is much more close/similar to the centroid q_c.

On efficiency: The global thesaurus is much bigger than the local one, then it will require more resources but it is built once for all at indexing time, not at query time and can be updated when new documents are added.



6.3.3 QE with statistical thesaurus

It is based on using clusters of documents to cluster terms. Documents are clustered (with threshold based algorithms). If the documents in a cluster are very similar (threshold based choice) and they are not too many (other threshold) then select (in the cluster) the terms with the lowest frequency in the collection (higher idf) and build a cluster of terms with them.

In this way, terms with low frequency can be clustered: it is difficult because there's not much info, but they are the most useful/discriminating ones.

We use these term to add them to the query similarly as before. Threshold selection is difficult because it is collection dependent.

On effectivness of global analysis: it seemed not effective until 90s. More modern techniques (like the last two) are effective also when combined with local ones.

6.4 Semi-Automatic reformulation

It is not the system that adds terms to the query automatically but it proposes terms to the user to be added to the query, and the user selects some of them. Pros and contra: the user can't add/manage 20 terms, some of wich are apparently not related but only the user (maybe) know the information need.

7 Categorization and Clustering

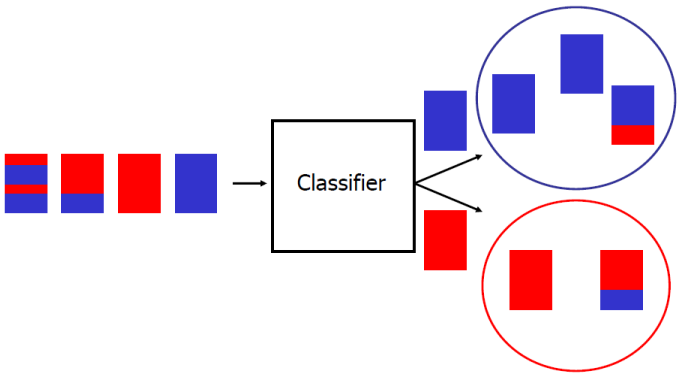
Given a set of classes (or categories), the aim is to determine (automatically) which class(es) a given object belongs to. Examples:

- Spam categorization: 2 categories
- Sentiment analysis: opinions/reviews etc.
- Information filtering: 2 categories: interesting and not interesting
- Email message management: many categories/folders
- Rank of retrieved docs: relevant, not relevant etc.

The primary difference between classification and clustering is that classification is a supervised learning approach where a specific label is provided to the machine to classify new observations. On the other hand, clustering is an unsupervised learning approach where grouping is done on similarities basis.

7.1 Classifier

A classifier is trained on a set of data called training set. A human provides the correct classification (supervised learning). The system learns the features that the documents should have to belong to one class or another and then it uses the features to infer the class of the new document(s). There are multiple approaches: from Machine learning, AI. The simplest one is the Naive Bayes classifier.



7.1.1 Naive Bayes Classifier

The idea behind this classic classifier is similar to the IR probabilistic model (BIM). The probability (similarity measure) of the document d in category c:

Bayes

$$p(c \mid d) = \frac{p(c) \cdot p(d \mid c)}{p(d)} \approx p(c) \cdot p(d \mid c) =$$

$p(d)$ = for every category

Representation of d

$$= p(c) \cdot p(\{k_1, k_2, \ldots, k_n\} \mid c) =$$

Independence

$$= p(c) \cdot \prod_{i=1}^{n} p(k_i \mid c)$$

$$p(c \mid d) = p(c) \cdot \prod_{i=1}^{n} p(k_i \mid c)$$

Estimate how frequent the class c is:

$$p(c) = \frac{\#\, docs\ in\ c}{\#\, of\ total\ docs}$$

Estimate how much k_i is a good indicator for c:

$$p(k_i \mid c) = \frac{\#\, occ\ of\ k_i\ in\ c}{\#\, total\ occ\ in\ c}$$

d is classified into the c having the greater $p(c|d)$.

This classifier has two problems:

1. New document with mixed terms: some terms from one category, some from another with a clear majority for one category.
2. New document with a new term: the new term does not occur in any previous document, and then in any cateogry but all the other terms are in the category.

We would even know where to classify the new document but all computation give zero as results. The solution to this problem is called smoothing: we smooth the probabilities, adding a small amount to remove zeros.

$$p(k_i \mid c) = \frac{\#\, occ\ in\ c + 1}{\#\, total\ occ + \#\, vocabulary\ terms}$$

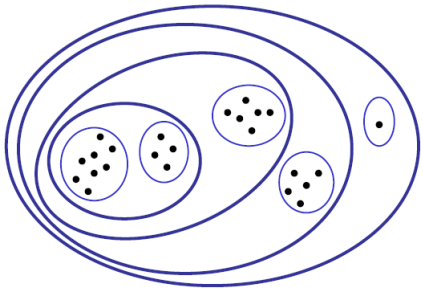
Intuition: each term occurs once in any category and then the training documents arrive.

Naive Bayes classifier is the simplest classifier (BIM is a particular case in which the are two categories: relevant and nonrelevant).

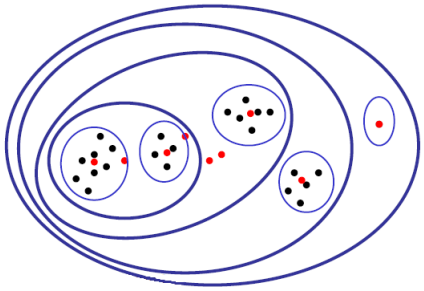
There are more complex and effective classifiers: SVM (Support Vector Machines), etc. and tecniques borrowed from ML, for example clustering.

7.2 Clustering

A cluster is a group, subset, bunch. Clustering is the process to organize the elements into groups of similar elements (in some way). The aim is that given a set and a similarity relationship the algorithm group similar elements together. Example:



For each cluster a centroid can be defined. For example the average of the elements in the cluster or the center of gravity.



7.2.1 Categorization $\neq$ Clustering

It is important to note that clustering and categorization are not the same.

Categorization:

- Predefined categories;
- An element has to be assigned to one or more categories.

Clustering:

- No predefined categories;
- Elements have to be grouped on the basis of their similarity;
- Only at the end groups will correspond to categories (according to some principle).

7.2.2 What and Why to cluster in IR?

In general, everything can be clustered but what is needed is a set and a similiratity relationship. In IR we usually cluster documents, collection, retrieved docs., terms, queries.

To cluster a collection can be useful in IR because it increases efficiency: query is matched with the centroid rather than with individual documents. Even without query, the end-user browses the clusters.

To cluster retrieved documents help to show the results in a more effective way. To cluster terms helps for formulation/reformulation.

7.3 Clustering algorithms

There are two main approaches to clustering:

- **Top-down:** the collection is a single large cluster and it is partitioned, separating dissimilar elements.
- **Bottom-up:** each document is a small cluster, and the most similar clusters are clustered, then repeat until when the whole collection is clustered.

There exist several techniques/algorithms for clustering: Hierchical clustering algorithms: single link, complete link, group average, etc. and heuristic clustering algorithms: more efficient, lower computational complexity.

7.3.1 Hierarchical clustering

The idea behind hierarchical clustering (common to all HAC algorithms) is to start from the single elements (bottom-up approach), put each element in its own cluster, choose the two most similar clusters and merge them. Repeat this until a single cluster is obtained. Algorithms differ from how to define/compute cluster similarity:

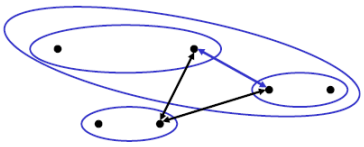
- **Single link:** Similarity between two cluster is the **maximum** similarity between elements of the two clusters;
- **Complete link:** Similarity between two cluster is the **minium** similarity between ...;
- **Group average:** Similarity ... is the **average** ...

More in detail:

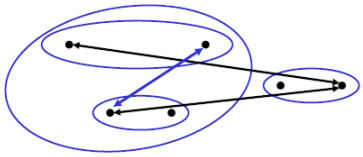
- **Single link:** distance from the nearest neighbor: privileges large and sparse clusters. Each element in a cluster will be more similar to at least one element in the same cluster than to the elements in the other clusters;
- **Complete link:** distance from the farthermost neighbor: privileges small and compact, dense clusters. Each element in a cluster will be more similar to the least similar element in the same cluster than to the least similar element in the other clusters;
- **Group average:** in between: distance from the average neighbor.

For IR, single link is usually the worst choice. Large and sparse clusters are not good for IR.

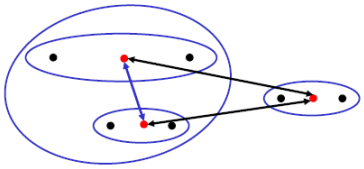
- **Single link**
  - Max sim: Min "distance"



- **Complete link**
  - Min sim: Max "distance"



- **Group average**
  - Avg. sim: Average "distance"
    - E.g., with centroids, but not necessarily, other averages/distances could be used (sim^2, ...)



7.3.2 Computational complexity

Computation of the Similiarity Matrix: the hierchical clustering algorithms need the similarity values for each pair of documents: The usual matrix M:

	d_1	d_2	...	d_j	...	d_N
k_1	w_{11}	w_{12}	...	...	...	...
k_2	w_{21}	w_{22}	...	...	...	...
...	...	...	...	...	...	...
k_i	...	...	...	w_{ij}	...	...
...	...	...	...	...	...	...
k_t	...	...	...	...	...	w_{tN}

We already know $M * M^T$ (term-term co-occurrence matrix). But what is $M^T * M$?

$d_1 = (k_1, k_2)$ $\quad$ $d_3 = (k_1, k_3)$

$d_2 = (k_1, k_2, k_3)$ $\quad$ $d_4 = (k_2, k_3)$

$M^T \cdot M = ?$

	k_1	k_2	k_3
d_1	1	1	0
d_2	1	1	1
d_3	1	0	1
d_4	0	1	1

$\cdot$

	d_1	d_2	d_3	d_4
k_1	1	1	1	0
k_2	1	1	0	1
k_3	0	1	1	1

$=$

	d_1	d_2	d_3	d_4
d_1	2	2	1	1
d_2	2	3	2	2
d_3	1	2	2	1
d_4	1	2	1	2

We:

- add 0 when one of the two is 0 (or both are 0)
- add 1 when both are 1 (i.e. when both docs. contain the same term)
- So, the number of common terms is counted: $M^T * M$ shows how many common terms occur in the two documents: on diagonal = number of different terms (about the length of the docs.)
- If 0s and 1s are replaced with weights, similar results: 0 if two docs. have no terms in common, >0 depending on how many "important" terms are in common.

Therefore $M^T * M$ is a **document/document similarity matrix**.

Hypotesis: documents that contain the same terms are semantically similar. This is not the only way: This computes the similarity as the cosine of the angle of the two term vectors. Other similarity functions can be defined.

Computational effort for clustering n docs: To build the similarity matrix $M^T * M : O(n^2)$.

- If clustering of a collection of 10^{10} documents $\rightarrow$ ~10^{20}
- Even if each matrix element is computer in 10^{-9} sec (1 nanosec): not realistic.
- It sums up to 10^{11} sec = ~10^6 days (more than 1000 years).

This is just to build the matrix, clustering hasn't started yet... this solution is not feasible.

The three algorithms have computational complexity of $O(n^2)$ with n = number of documents in the collection. Unfeasible on the whole collection, inexact/heuristic algorithms are used instead.

7.3.3 Heuristic algorithms

Examples:

- One-Pass clustering
- Rocchio
- K-means
- ...

They are $O(n)$. If each matrix element is computed in 10^{-9} seconds (1ns - still unrealistic) it sums up to 10sec. Sure, still unrealistic but clearly better than before.

One-pass clustering
Pick up the 1st document and put it in cluster n°1. Then pick up the 2nd doc. and, if it is similar enough (threshold) to cluster 1, add to it, otherwise put in a new cluster 2.

Pick up the n-th doc, compute its similarity with centroids of existing clusters. If above threshold, add to the cluster (and update centroid) also in more than one cluster. Otherwise create a new cluster.

This method depends on the order of the docs. With the thresholds there is the risk that clusters are too large or too small (even a single element). To avoid it, controls on:

- cluster size
- overlap
- n° clusters

If needed, splits or merges are performed. The effectivness of this method for IR is experimentally often not so worse than hierarchical exact clustering.

