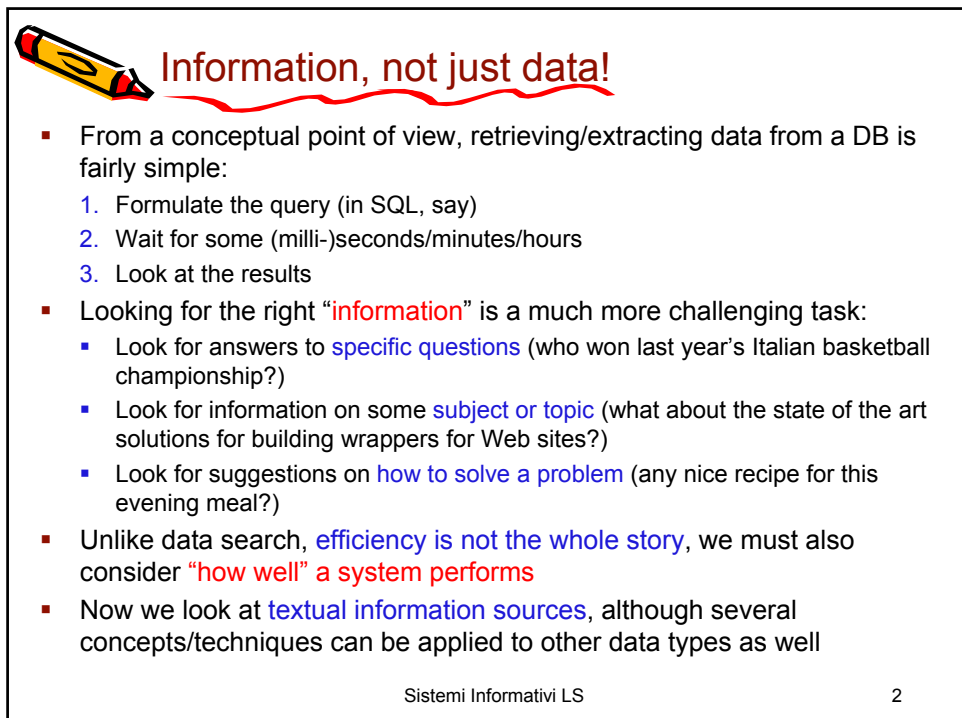


Searching Documents and Pages

Prof. Paolo Ciaccia
<http://www-db.deis.unibo.it/courses/SI-LS/>

05_SearchingDocs&Pages.pdf

Sistemi Informativi LS



Information, not just data!

- From a conceptual point of view, retrieving/extracting data from a DB is fairly simple:
 1. Formulate the query (in SQL, say)
 2. Wait for some (milli-)seconds/minutes/hours
 3. Look at the results
- Looking for the right “**information**” is a much more challenging task:
 - Look for answers to **specific questions** (who won last year’s Italian basketball championship?)
 - Look for information on some **subject or topic** (what about the state of the art solutions for building wrappers for Web sites?)
 - Look for suggestions on **how to solve a problem** (any nice recipe for this evening meal?)
- Unlike data search, **efficiency is not the whole story**, we must also consider “**how well**” a system performs
- Now we look at **textual information sources**, although several concepts/techniques can be applied to other data types as well

Sistemi Informativi LS

2



Information Retrieval (IR) systems

- The main task of an IR system is:
 - Given a query, which represents the “information needs” of the user, and a collection of documents
 - Retrieve the documents in the collection that are “relevant” to the query, returning them to the user in decreasing order of relevance
- (Some) issues:
 - How are documents represented?
 - How are queries expressed?
 - How does the system evaluate the relevance of documents? (this is the so-called “Retrieval Model” of an IR system)
 - How to implement the retrieval model efficiently?
- It has to be understood that the notion of relevance is a subjective one
 - I.e., two users might differ in evaluating a document as relevant/interesting or not



Document and query representation

- Documents are usually represented as bags (i.e., multi-sets) of “index terms”
- An index term can be:
 - a keyword, chosen from a group of selected words
 - This approach is particularly useful to classify documents, although it requires a manual intervention
 - any word, also known as full-text indexing
- Complex index terms may also be defined, such as groups of nouns (e.g., computer science)
- Alternatively, the composing terms are treated separately and the group is reconstructed by looking at the positions of the words in the text
- Queries follow a similar approach
- However, how query terms are combined is an issue...



1st step: Boolean queries

- The simplest retrieval model is based on Boolean algebra:

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

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

1 if play contains term,
0 otherwise

Sistemi Informativi LS

5



Computing the results

- For each term we have a binary vector, with size
N = number of documents in the collection
- Bit-wise Boolean operations are enough to compute the result:

Brutus = (110100), Caesar = (110111), Calpurnia = (010000)

(110100) AND (110111) AND NOT (010000) = **100100**

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

Result = 1 0 0 1 0 0

Sistemi Informativi LS

6



Is the matrix solution a good idea?