Rocchio clustering algorithm
It is based on "density" (number of neighbors) in the document space. It takes document at a time. If the n-th doc. d passes a density test (it has many neighbors) and it is not in any cluster yet: it is used as "cluster seed" to create a new cluster. The neighbor documents of d are added to the cluster.

K-means
k centroid are chosen: for example randomly, but if the choice is unfortunate, the results are worst. They should be far away. Documents are considered sequentially, each one is put in the cluster having the most similar centroid. Then, after all documents have been put in a cluster, the k centroids are recomputed. This is repated until centroids "don't change too much" (threshold). It is very popular and used a lot.

7.3.4 The Cluster Hypothesis

As already seen before, the cluster hypothesis says: "closely associated documents tend to be relevant to the same requests"

i.e. documents relevant to a given information need tend to be in the same region of the vector space: have similar vectors.

If it does not hold, relevance feedback does not work and neither for using clustering for retrieval. It is reasonable and seems true experimentally.

But why a single unique cluster?

7.3.5 Multi-Cluster Hypothesis

Not just one cluster, rather, more than one. At least for some collections (images, video, context, music, etc.). At least for some queries... at least in some cases.

Consequences: Cluster hypothesis is at the basis of:

- Cluster based IRS

- Visualizing clusters of retrieved docs
- Relevance feedback
- Similarity

It is therefore interesting/useful to understand the shape of clusters: unique vs multiple, stretched, dense, sparse, etc.

Can we evaluate if the multi-cluster hypothesis holds? we can calculate some multi-cluster measures:

- R-R: distribution of the similarity values ("inverse of distance") between relevant docs. and relevant docs.
- R-NR: between rel. and non-rel. docs.

8 Evaluation

Let's start with some relevant questions about evaluation:

- **Why?** to compare different IR systems, variants, approaches, algorithms.
- **What?** only the IRS (endosystem)? or with the user (ectosystem)?
- **How?** with or without the user? which measures/metrics?
- **When?** As soon as possible or as often as possible?
- **Where?** In the laboratory (more control)? or in the field (more realism)?

There are some issues related to effectivness calculation that must be taken into consideration: IR systems work at the topicality level but the user is not interested in that, but rather in usefulness/utility. System relevance $\neq$ user relevance. What really counts is user satisfaction; relevance is situational, it depends on the specific situation. Relevance is a primitive concept, it does not need an explicit definition. Relevance is also subjective; relevance assessors disagree. The disagreement between different relevance assessors does not affect the results of evaluation experiments. A good definition of relevance can be: **Relevance is the A of a B existing between a C and a D as determined by an E** where:

- A = Measure, estimate, judgment...
- B = Utility, matching, satisfaction...
- C = Document, doc. representation, information provided...
- D = Question, question representation, information need...
- E = Requester, intermediary, expert...

[Sarecevic, 1975].

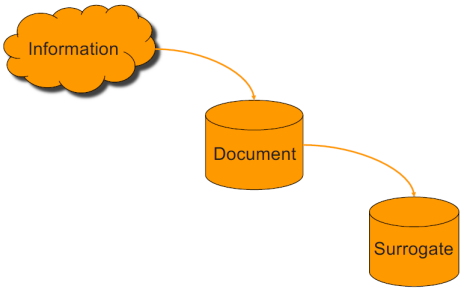
8.1 An attempt of a more systematic classification

Each relevance is a point in a 4D space:

1. Information resources
2. User problem representations
3. Time
4. Components (topic, task, context)

1st dimension [Information Resources]:

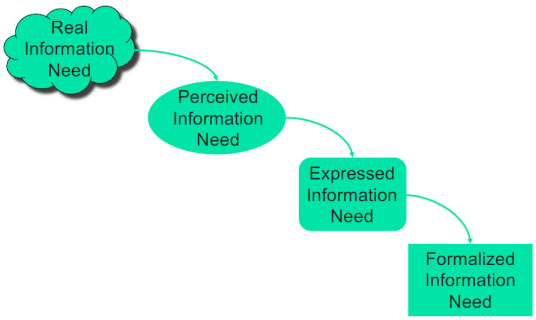
- Surrogate: representation of the document inside an IRS

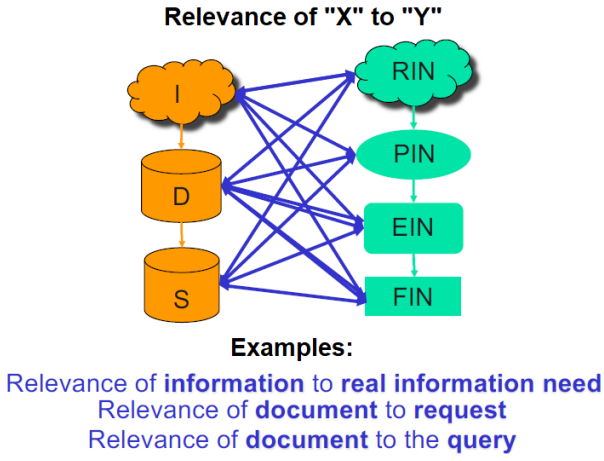


- Document: the physical object that the user has access to
- Information: what the user gets from the document

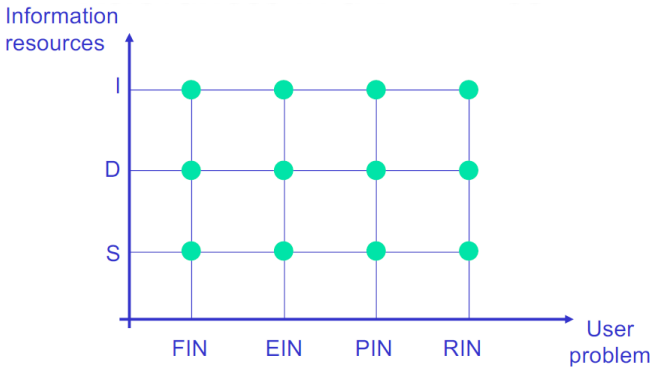
2nd dimension [User Problem Representation]:

- RIN: real information need
- PIN: perceived information need
- R, EIN: request, expressed in natural language, expressed/explicit information need
- Q, FIN: query, formalized information need

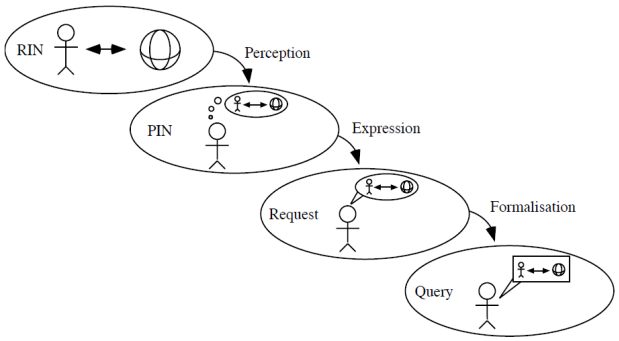




A bit more tidy version of relevances in a 2D space:



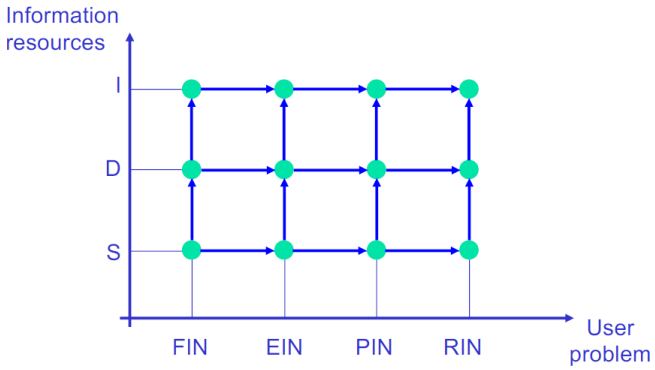
Operations in 2nd dimension:



Problems and difficulties in 2D:

- Perception (RIN → PIN): difficult; ASK, ISK, USK, ...
- Expression (PIN → EIN/R): difficult, label effect, vocabulary problem
- Formalization (EIN/R → FIN/Q): difficult for non natural language, vocabulary problem

Relevance in a 2D space with order:



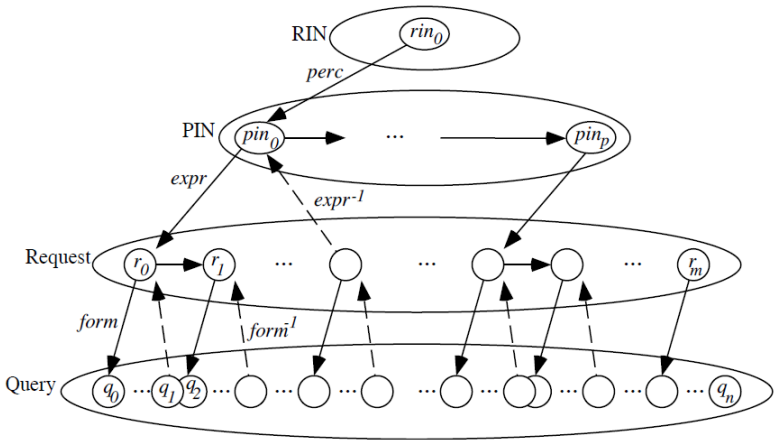
Orders on the single dimensions:

- S < D < I
- Q (EIN)< R (FIN) < PIN < RIN

Induce, cause a (partial) order on the 2D space. Higher relevance are closer to the user and more difficult to measure. Lower relevance are closer to the system and more operational. So, is relevance a point in a 2D space?

3rd dimension [Time]: With 2D everything is static but there are several time points that are interesting:

- time of RIN creation
- time of the first query formalization
- time of the first reformulation
- time when the user leaves the system
- time where the user uses the retrieved docs



4th dimension [Components]:
What about topicality? A document might be topical (ok as far as the topic is concerned) but be not useful anyway because:

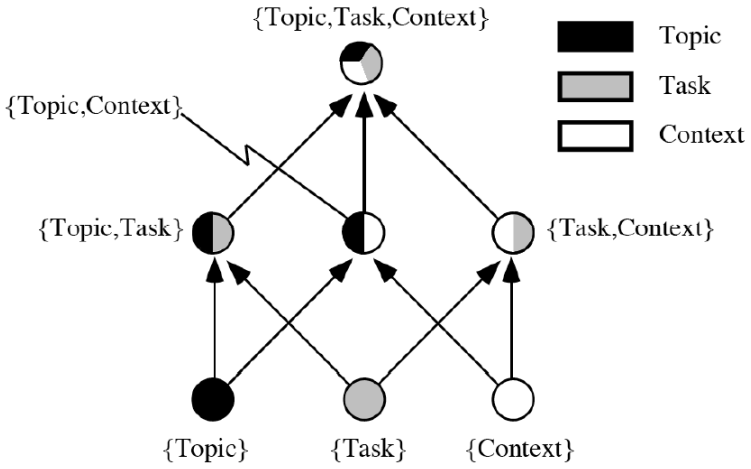
- too long/short/complex/...
- already known
- not understandable by the user
- ...

More sistematically, components of a document are:

- **Topic (To):** the topic, the thematic area, the concepts in a document, what the document is about;
- **Task (Ta):** for what task(s) the document is adequate (urgent tasks will require short documents);
- **Context (Co):** Everything that is neither To nor Ta but affects anyway relevance (e.g. user's background/previous knowledge).

Components and relevance: A, D (or S, I) might be relevant to a RIN (PIN, EIN/R, FIN/Q) with respect to the To (Ta, Co) component but not for the other ones. Or, for any set of components.

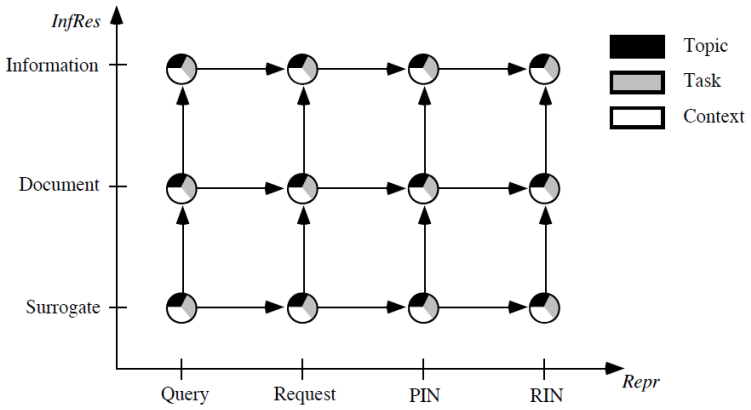
The order is only partial:

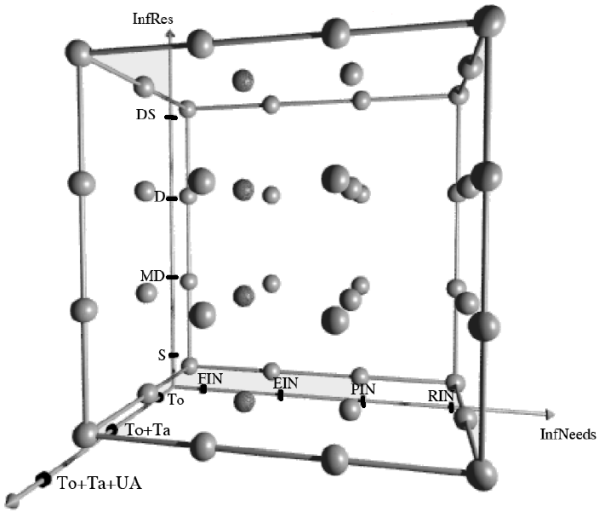


Different relevances:

- r(S,Q,t(q),To): The relevance of the surrogate (representation of the document inside an IRS) to the query at query time for what concerns the topic component only. This is the **classical relevance**; the relevance IRSs work(ed) on.
- r(I,RIN,t(f),{To,Ta,Co}): The relevance the user is interested in. I = Informations of the document, Real Information Need, $t(f)$ = time of formulation, relative to topic, task and context.

Graphical representation:





8.1.1 Beyond topical criteria & Comments

Topicality and system relevance are two conceptually very different things. To is represented by terms in the docs. and Ta is represented (in part) by metadata: doc. length, Publication date, Language, Kind of publication, etc.

Relevance criteria that go beyond the topic: much research has been done, long lists of dozens of criteria. In last studies, very few new criteria: probably almost all of them have been identified.

Examples: reliability, novelty, accessibility, availability, graphical format, numerical data, correctness, clarity, coherence, consistency, pleasure, etc.

Consequences:
Pay attention to which relevance is being used during evaluation. Possible use during relevance feedback: A document that is not relevant for Ta might anyway contain good reformulation terms. A document not relevant for To but relevant for Ta might contain useful metadata for reformulation.

Some vector space models incorporate metadata and in such a case the To and Ta dimensions could/should be processed in different ways.

8.2 Relevance in mobile

With mobile we mean mobile devices. Relevance can be different: sometimes there is not an information need but rather a need of "things/objects". Things to buy or things related to the need in the physical world. Often there is an urgency need.

Besides "what is inside user's mind" also "what is outside the user mind": location, trajectory, speed, noise, light level, congitive load and concentration level etc.

Ta and Co are not only hidden inside user mind, they are automatically derivable; at least in part.

The summary is that we don't really know "what is relevance". There are several relevances and they can be classified. It is important to take that into account when evaluating.

To evaluate with respect either a relevance or another one and it seems even more important for mobile IR than for IR.

8.3 Naive measures for evaluations

Let's start with the simplest naive measures of evaluation that we can think about:

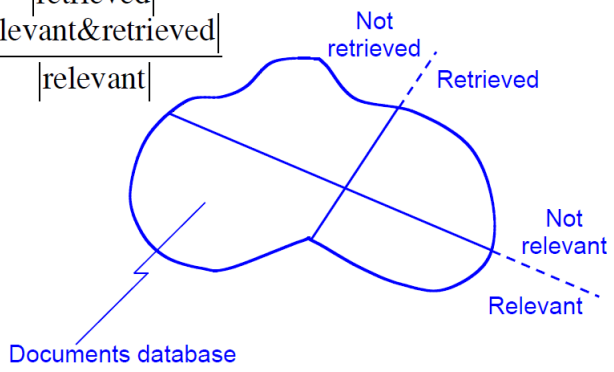
- Count the number of relevant documents among the retrieved ones;
- Count the number of irrelevant documents in the not retrieved.

Both are needed, an IRS that retrieves the whole collection or that retrieve anything is useless. Two very important measures are precision and recall:

- **Precision: P =** high precision of relevant = retrieved relevant/retrieved
- **Recall: R =** quantity of relevant = retrieved relevant/all relevant

$$P = \frac{|\text{relevant\&retrieved}|}{|\text{retrieved}|}$$

$$R = \frac{|\text{relevant\&retrieved}|}{|\text{relevant}|}$$



Retrieved documents are ranked. A system that retrieves the relevant ones in earlier rank positions than another should be rewarded. But P&R do not differentiate. What can be done is to compute them at various stages of retrieval (P/R curve).

	Retrieved	Non retrieved	
Relevant	a	b	$n_1=a+b$
Non relevant	c	d	
	$n_2=a+c$		$N=a+b+c+d$

$$P = \frac{a}{n_2} \qquad R = \frac{a}{n_1}$$

8.3.1 Beyond Precision and Recall

- **Binary relevance:** either a doc. is relevant or it is not
- **Binary retrieval:** either a doc. is retrieved or it is not
- IR systems ranks the retrieved docs. with binary relevance: this is the classical working hypohesis in IR effectiveness evaluation.

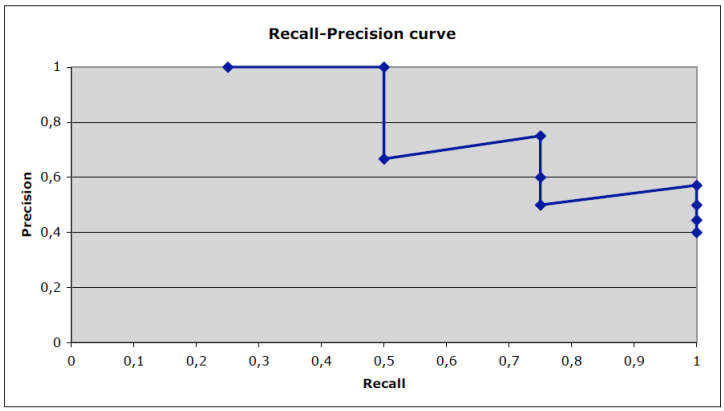
Example:

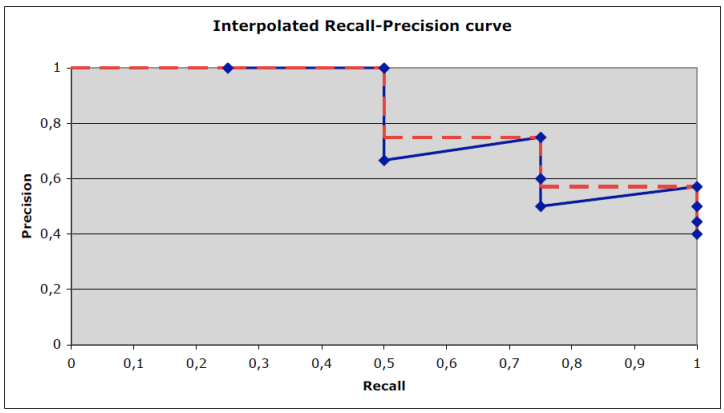
1 = relevant
0 = not relevant
- 4 relevant docs. in the collection

- "Stratified" P e R
(After having retrieved k docs.)

Rank	Rel?	R	P
1	1	**0,25**	**1**
2	1	**0,5**	**1**
3	0	**0,5**	**0,67**
4	1	**0,75**	**0,75**
5	0	**0,75**	**0,6**
6	0	**0,75**	**0,5**
7	1	**1**	**0,57**
8	0	**1**	**0,5**
9	0	**1**	**0,44**
10	0	**1**	**0,4**

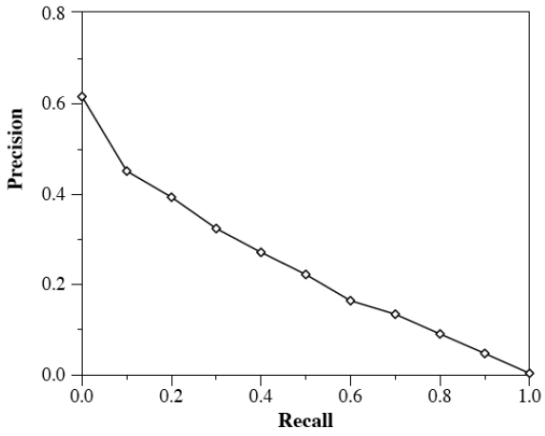
We can graph with R and P:





Over more queries: an average of the curves on each query is computed: at **fixed levels of recall**, usually, 11: 0.0, 0.1, 0.2, ..., 1.0.

Before, each curve is smoothed: from sawtooth curve to step curve then the average of the step curve is computed:



We now have a procedure to understand if an IRS is more effective than another: some needs/queries are chosen, the relevant docs. for each query are found. The two IRS are run on the query set and the one with the highest curve wins. (In reality it's not that simple, we will now see other metrics and advanced evaluation techniques). The R-P curve is just the golden standard to compare two or more IR systems.

8.4 Evaluation Metrics

Other than simple metrics seen in the previous chapter, there are a lot of classic alternatives to P&R. They are a lot (more than 100); we will see the most popular ones.

8.4.1 Alternatives to P and R

The first one is **Fallout (F)**: Proportion of irrelevant & retrieved docs. (with respect to all the irrelevant in the collection). F is like the opposite of the recall, if recall is Ret-Rel/Rel; fallout is Ret-NotRel/Not-Rel. The lower the better; or 1-F for the opposite. It can be seen as the "recall of non-relevant docs."; it must be lower as possible.

	Retrieved	Non Retrieved	
Relevant	a	b	$n_1=a+b$
Non relevant	c	d	$N-n_1$
	$n_2=a+c$		$N=a+b+c+d$

$$P = \frac{a}{n_2} \quad R = \frac{a}{n_1} \qquad F = \frac{c}{N - n_1}$$

Another one is **Generality factor (G)**: Proportion of relevant docs. (independent from retrieval, it depends only on the query). Relevants for the query/all docs.

	Retrieved	Non retrieved	
Relevant	a	b	$n_1=a+b$
Non relevant	c	d	
	$n_2=a+c$		$N=a+b+c+d$

$$P = \frac{a}{n_2} \quad R = \frac{a}{n_1} \quad F = \frac{c}{N - n_1} \qquad G = \frac{n_1}{N}$$

The relation between P, R, F and G:

$$\frac{R}{F} = \frac{P/(1-P)}{G/(1-G)}$$

$$RG(1-P) = FP(1-G)$$

Then we might well be not using the P&R pair. Other pairs are equivalent.

Critiques to P and R: There is no certainty on relevant documents in the collection: Precision P can be computed exactly, Recall R cannot. Maybe in some test collections, but not in practice (user using a system on his/her own need).

It is not clear what is more important for the user, P or R? Usually, Precision is most important; But for some needs/cases/users, Recall can be more important (medical, legal, etc.) while for others, Precision is more important (Web, MobileIR, etc.).

The relation between P & R and user satisfaction is unclear. Binary relevance is another problem. Not a single number for comparison, but a P-R Curve (not very practical).

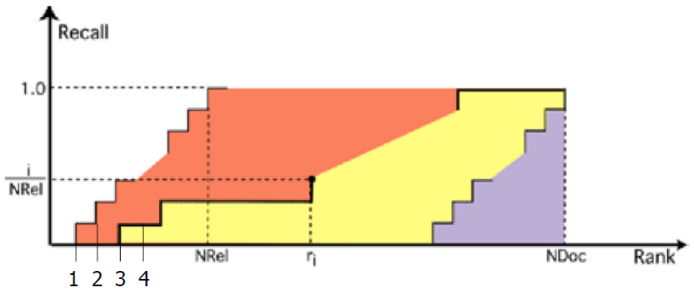
Averages of P and R

F is the harmonic mean of P and R. Equivalence (if $b = 1$ then $E = 1 - F$). Other variants exist.

$$F = \frac{2}{\frac{1}{R} + \frac{1}{P}} = \frac{2RP}{R+P} \qquad E = 1 - \frac{1+b^2}{\frac{b^2}{R} + \frac{1}{P}}$$

8.4.2 Normalized Recall (and P)

Both curve and single number. Orange area = ideal, alla relevant docs. retrieved first. Actual = yellow (middle case), worst case = purple.
If there are n relevants, every retrieved relevant will produce a "step" of height $1/n$.