- Assume we have a collection of $N = 1M$ documents
- Also assume that the overall **number of distinct terms** is $V = 100K$, with each document containing, on the average, **1000** distinct terms
- The matrix consists of $100K \times 1M = 10^{11} = 100G$ boolean values, with only **1% (1G)** of 1's
- Space overhead suggests to look for a more effective representation
- Further, consider taking bit-wise AND and OR over vectors of 1M bits...
- The commonest solution adopted in text retrieval system is a structure known as "**inverted index**" (also: "**inverted file**")
- There are many variants of the inverted index, aiming to:
 - Support different query types
 - Reducing space overhead
 - ...



Building the inverted index (1)

1) Documents are parsed to extract terms...

doc 1

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

doc 2

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

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

2) Terms are sorted...

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



Building the inverted index (2)

3) Multiple occurrences of a term in the same document are merged and frequency information is added...

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

4) The index is then split into a "dictionary/vocabulary" and a "posting file"

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

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



Inverted index size

- Consider the size of the
 - Dictionary:** with 100K terms, even assuming that a vocabulary entry requires 30 bytes on the average, we need just **3MBytes**
 - Empirical law: $V = kn^b$ where $b \approx 0.5$, $k \approx 30-100$ and n is the total number of terms (tokens) in the documents
 - Posting file:** if each of the 1M documents contains about 1000 distinct terms, we have **1G entries** in the posting file, each of them referenced by a distinct pointer
- A more effective space utilization is obtained by means of **posting lists**:
 - For each distinct term, have just one pointer to a list in the posting file
 - This "posting list" contains the id's of documents for that term and is ordered by **increasing values of documents identifiers**
 - Continuing with the example, this way we save 1G – 100K pointers!
 - Techniques are also available to "compress" the info within each list

Term	N docs	Tot Freq
...
caesar	2	3
...

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



Using the inverted index with Boolean q.'s

- **ANDing** two terms is equivalent to **intersect** their posting lists
- **ORring** two terms is equivalent to **union** their posting lists
- **t1 AND NOT(t2)** is equivalent to look for doc id's that are in the posting list of term t1 but not in that of t2

Term	N docs	Tot Freq
computer	5	23
principles	1	3
science	3	20

Doc #	Freq
3	2
5	5
8	11
10	3
13	2
5	3
2	10
5	2
8	8

q = computer **AND** science **AND** principle

It is convenient to start processing the shortest lists first, so as to minimize the size of intermediate results. We have the Ndocs info in the dictionary!

Union and intersection take linear time, since posting lists are ordered by doc id's!



What to index?

- Most common words, like “the”, “a”, etc., takes a lot of space since they tend to be present in all the documents
- At the same time, they provide little or no information at all
 - However, what about searching for “to be or not to be”?
- A (language-specific) **stopword** list can be used to filter out those words that are not to be indexed
- The “**rule of 30**”:
 - ~30 words account for ~30% of all term occurrences in written text
 - Eliminating 150 commonest terms from indexing will cut almost 25% of space

Remark: in practice, things are more complex, since we may want to deal with:

- Punctuation: State-of-the-art, U.S.A. vs. USA, a.out, etc.
- Numbers: 3/12/9, Mar. 12, 1991, B-52, 100.2.86.144, etc.
- ...



Stemming

- In order to save space and to improve the chance of retrieving a document, a process called “stemming” is usually executed before indexing, so as to **reduce terms to their “roots”**
 - e.g., automate(s), automatic, automation all reduced to automat

for example compressed and compression are both accepted as equivalent to compress.



for exampl compres and compres are both accept as equal to compres.

- It is experimentally shown that **stemming can reduce the number of terms by ~40%, and total index size by ~30%**
- For details on how stemmers (= stemming algorithms) operate:
<http://www.comp.lancs.ac.uk/computing/research/stemming/general/index.htm>



Thesauri

- **Synonyms** can be viewed as equivalent terms
 - E.g., car = automobile
- A text retrieval system usually comes with a **thesaurus** that, in its simplest form, consists of:
 1. A list of (important) terms
 2. For each term, a set of related words
- Related term = **synonyms**, **hypernyms** (car is a kind of...), **hyponyms** (... is a kind of car), etc.
- Actually, a thesaurus constitutes a **semantic network of terms**
- Take a look at
 - **Wordnet** (www.cogsci.princeton.edu/~wn/)
 - **Merriam-Webster thesaurus** (www.m-w.com)



The WordNet interface

Overview for "car"

The noun "car" has 5 senses in WordNet.

1. **car**, auto, automobile, machine, motorcar -- (4-wheeled motor vehicle; usually propelled by an internal combustion engine; "he needs a car to get to work")
2. **car**, railcar, railway car, railroad car -- (a wheeled vehicle adapted to the rails of railroad; "three cars had jumped the rails")
3. cable car, **car** -- (a conveyance for passengers or freight on a cable railway; "they took a cable car to the top of the mountain")
4. **car**, gondola -- (car suspended from an airship and carrying personnel and cargo and power plant)
5. **car**, elevator car -- (where passengers ride up and down; "the car was on the top floor")

Search for of senses

- Show glosses
 Show contextual help

Results for "Hyponyms (...is a kind of this), brief" search of noun "car"