8.4.3 Expected Search Length

ESL(N) = number of docs. that one needs to examine (according to the rank) to have N relevant docs. It is not a query dependent number as P or R. It is a function that given the desired number of relevant docs. returns how many docs. are needed to be examined.

- Expected: because the average over queries is computed;
- Length: in terms of docs. to be examinated.

The definition is valid also if the set of retrieved docs. is only partially ordered (rather than a total order). Some computations are needed (because the documents are examined sequentially).

8.4.4 Average Precision

Rank of retrieved docs, "sliced" at ranks. The precision values for each relevant document is considered (0 if the doc. is not retrieved).

Their average is computed. It is like, but not as the average of the steps of the P/R curve (normalized). Since the abscissa is 1, it is the area under the P/R curve.

Example:

- Average of P values at the rank levels corresponding to relevant & retrieved docs
  - If not retrieved, P = 0, then AvgPrec decreases if not all the relevant are retrieved
  - IT IS NOT the average at the standard 11 R levels!

Rank	Rel?	R	P
1	**1**	0,25	**1**
2	**1**	0,5	**1**
3	0	0,5	0,67
4	**1**	0,75	**0,75**
5	0	0,75	0,6
6	0	0,75	0,5
7	**1**	1	**0,57**
8	0	1	0,5
9	0	1	0,44
10	0	1	0,4
			0,83

8.4.5 MAP - Mean Average Precision

Terminology:

- Average Precision (AP) is for one query;
- MAP is the average of AP values over more queries;
- Often, usually, it is called simply average precision, or uninterpolated MAP.

Interpolated MAP is something different:

- It is the average of the average P values at the 11 standard recall levels (0; 0.1; 0.2; ...; 1.0). It is much less used.

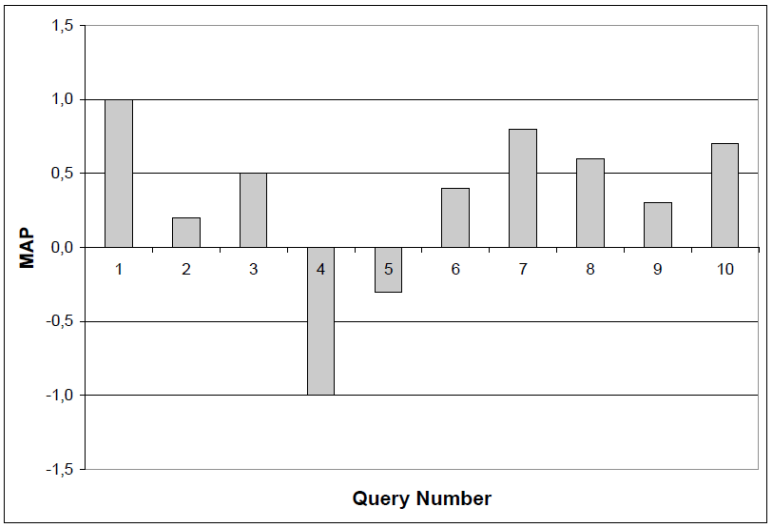
8.4.6 Histograms

MAP, or rather AP, is often shown as histograms, to highlight a system behavior.

- Over more queries and/or specific queries;
- With respect to other systems.

One bar for each query. Each bar shows the displacement plus or minus from the mean (median) over all systems (usually there is high variation over different queries). They are useful for analysis on specific queries.

An example of AP histogram:



8.4.7 R-precision

It is the precision at the Rth-rank position where R is the number of relevant docs.

If a query has R relevant docs, the precision after the first R retrieved docs. is:

- Max = 1 (if the first R are all relevant)
- Min = 0 (if the first R are all irrelevant)

8.4.8 Other Metrics

P@n with n=1,5,10,. . . ,N

- Precision value after N docs. have been retrieved;
- n=10 often used for Web search;
- n=1 useful for "I'm feelking lucky".

R-precision: P@R = Precision after R documents (R=number of relevant).

8.4.9 DCG - Discontinued Cumulative Gain

Category relevance, ranked retrieval in which we have N relevance levels: 0,1,2,...,N-1. The highly relevant docs. should be retrieved in the early rank positions (because the user gains more).

DCG measuers the gain that a doc. gives to the user "discounting" by $log(rank)$. Usual average over queries. It has a problem: queries with more relevant docs. count more than queries with fewer relevant docs. More gain for queries with a lot of relevance. The solution is NDCG.

8.4.10 NDCG - Normalized DCG

It performs normalization to avoid that queries with more relevant docs. count more than queries with fewer relevant docs. Actual DCG value is divided by the ideal DCG i.e., the value of the ideal rank. Then the average among values on the same scales (all having 1 as maximum) is computed.

8.4.11 Sliding ratio

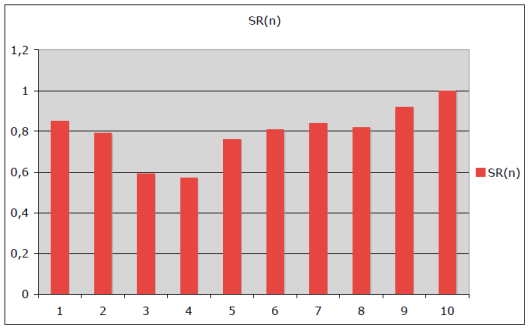
Relevance values here are not binary, not categories, but continuous values (Relevance Values). The ideal system should rank the retrieved docs. in decreasing order of relevance values. The actual system will do worse. Sliding ratio measeures the distance between ideal and actual systems.

- w_i are the real/actual
- W_i the ideal

$$SR(n) = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} W_i}$$

Example:

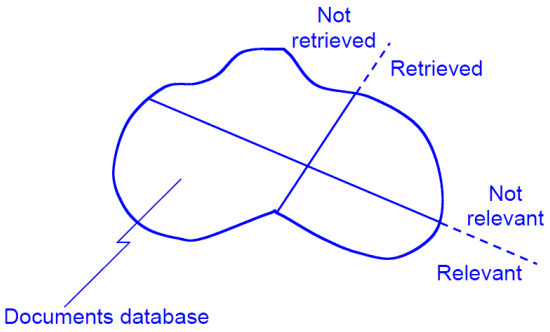
	Actual		Ideal		SR(n)
n	w_i	sum(wi)	W_i	sum(Wi)	
1	7.0	7.0	8.2	8.2	0.85
2	5.0	12.0	7.0	15.2	0.79
3	0.0	12.0	5.2	20.4	0.59
4	2.5	14.5	5.0	25.4	0.57
5	8.2	22.7	4.5	29.9	0.76
6	4.5	27.2	3.7	33.6	0.81
7	3.7	30.9	3.1	36.7	0.84
8	1.1	32.0	2.5	39.2	0.82
9	5.2	37.2	1.1	40.3	0.92
10	3.1	40.3	0.0	40.3	1.00



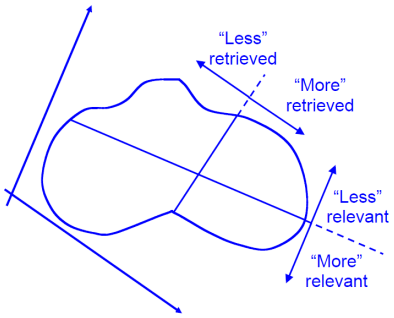
8.4.12 ADM - Average Distance Measure

ADM is a less classic metric. But before, let's see some concepts.

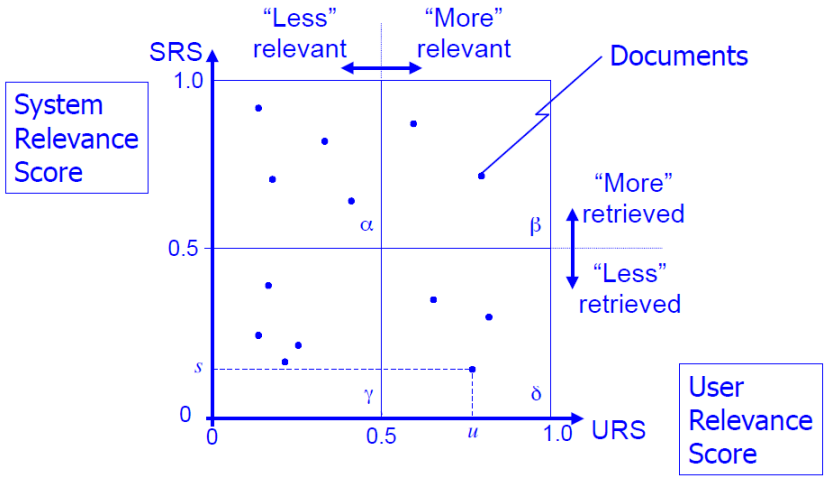
From binary releveance...



...to continuous relevance:

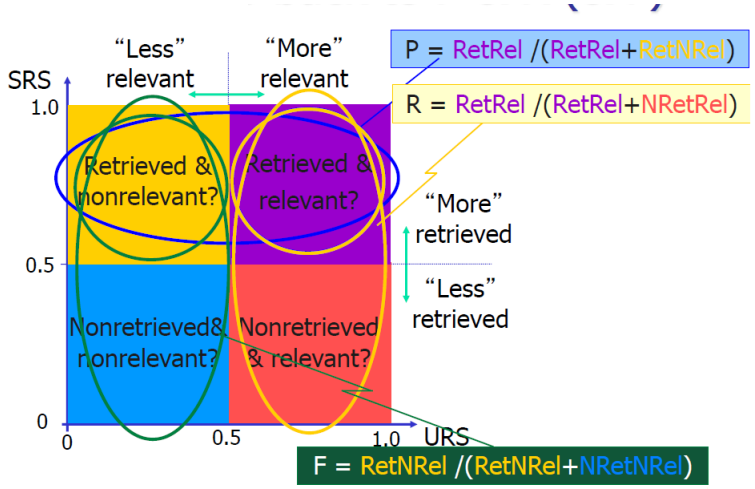


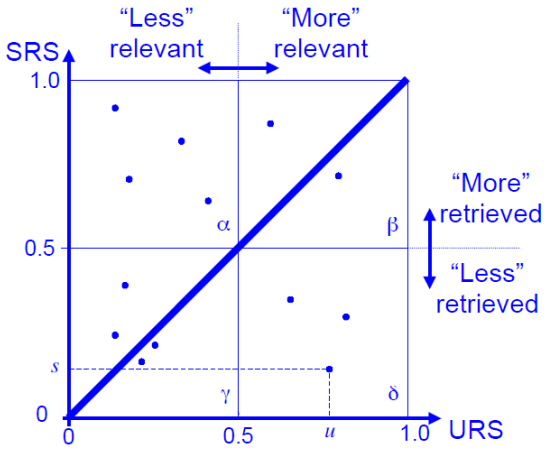
The cartesian plane **URS/SRS**:



- **SRS:** System relevance score: relevance value as estimated by the system;
- **URS:** User relevance score: relevance value estiamted by the user.

They are real numbers, normalized between 0 and 1. They are different from RSVs (Retrieval Status Values), insensible to any order-preserving, monotic change. Estimate of the probability of relevance.

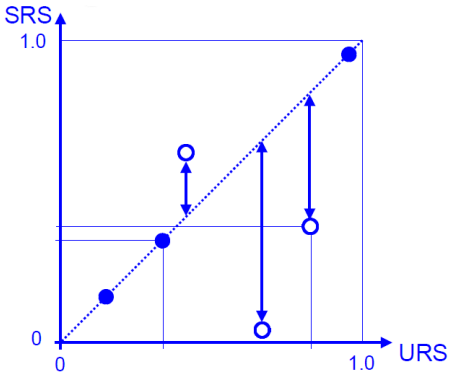




The ideal system should give the same relevance to a document as the relevance given by the user. Which means, that SRS should be equal to URS. The perfect system will have points only in the highlated diagonal line of the graph.

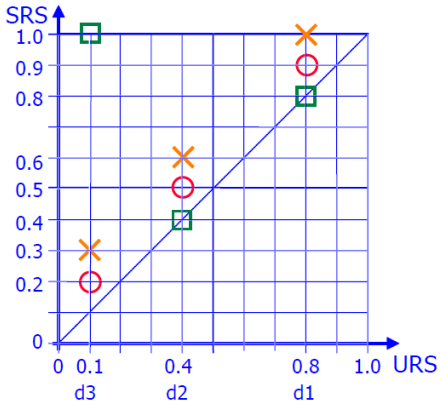
The more the difference, the worst the system is in evaluating relevance. The concept is similar to the mean-squared-error in a linear-model.

ADM - Average Distance Measure:
ADM for one query is equal to 1− avg. of the distances between SRS and USR over all the docs. ADM for an IRS: average over all the queries.



$$ADM_q = 1 - \frac{\sum_{d_i \in D} \left| SRS_q(d_i) - URS_q(d_i) \right|}{|D|}$$

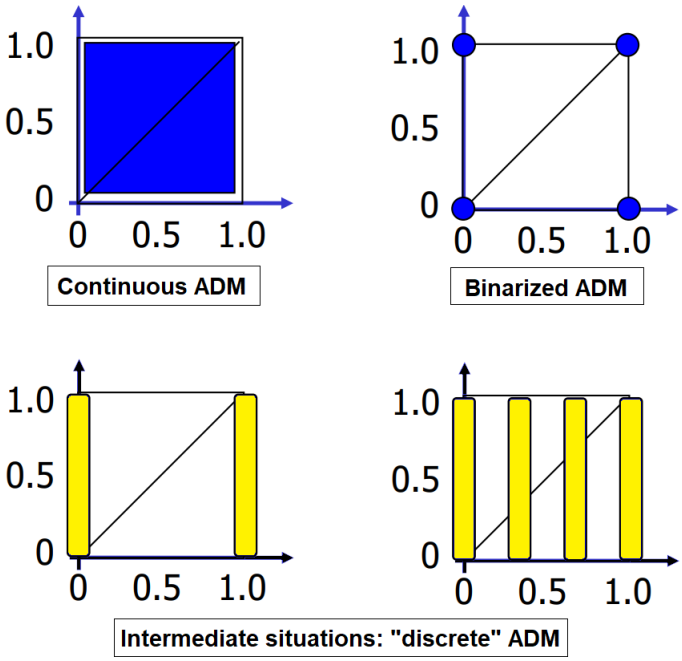
Example:



Docs.	d1	d2	d3	ADM
URS	0.8	0.4	0.1	
IRS1 ○	0.9	0.5	0.2	0.9
IRS2 ×	1.0	0.6	0.3	0.8
IRS3 □	0.8	0.4	1.0	0.7

What is needed for ADM?

- The ideal situation: continuous SRS and URS.
- Worst situation: binarized ADM.
- Intermediate situations: discrete ADM.



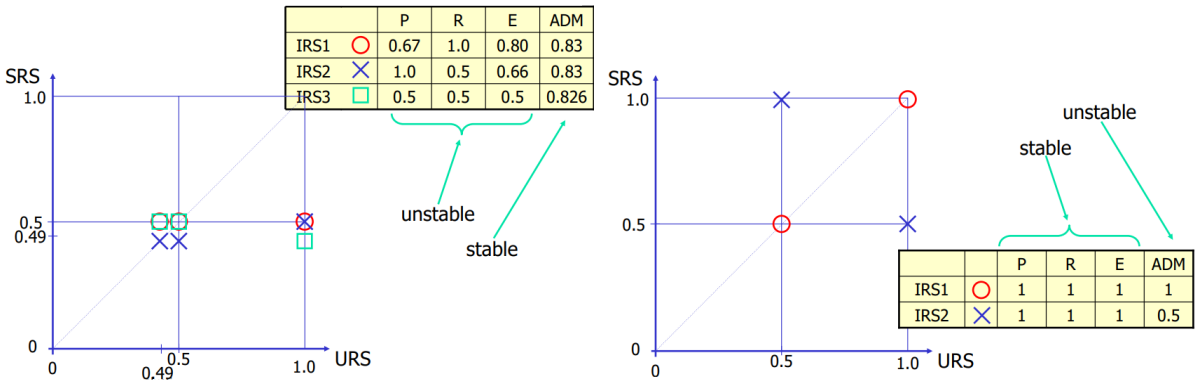
Intermediate situations can be: categories, one of the two (URS or SRS) binary/discrete and the other one continuous.

ADM vs P and R

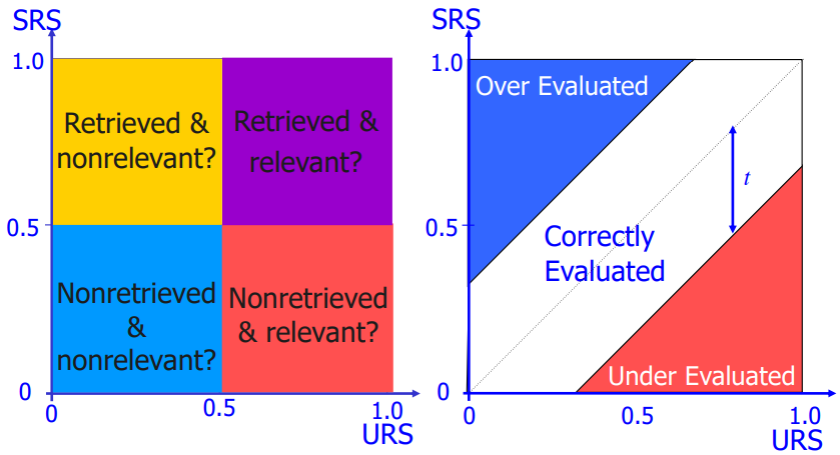
Precision and recall are:

- Hypersensitive to the relevant/not relevant and retrieved/not retrieved thresholds (borders).
- Insensitive to variations within some areas.

To the left: hypersensitive; to the right: insensitive:



Summary:



The **main problem** with ADM is that it's not Top-Heavy (higher rank document are not associated with higher rank).

8.5 Test Collection and User Study

We will now see two approaches (or, evaluation methodologies to evaluation): Test collection and User study (plus Log analysis).

- **Test collection** (benchmark): A test collection is built with a set of docs, set of requests (written description of info needs), set of relevant docs. for each request, as decided by human judges/assessors. Different systems are compared: each one searches for every request. Effectiveness is measured (with some metrics).
- **User study**: some users are required; they are asked to use a system. Effectiveness is measured as well as user satisfaction, fatigue, etc. This is done in laboratory (choose participants, bring them in the lab; realistic situation; with true needs or induced ones). On the field: real life (gathering data of user using a sytem on their own needs).

Test collection:

- Pro: I have the relevant docs. (not always/all of them...); replicability; less if no effort for GUI.
- Cons: Requests; not information needs; "fake" relevance (assessed by third parties on requests).

User studies:

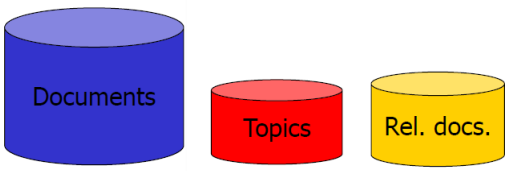
- Pro: More realistic; cognitive aspects; less effort for indexing.
- Cons: effort, lower replicability; no recall measurements.

Clearly, they complement each other. Both of them are valid, better if done both.

8.5.1 Test collections

How a test collection is made:

- Document collection;
- Set of request (topics);
- For each request, set of relevant docs: binary judgments, or category judgments; or "qrels" (file in which for each topic there's the list of relevant documents).



1st generation: small collection, Cranfield, ISI, CACM; 2nd gen.: larger document collection with pooling, competitions, TREC (1st one of 2nd gen.; most important one); 3rd gen.: TREC, NTCIR, CLEF, INEX; FIRE.

Pooling: 1st generation test collection were small. With some patience, (almost) all relevant docs. could be found. This is becoming always more difficult. From 2nd generation **pooling** has been used.

- The first x retrieved docs. by each system are considered;
- All of them together form a "pool";
- Only the docs. in the pool are evaluated; the other ones are considered not relevant;
- Hope/conjecture: a relevant doc. will be retrieved by at least one system (and in the first x rank position).
- No pooling without contemporary participation;
- Need of pooling when the collection is large.

Most important test collections/competitions:

- TREC: Text REtrieval Conference;
- NTCIR: Nii Test Collection for Information Retrieval systems (NII: National institute of Informatics; Japan);
- CLEF: Cross Language Evaluation Forum;
- INEX: INitiative for the Evaluation of XML retrieval.

On the utility of test collections: they're useful and objective. Repeatability; benchmark. TREC has led to a significant increase of IRSs effectivness. Huge amount of data to work with.

8.5.2 User Studies

Example of measuered variables:

- Performance;
- User satisfaction;
- User behavior (logging, videorecording).

8.5.3 On the importance of evaluation

Evaluation is fundamental in Information Retrieval. It has a strong tradion, based on scientific methods.

8.5.4 Test collections and user study: pros and cons

Test collection pros:

- If many systems search a collection, even a large one, (almost) all the relevant documents will be found (but not for veeery large ones (TeraByte)).
- Replicability: precise context; by re-doing the experiment I will obtain the same results.
- Huge amount of data to be used.

User studies pros:

- The system is evaluated in a more realistic setting, with real/realistic users, almost on the field.
- Cognitive variables like user satisfaction, fatigue, etc. can be measured.

Test collection cons:

- Relevance judged by the assessor is neither system relevance nor user relevance (it's "fake").
- Requests (topics) are not real information needs, they are induced.
- Effort: need to index very large collections.

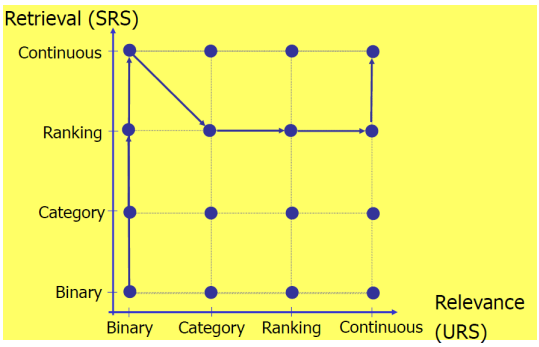
User studies cons:

- There is not a set of relevant documents. Can't measure recall.
- Replicability is more difficult (with other participants/users or even the same ones if the experiment is repeated - different results can be obtained). It is less objective because there is some variability.
- Effort in organizing the experiment.

8.6 Metric classificaiton

Measures of retrieval effectivness: classification on the basis of underlying notions:

- Relevance (binary, ranking, continuous);
- Retrieval (binary, ranking, continuous).



- Binary relevance - Binary retieval: precision, recall, fallout, e-measure
- Binary relevance - Ranking retrieval: P/R curves, Normalized P and R, Excepted search length, MAP, ...
- Binary relevance - continuous retrieval: System relevance Score (SRS) $\in$ [0..1] which is differente from Retrieval Status Value (RSV), used for ranking. Swet's E-measure.
- Category relevance - Ranking retrieval: nDCG,RHL,...
- Ranking relevance - Ranking retrieval: preference, Yao's ndpm,...
- Continuous relevance - Ranking retrieal: sliding ratio
- Continuous relevance - Continuous retrieval: User Relevance Score (URS), the "real" relevance value; ADM

Year	Relevance:	Binary			Rank			Cont.			TREC	INEX	NTCIR
	Retrieval:	B	R	C	B	R	C	B	R	C			
1960	**Precision**	•											
	Recall	•											
	Fallout	•											
	Generality F.	•											
1965	**E-(F-)Measure**	•											
	R-P curve		•								•	•	•
	R-Fallout curve		•										
	Normalized R		•										
	Normalized P		•										
	ESL		•										
	Sliding Ratio								•				
1970	Novelty Ratio	•											
	Coverage Ratio	•											
	Relative Recall	•											
	Recall Effort	•											
	Utility	•											
1975	**MAP**		•								•	•	•
	P@N		•								•	•	•
	R-Precision		•								•		•
	Interpolated MAP		•										
1990	Satisfaction								•				
	Frustration								•				
	Total								•				
	Usefulness					•							
	ASL		•										

Year	Relevance:	Binary			Rank			Cont.			TREC	INEX	NTCIR
	Retrieval:	B	R	C	B	R	C	B	R	C			
1995	NDPM					•							
	Ranked Half Life									•			
	Relative Relevance					•							
2000	Classif. accuracy	•											
	DCG		•			•			×				
	AWP		•			•							
	Weighted R-Prec.		•			•							
	ADM	•	•	•	•	•	•	•	•	•			
	XCG		•			•			×			•	
	bpref		•										
	Q-measure		•			•						•	
	R-measure		•			•						•	
	Tolerance to irrelev.		•									•	
	Estimated Ratio of Relevance		•									•	
	Kendall, Spearman					•							
	Normalized xCG		•			•						•	
	Mean average nxCG at rank n		•			•						•	
	Effort-prec./gain-rec. @std. gain-rec. point		•			•						•	
	Non-interpolated mean average effort-prec.		•			•						•	
	Interpolated mean average effort-prec.		•			•						•	

9 WEB Information Retrieval

WIR has a lot of differences if compared with "classic" IR (what we've seen since now):

- Dimension of the collection (the web);
- Different contents formats;
- The collection is extremely dynamic;
- Multilingual contents;
- Spam and quality;
- Social aspects;
- Geographic aspects;
- etc.

The consequence is that standard IR techniques can be used, but not only and not always. Search Engines for the web are called Web Search Engines.

"Classic" IR is 50+ years old:

- Mainly concerned with bibliographic collections and/or scholarly papers;
- Queries were often formulated by an intermediary ("librarian");
- User is often the customer of a library, or a scholar.

WIR is younger, it is about 20 years old:

- Different collection(s);
- Different users;
- IR is a good starting point for WIR but it is not enough.

Let's now see some WIR characteristics in detail.

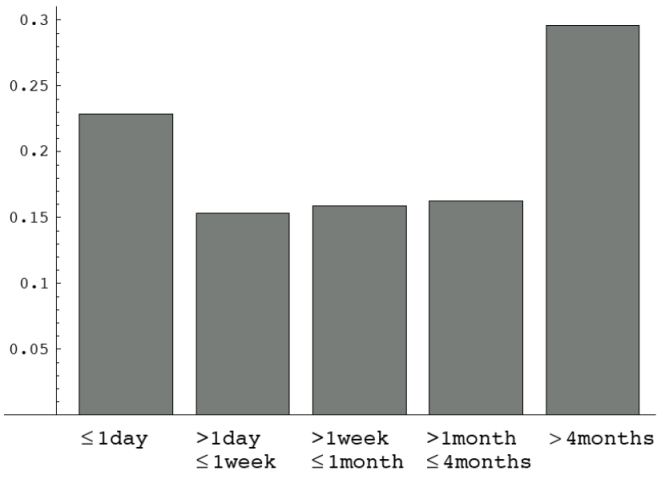
Heterogeneous collection
The collection; in WIR; is heterogeneus. In fact on the web we can have/find:

- Different languages: hundreds, even thousands (dialects, etc.); encodings, etc.;
- Different media (multimedia): not only text documents;
- Different formats and standards;
- Different topics. The documents are not all about the same topic or kind of topics;

Size
The size of the web is huge, we have billions of pages, web servers, IPs. Even more if we also consider the "Deep Web".

Dynamic collection
The "Web Collection" is very dynamic, it changes very fast and in every moment:

- New pages continuously added, deleted or modified.
- Pages can be dynamic (php, JSP, ...) which means that they can be theoretically infinite.



Other difficulties related to pages can be:

- Duplicated pages;
- Spam pages (web pages to cheat SEs and users – fake keywords etc.).

Information need classificaiton
There are three general categories of web queries (different types of information need):

- **Informational** (about 40%): User wants to learn something on some topic;
- **Navigational** (about 25%): User wants to go to a specific web page;
- **Transactional** (about 35%): User wants to do something (on the web): buy something, access a certain service (e.g. get weather prediction), etc.

User diversity
Usually, on the web, we have different users with different informations needs, expectations, background knowledge, bandwidth, behavior and so on. They can differ a lot.

9.1 Anatomy of a Web Search Engine

A search engine for the web has three main modules/components:

- **Crawler** (spider, robot)
- **Indexer**
- **Query processor**

Crawler: Extracts from the web collection the items to be indexed. It performs recursive web browsing and for each known URL it fetches the page and execute parsing (extracts URLS: new known urls - and passes pages/terms to the indexer).

There are several policies for managing new pages URLs: direct submission, random URL generation, etc.

Indexer: It processes the data downloaded by the crawler and builds the inverted index. Classical IR techniques can be used: stemming, phrases, capitalization, stopwords, etc.

Query processor: It accepts the query from the user and return the answers (documents). It has a user front end (UI, etc.) and a back end (interaction with the index, etc.)

An obvious issue for all three modules is run-time efficiency/performance: they work with a huge amount of data (10 thousand queries-requests/sec).

10 The Web Graph

The topology of the web can be seen as a graph. But what's the number of nodes? the shape? number of arcs? etc.

We will use this formal definition for the web graph:

$$G = (N, A)$$

$$N = \text{ set of nodes (vertex)}$$

$$A = \text{ set of arcs (edges)}$$

Nodes $n \in N$ and Arcs $a \in A$ could be seen as:

- Web pages and HTML links;
- Web pages and cosine similarity between pages;
- Web hosts and HTML links between different hosts;
- Physical hosts and physical network connections.

The Web graph we use for IR is **Web pages** and **HTML links**.

First question: $|N|$**?**
How we can compute the number of nodes? today it's estimated to be greater than 20 billion.

The naive approach consists to start crawling and counting and to stop crawling when we can't find any new pages. This approach has some problems: dups, web is "infinite", when finished it's too late (dynamic, time) and what if the web graph is not connected?

The web is actually infinite because of some website with infinite (generated) pages.

We can give an operational definition of $|N|$. It is the number of pages indexed/able by at least a search engine: "indexable web"; "index size of a search engine".

There are several techniques to estimate $|N|$:

1. **Public declarations** (by SEs)
2. **Capture & recapture** ("how many fish in a lake" technique): capture 100 fishes, label them, leave them free. Capture 1000 more fishes. How many are labeled? if a few then there are a lot of more fishes. If a lot, there are a few more fishes. This works with the random sampling assumption.
3. **Relative SEs Sizes** (random queries or random searches): We works on intersections and unions of indexed pages by SEs, by doing so we can estimate the total amount. It can be done with SEs cooperation (declarations) or by doing random queries and working on the retrieved document sets.
4. **Random IP addresses** (HTTP requests at random)

5. **Random walks**: start from a web page (node) and at each step we go to another page, which is the probability to be in a given node n after $t \to \text{inf}$?

We should be cautious with those techniques because they are gross simplifications. They are acceptable estimates but not final recipes for the right answer.

Second question: what's the shape of the web graph?
Which shape/topology has the web graph? is it random? does it follow a hierarchy?

Surely it is not a regular well organized graph. It has been hypothesized that the web is a random graph. If we create nodes and arcs randomly we obtain an interesting mathematical object.

When an N nodes random graph is connected? When the average of number of arcs per node is greater than $ln(N)$ (which is actually a small number).

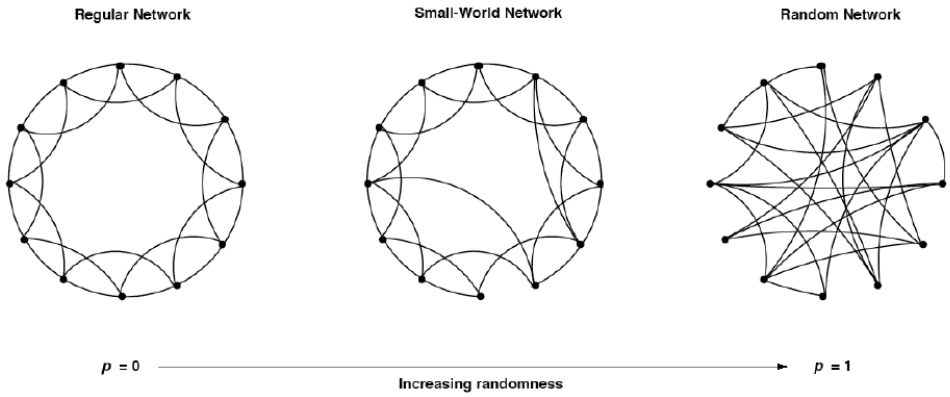
Actually, **the web graph is neither regular nor random**.

- Nodes are not random: They are affected by real world entities;
- Links are not random: They reflect social relationships, already existing links, etc.;

As for the regular vs random graphs we have that:

- Features of a regular graph: Long "shortest paths"; Neighbors ("friends") know each other;
- Features of a random graph: Short "shortest paths"; Neighbors ("friends") do not know each other;
- Is there something "in between"? Both of them does not reflect reality.

Small world networks are halfway between the two classical topologies: regular and random graphs.



The procedure for creating a small world network is:

- to start from a regular ring network
- to take each arc and with probability p re-wire it randomly
  - $p = 0$: regular network (well organized)
  - $p = 1$: random network (not well organized)
  - p in between 0 and 1: small-world network

The networks we "work on" (the web) has:

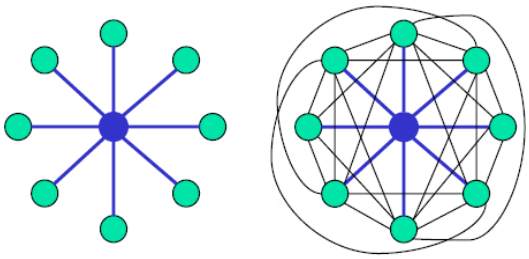
- n = number of nodes
- k = number of arcs per node ($nk/2$ = total number of arcs)
- $n >> k >> ln(n) >> 1$
  - $n >> 1$: n very large (large network)
  - $n >> k$: sparse networks
  - $k >> ln(n)$: not sparse enough to become disconnected

Two measures: L and C
L = average length of shortest path between two nodes.

- Path length = number of arcs
- $L(u, v)$ = length of shortest path between u and v
- L = average on all (u, v) pairs
- $L(p)$: how does L vary as a function of p?

C = clustering coefficient.

- Consider a node v and its neighbors (number of neighbors: k_v)
- How many arcs among the neighbors?
  - Max: $(k_v(k_v - 1))/2$
  - Let C_v be the fraction of these arcs that actually exists
- C = average C_v on all nodes of the network
- $C(p)$: how does C vary as a function of p?



L and C: intuition

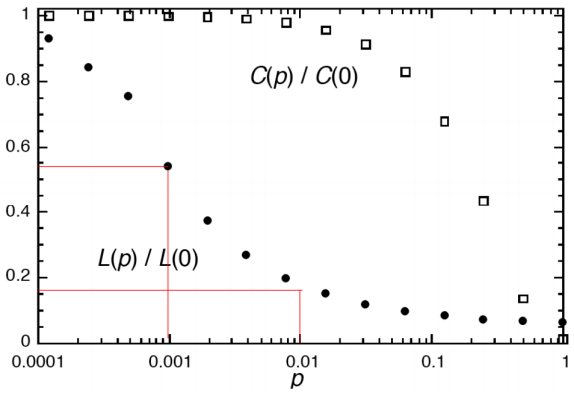
Consider a friendship networks: nodes = persons and arcs = friendship (acquaintance) relations. L measures how many persons (friends, then friends of friends, then etc.) I have to go "through" to reach a person. C measures the "clique level": how much friends of a person are friends of each other. L is "global", C is "local".

Experiments on L(p) and C(p)

To the limits we have:

- $p = 0$ (regular) then:
  - $L(0)$ about $n/2k >> 1$: long distances (large world)
  - $C(0)$ about $3/4$: very clique-ish
- $p = 1$ (random) then:
  - $L(1)$ about $ln(n)/ln(k)$: short distances (small world)
  - $C(1)$ about $k/n << 1$: not very clique-ish

Is L large iff C is large? no is the answer:



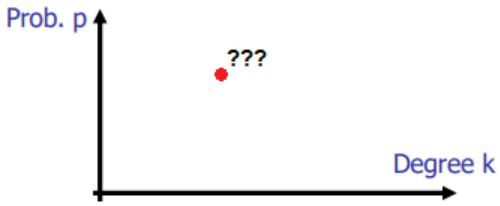
- $L(p)/L(0)$ decreases as p increases: small world;
- $C(p)/C(0)$ decreases as p increases.

Degree distribution

Degree of a node: number of links, in and out links.

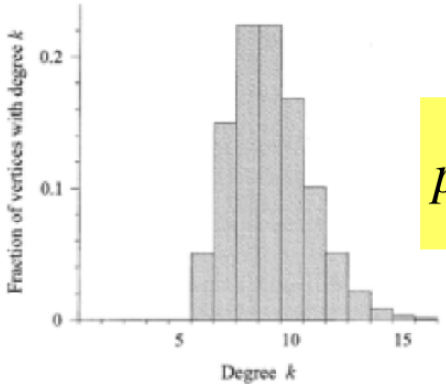
What's the probability to find a node with a given degree?

- Regular networks: usually all nodes have same degree, or very similar;
- Random networks: poisson distribution;
- Small world networks: poisson but with lower cutoff.



Small world: degree distribution

As we said, small world networks have degree with Poisson distribution:



$$p_k = e^{-cp}\frac{(cp)^{k-c}}{(k-c)!}$$

It has a "strange" distribution with a peak and lower threshold/cutoff. It is very different from real networks.

So, if we summarize our results on small world networks:

- Explain the "small world" effect: (diameter L low), not explained by regular networks, but simply "inherited" from the "randomness".
- Explain the "clicquish nature", (clustering coefficient C high), not explained by random networks, but simply "inherited" from the "regularity".
- They are an interesting "mix": but they do not explain the degree distribution found "in nature.

10.0.1 Results from 3 studies about web shape: summary

We will now summarize three papers:

- Barabasi & co. - small world;
- Broder & co. - bow-tie shape;
- Bharat & co. - Web hostgraph;

1st study (Barabasi & Co.)

They used a crawl set in order to study on the distribution of:

- Outdegree $P_{out}(k)$ = probability that a page has k out-link;
- Indegree $P_{in}(k)$ = probability that a page has k in-degree.

The result of the study is that they both follow a **power-law** distribution; which is different from the distribution for random networks and small worlds networks.

The distribution, by the way, has a "long tail": a few pages has high number of inlink and outlink.

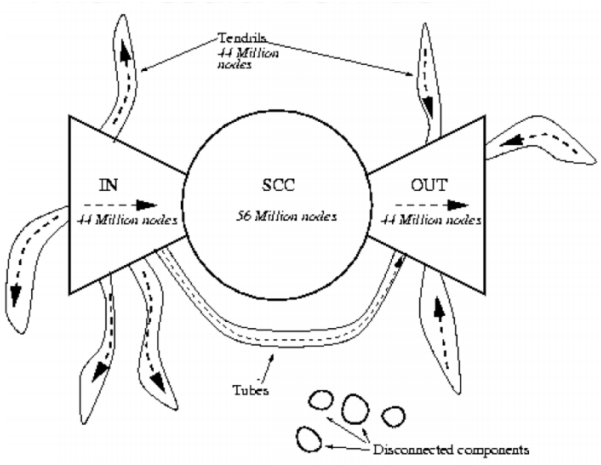
After those results, they built an artificial network with the same number of nodes and same distributions of (previously found) $P_{out}(k)$ and $P_{in}(k)$. On this network they measured the diameter (average distance between nodes). The result is that the diameter has a logarithmic growth in respect to the number of nodes N. This proves that the web is like small world networks.

2nd study (Broder & Co.)

The study consisted in a "big-web-crawl" experiment for checking power-law and small world hypothesis.

The result was:

- Power-law distribution hypothesis confirmed; but actually for in-degree is **Zipf law** (which has better "match"). For out-degree it is powerlaw but different for low levels.
- As for the shape, it's not a small world, but rather a more complex structure **"Bow-tie"** shape. The study was based in analyzing WCCs and SCCs of the web graph. They also find out that the biggest WCC is composed by about 91% of the web nodes/pages. The largest SCC is about 28%.



- **SCC**: set of nodes for which there's always a direct path between two of them.
- **WCC**: set of nodes for which there's always a in-direct (direction of arrows is ignored) path between two of them. They are usually bigger than SCCs.

3rd study (Bharat & Co.)
It was focused on the objective of considering the website graph insted of the page graph. Nodes were websites and arcs were links between two different pages.

The result was mainly a list of confirmations of previous results/hypothesis; one of this is that in and out degree (on the website graph) have zipf distribution.

11 LAR: Link Analysis for Ranking

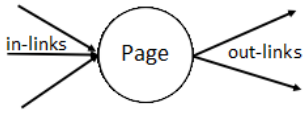
Link Analysis for Ranking is about using links data to help the ranking phase of documents. There are 3 main approaches to LAR:

- **Naive**
- **PageRank** (query independent)
- **HITS**, Kleinberg (query dependent)

11.1 Naive LAR

It was used by the first generation of SEs. There are 2 naive approches for calculating the score (related to links data) of a page:

- **Indirect popularity**: score of a page = number of in-link + number of out-link
- **Direct popularity**: score of a page = number of in-link



Pages are retrieved on the basis of the topic (page content) and ranked on the basis of their popularity. There are some problems with this approach:

- Easily spammable: very easy for out-links but also for in-links (one can create fake pages with links);

First generation of SEs failed because they did not exploited conenctivity and links info. So, let's see some more refined alternatives: **PageRanks** and **HITS**.

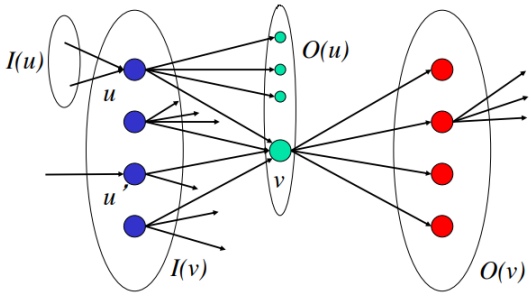
11.2 PageRank

PageRank is also called/considered "The Fortune of Google". The value PageRank of page u is high if u is linked from pages with high PageRank (it's a recursive definition). It increase for u if it is linked:

- from many pages;
- from pages having high PageRank;
- both: from many pages with high PageRank.

It is **query independent** because each web page has its own "quality" score. Query does not influence the pagerank score of a given page.

PageRank: let's call it $r(v)$. The formula is:



$$\mathbf{r}(v) \approx \sum_{u \in I(v)} \frac{\mathbf{r}(u)}{|O(u)|}$$

Notation:

- $r(v)$: PageRank of page v
- $I(v)$: set of pages that link v (in-links)
- $O(v)$: set of pages that are linked from v (out-links)

Each page u has its PageRank value $r(u)$. Page v receives from page u a portion of its PageRank equal to the portion received by any other page linked from u:

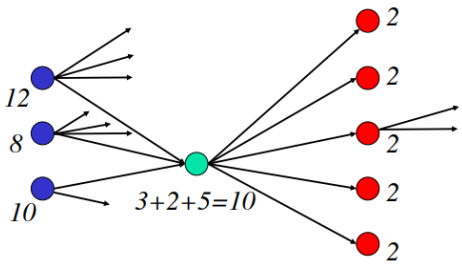
$$\frac{\mathbf{r}(u)}{|O(u)|}$$

Each page u distributes its PageRank value $r(u)$ to the linked pages (those in $O(u)$ set).

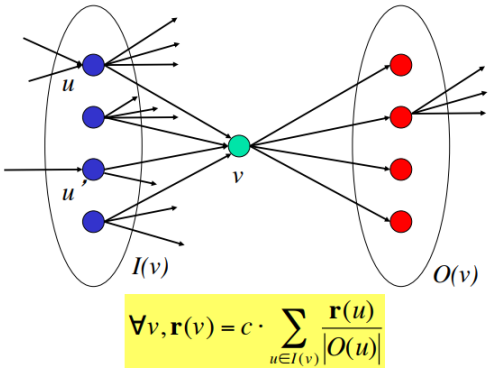
- $+\ r(u)$ is high: $+$ pages in $O(u)$ gain. If $r(u) > r(u')$ and $|O(u)| = |O(u')|$, v gains more from u than from u'.
- $+\ O(u)$ is low, $+$ pages in $O(u)$ gain (because they don't have to "share" $r(u)$ with others). If $r(u) = r(u')$ and $|O(u)| > |O(u')|$, v gains more from u' than from u.

Actually, a normalization factor c is needed so that the total sum value of pagaranks is fixed.

Example of PageRank:



A more precise definition:



11.2.1 Formalization: Matrices and vectors

We can collect the PageRank values of all pages u_i in a (row) vector r.

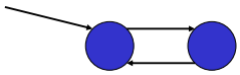
$$\mathbf{r} = \boxed{\mathbf{r}(u_0) \mid \mathbf{r}(u_1) \mid \ldots \mid \mathbf{r}(u_n)}$$

We build r iteratively, all together. Let start from an initial r_0: how to go to r_1, r_2 etc? We can do so by building a matrix P such that $r_{i+1} = r_i * P$. We repeat until a termination condition is reached (next value is equal or under a threshold):

$$(\mathbf{r}_{i+1} = \mathbf{r}_i \text{ Or } |\mathbf{r}_{i+1} - \mathbf{r}_i| < \varepsilon)$$

The are some problems:

- Initial values?
- Convergence? The problem of "rank sink":



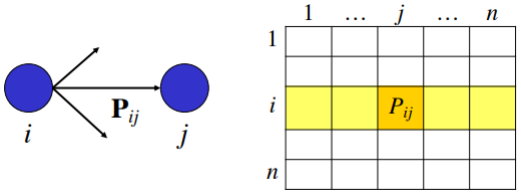
Before answering, let's see a formalization tool: random walks.

Random walks: Let's imagine a browser/user that walks randomly among web pages. It starts from a random page and at each step, it selects a random link and follows it (all links have the same probability to be the choosen one).

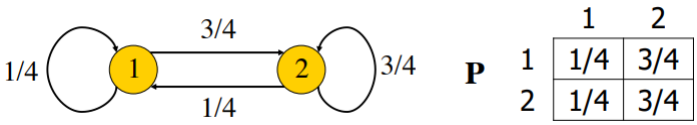
- Which is the probability that the user/walk is on a given page?
- Limit $t \to \infty$ (if the limit exists, there is a stable state)
- PageRank of a page = probability that during a random walk the user is on that page.
- The formal tool for doing so is/are the Markov chains.

Markov Chains

We have n states $\{1, 2, 3, \ldots, n\}$. P is a n x n matrix with transitions probabilities. At a given time-step (discrete time) the Markov chain is in one of the n states. For every current state i the Markov chain P tells us the probability that the next state will be j for every i.

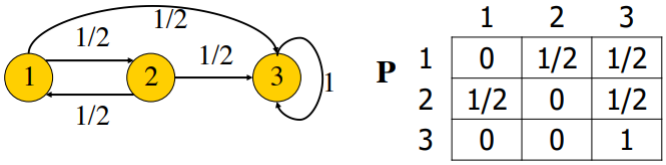


An example:



Intuition: the random walk will stay more in state 2 than in state 1. Sum of each row in P must be equal to 1 (has to go to some state, or maybe stay in the current one).

Another example:



Intuition: more in 3; once there, not way to get out.

Current position and Probability vector

We can represent the current position in a state i by a (row) vector $x = (x_1; \ldots; x_n)$ where:

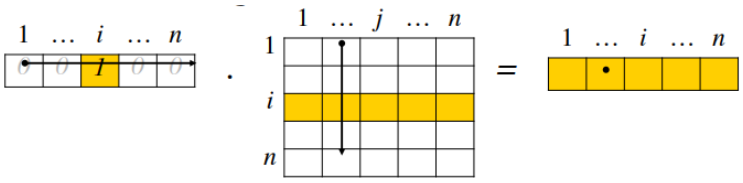
- all components have a 0 value but the i-th, having a 1 value and representing our position in state i.

Generalizing, the sum of the vector value is 1. The Markov chain is in state i with probability x_i.

A step of the walk

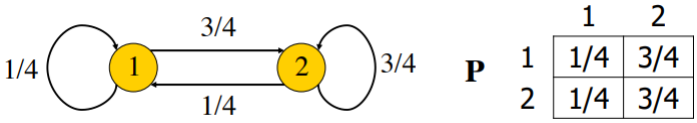
If the probability vector is $x = (x_1; \ldots; x_n)$ what will it be at next step?

- If it is like (0,0,0,0,1,0,0,0) (i.e. the chain is in state i with absolute certainty).
- Looking at the transition matrix P, i-th row, we have the probabilities to proceed in any state when starting from i.



The generalization to values different from 0 and 1 is immediate, it's just a weighted sum/proportion.

Example:



P

	1	2
1	1/4	3/4
2	1/4	3/4

$\mathbf{x}_0 = (1, 0) \rightarrow \mathbf{x}_1 = (0.25, 0.75) \rightarrow \mathbf{x}_2 = (0.25, 0.75)$
$\mathbf{x}_0 = (0, 1) \rightarrow \mathbf{x}_1 = (0.25, 0.75) \rightarrow \mathbf{x}_2 = (0.25, 0.75)$
$\mathbf{x}_0 = (0.6, 0.6) \rightarrow \mathbf{x}_1 = (0.25, 0.75) \rightarrow \mathbf{x}_2 = (0.25, 0.75)$

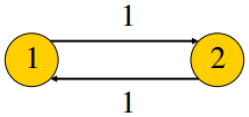
Here we have a quick convergence. This is not always the case...

Eigenvector
Therefore, step of the walk: $x_{i+1} = x_i * P$. Anyway, we're looking for the stable distribution, $x_{i+1} = x_i$. That is, we are looking for x such that $x = x * P$. x is the left eigenvector of P. Does it always exist?

Stable distribution: Any ergodic Markov chain has a steady distribution:

- Any state has a long term visit rate. The rate converges, it is stable.
- On the long term (t tends to infinity), any state is visisted proportionally to its visit rate (and it is independent from the starting point of the random walk).

There exists a path from any state to any state (connected): Ergodic markov chain. It is possibile to be in any state at any step with probability greater than 0. An example of chain connect but not ergodic:



11.2.2 Summary

We can model a random walk on the Web as a Markov chain:

- Transition matrix obtained from links
- Probability vector represents the position

If it is ergodic, the Probability vector has a stable limit: we can use that limit as PageRank. But: is a random walk on the Web ergodic? Of course it isn't: the web is not connected. In order to fix that, we introduce a trick called "teleporting".

11.2.3 Teleporting

During our walk on the web, we click on a lot of links and sometimes we get bored. We then jump (teleport) to a random page.

- p(follow a link) + p(teleport) = 1;
- p(teleport) = constant = 0.1 (for example)

Therefore, random walk with teleporting is a Markov chain; ergodic, thus it has a steady state distribution, each state has its own probability. Any page has its own probability that during the random walk, the user is there. PageRank of the pages is the probability distribution.

Click here for a Youtube recap video of PageRank.

11.3 HITS

The idea behind HITS is to build two related sets of pages:

- **Hub pages (H) - good out-links**
  - Pages containing good links to other pages: yahoo, good bookmarks, etc.
  - ("Hub", for airports: airports from where you can fly to many places).
- **Authority pages (A) - good in-links**
  - Pages of well recognized people/entities on the Web: pages with high "trust";
  - Pages that are linked a lot from other good/trusted pages.
- Good hubs link good authorities; good authorities are linked by good hubs.

The algorithm doesn't work on the whole web collection and it is a query time algorithm. The set of retrieved pages, named **Root Set (RS)** is the set HITS starts with.

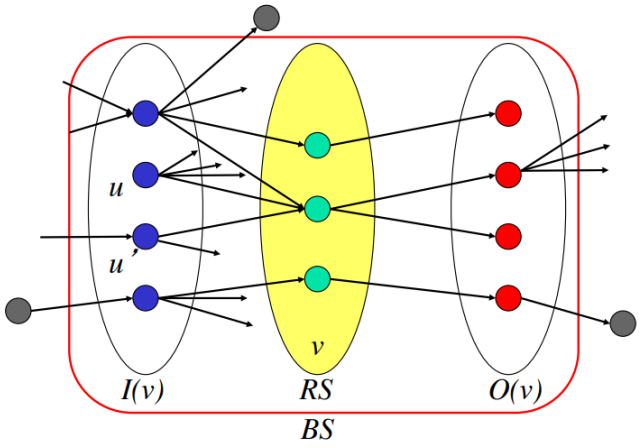
We start from RS and we expand it with:

- $I(v)$: in-pages (all the pages u such that $\exists$ v in RS t.c. $u \rightarrow v$)
- $O(v)$: out-pages (all the pages u such that $\exists v$ in RS t.c. $v \rightarrow u$)

By doing so, we build the so called **Base Set (BS)**:

- Good Hub/Authorities candidate pages.
- The algorithm works on the BS.

H and A are selected from BS by computing, for each page x in BS a hubness value $h(x)$ and an authority value $a(x)$.



$$\mathbf{h}(x) = \sum_{x \to y} \mathbf{a}(y) \qquad \mathbf{a}(x) = \sum_{y \to x} \mathbf{h}(y)$$

- The H value of x depends on how x links high A pages
- The A value of x depends on how x is linked from high H pages

11.3.1 From RS to BS

Why the expansion set (RS to BS)? To increase recall.

- Authoritative pages about the query, that do not contain query terms;
- Hub pages that "just mention" a lot of pages that are about the query.

Locality principle on the web: Linked pages $\rightarrow$ similar topics. Not true anymore for chains longer than 1 link; topic generalization and topic drift occour.

11.3.2 Algorithm

- Begin by assigning initial $h(x) = 1$ and $a(x) = 1$ to any page x in the BS.
- Then, iteratively update by the previous formulas $h(x)$ and $a(x)$.
- Stop when converges (when $h(x)$ and $a(x)$ do not vary anymore).
- At the end, "top hub" pages and "top authority" pages are obtained.

HITS works on the retrieved document set; to keep compelixity low and to be reasonably sure that the set contains good H/A. The iterative process does not depend from the query. (BS does, but once BS is fixed, the query can be forgotten; nor from the language (content independent)).

The implementation it's very similar to the matrix implementation for PageRank. We have two vectors: h and a for hubs and authorities. At the starts every page has a and h equal to 1; we iterate until convergence.

11.4 Comparison and Remarks

- **PageRank:**
  - **Pros:** not spammable, quality index for the whole web, used in Google (and in the majority of the SEs).
  - **Contra:** not query specific, ok for large graphs, complex computation.
- **HITS:**
  - **Pros:** query specific, works on small graphs.
  - **Cons:** easily spammable (on the hubs), complex computation (but on small graphs), convergence?

11.5 Spam and Anti-spam techniques

Let's now see some spam techniques which have been utilised during the 1st generation of SEs.

- **Term spam**: to add multiple (visible or not visible - graphically hided on the page) occurences of a term in order to increase the page rank (meta-keyword, color of the text = color of the background, CSS tricks, anchor text, etc.).
- **Cloaking**: to return fake pages to SE crawlers; SE index with fake keywords.
- **Doorway pages**: pages optimized for a term or few terms that redirect to the "true" page, usually a commercial one.

Techniques for 2nd generation of SEs:

Link Spam:

- Mutual linking;
- Hidden links;
- Links with cheating text;
- Domain flooding: many domains linking/redirecting to a page.

Robots:

- Fake clicks;
- Fake queries;
- Automatic add-URLs (to SEs).

As for anti-spam techniques, the main approach is to build a more robust link analysis "procedure". For example we should ignore statistically unsound connectivity (or text); exploit link analysis to find spammers.

One can even use Machine Learning techniques for detecting spam. Another approach is to add Anti-Robot tests.

11.6 SEO - Search Engine Optimization

With SEO we talk about activities aimed to improve the rank of a website ("to be ranked first"). Legit techniques (according to SEs) are to create good quality websites; not legit ones are: link farm, keyword stuffing, etc.

Tasks of a SE optimizer:

- User oriented optimization;
- SE oriented optimization;
- Customer reporting and Maintenence.

With User Oriented Optimization we mean to aim at retaining the user; techniques can be:

- To build a website with a good structure, navigability, content, usability. In general to follow HCI guidelines and to create "a good website for the user".

For SE oriented optimization we should aim at:

- 1st gen: density (quantity of keywords); proximity (of the keywords); titles, bold, meta-data; well structure and emphasized texts.
- 2nd gen: popularity (mutual link, larger and larger link "islands"; focus on LAR).
- to be careful to exaggerated optimization: the website can be detected as spam.

The bottom line is to build a **good quality website** by trying to create high quality even without being biased of knowing exactly how quality will be measured (by SE algorithms).

11.7 Dups and Mirrors

Definitions:

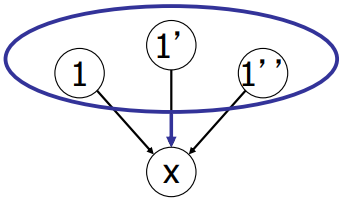
- **Duplicated pages (dups)** are those pages that are (almost or completely) equal.
- **Mirrors** are those pages that mirrors systematically the whole content of another site.

Dups exists (even for legitimate reasons) because of local copies, updates, spam, crawler traps, dynamic urls (a calendar?) spider errors etc. A SE wants to find dups because of two main reasons:

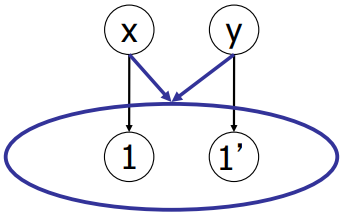
- Resource saving: RAM, disk, network (during crawling);
- User satisfaction: lower with duplicated pages in retrieved documents.

The main reasons to have mirrors (also here they can even be legitimate ones) are: local copies, load balancing, location specific data, updates, spam, crawler traps, different names for same IP, etc. Why should a SE want to find mirrors?

- Smarter Crawling: fetch from most fast/up-to-date host, avoid double crawls;
- Provide reundancy in the results list (if this site is down, try this mirror);
- Better link analysis: in-link combination (to dup pages) and out-link combination (from dup pages).



- x has an authority value higher than its "true" value (3 inlinking pages instead of 1)



- 1 and 1' have less authority than their "true" value (they are linked from 2 pages, not from 1)

12 Crawling

We will now see a little bit into details the crawling phase. As we already know, the crawler is a program module that fetches automatically web pages to create a local index (to feed the indexer module) and to add them to a local collection. A crawler is fundamental for a SE in order to have a good quality collection which is up-to-date and large.

A crawler begins with a set of URL seeds. It inserts the URL seeds in a URL queue and fetches the pages in the URL queue, in some order. It extracts the URLs from the fetched pages and adds them to the queue. Loop that.

There are 2 kinds of crawlers:

- **Batch Crawler (periodical)**: it fetches pages until a certain criterion is reached; then stops. If restarted, it restarts from scratch, building a new collection.
- **Incremental Crawler (continuous)**: even if the criterion is reached (collection is complete), it goes on (re)fetching pages, substituting the old ones, replacing the unimportant ones, updating etc. This is the most used version.

There are some factors to be considered while crawling:

- How to crawl? the ideal order is to crawl firstly the best pages and to avoid dups.
- How much to crawl?
- How often to crawl? too frequently will mean no updates to do.

The overall effectivness of crawling depends on the variation rate of the web, with a static web, periodic is equal to incremental. With a very dynamic web, incremental is potentially more effective (if it is known which pages vary and when). It's very difficult to measure crawl effectiveness.

Crawling factors can be divided into five groups:

1. Page factors
2. Crawler load factors
3. SE factors
4. User factors
5. Server load factors

(1) - Web page factors
To crawl changed pages first. Least recently changed first and first changed first crawled. If page A has changed earlier than B than A should be checked first.

We should check most frequently page first. We should also consider how much a page changes when it changes. We should update most heavily changing page first.

(2) - Crawler load factors
We should consider the age in the index ("oldest crawled first"), page size ("small pages first"). Some pages are on slow servers, so it's better to update first pages on faster servers.

(3) - SE factors
We should update most retrieved type of pages first. We should also update highest pagerank pages first.

(4) - User factors
We should update pages that are most frequently clicked by users first; then the others.

(5) - Server load factors
We should avoid to overload web servers when crawling on them. We should avoid to crawl too often from the same web server.

Some factors are independents, not comparable; or at least it's not easy. Some of them depends on others, combination it's not easy, we have to choose what's most important.