Sense 1
 car, auto, automobile, machine, motorcar -- (4-wheeled motor vehicle; usually propelled by an internal combustion engine)
 => ambulance -- (a vehicle that takes people to and from hospitals)
 => beach wagon, station wagon, wagon, beach wagon, station wagon, wagon -- (a car that has a long body)
 => bus, jalopy, heap -- (a car that is old and unreliable; "the fenders had fallen off that old bus")
 => cab, hack, taxi, taxicab -- (a car driven by a person whose job is to take passengers where they want to go)
 => compact, compact car -- (a small and economical car)
 => convertible -- (a car that has top that can be folded or removed)
 => coupe -- (a car with two doors and front seats and a luggage compartment)
 => cruiser, police cruiser, patrol car, police car, prowl car, squad car --
 (a car in which policemen cruise the streets; equipped with radiotelephonic communications to headquarters)
 => electric, electric automobile, electric car -- (a car that is powered by electricity)
 => gas guzzler -- (a car with relatively low fuel efficiency)
 => hardtop -- (a car that resembles a convertible but has a fixed rigid top)
 => hatchback -- (a car having a hatchback door)

Sistemi Informativi LS

15



Using thesaurus' information

- If your query includes the term "car", there are two possibilities to exploit thesaurus' information:
 1. The system can "expand" the query
 E.g., car AND tyres becomes (car OR automobile) AND tyres
 2. The system can build the inverted index by also placing docs containing the term "car" in the posting list of "automobile", and vice versa
- Usually query expansion is preferred, in order to avoid excessive index growth
- However, query expansion slows down query processing...

Sistemi Informativi LS

16



Phrase and proximity queries

- Searching for **phrases** (e.g., “the challenge of information retrieval”) can be implemented by extending the entries in the posting file with **positional information** (= position of the term in the document)

Term	N docs	Tot Freq
computer	5	23
principles	1	3
science	3	20

Doc #	Freq
3	2: 13, 26
5	5: 20, 27 85, 112, 215
8	11: ...
10	3: ...
13	2: ...
5	3: 21, 35, 45
2	10: ...
5	2: 28 101
8	8: 35, 51, ...

q = “computer science”

- Proximity queries** are a relaxed version of phrase queries, where we just require that the query terms are “close” each other
E.g., Gates NEAR Microsoft



Limits of the Boolean retrieval model

- Although the Boolean model has a clear semantics and is also appreciated by experienced users, it has several **drawbacks**:
 - Unexperienced users** have difficulties in understanding what Boolean operators really mean
 - AND sometimes means disjunction**: I want to know about “cats AND dogs”
 - OR sometimes implies exclusion (XOR)**: docs about “IBM OR Oracle”
 - NOT (and De Morgan’s laws as well!) is not easy to understand**
 - No notion of “**partial matching**”
 - No **ranking** of the documents
 - “**Near-miss**” and “**Information overload**” problems
- Several extensions of the basic Boolean model have been proposed, in order to solve above problems
- We skip them, and directly move to consider the “**vector space model**”...



Preliminaries

- The Boolean model just looks at binary (1/0) information:
presence vs. absence of a term in a document
- Thus, **for a conjunctive query all query terms must be present for a document to be considered relevant**
- Let's take a different perspective: given a set of search terms, which translate our information need, we could argue that:
 1. The more such terms a doc contains, the better such doc is
 2. For a given search term, the more occurrences of such term in a doc, the better the doc is
- From 1. we derive that **not all terms need to be present for a document to be relevant**
- From 2. we derive the need to take into account **term frequency information**



Weighting terms: tf.idf

- Given a **term** t_i and a **document** doc_j , we can provide a measure of how well doc_j "fits" t_i , that is, which is the relevance of doc_j w.r.t. t_i
- Let $w_{i,j}$ denote the "**weight**" of t_i in doc_j
 - For the Boolean model it is $w_{i,j} \in \{0,1\}$
- In the **tf.idf weighting schema**, $w_{i,j}$ depends on two factors:
 - The first is **the number of occurrences of t_i in doc_j** , $freq_{i,j}$, and is called the **term frequency** of t_i in doc_j , also denoted **tf_{i,j}**
 - One could also take "normalized" frequency values, i.e., **tf_{i,j} = freq_{i,j} / max_i{freq_{i,j}}**
 - The second is a factor that is related to the "discriminating power" of t_i in the collection, and is **negatively correlated with the number of docs, N_i , in which t_i appears** (this is Ndocs in the inverted file figures).
This is called the **inverse document frequency** of t_i , denoted **idf_i**, and can be computed as **idf_i = log(N / N_i)**, where N is the number of docs in the collection
 - A term present in each document has **idf_i = 0**,
- The **tf.idf** scheme computes $w_{i,j}$ as **$w_{i,j} = tf_{i,j} * idf_i$**
- We still have $w_{i,j} = 0$ if t_i is not present in doc_j



Documents as vectors: the VSM

- If we view each term as independent of (orthogonal to) the others, our docs are vectors in the \mathbb{R}^V space, with values along the i-th coordinate given by $w_{i,j}$
- This is called the **Vector Space Model (VSM)**, for which a measure of similarity among vectors can be easily defined...

A vector in \mathbb{R}^7

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.1
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

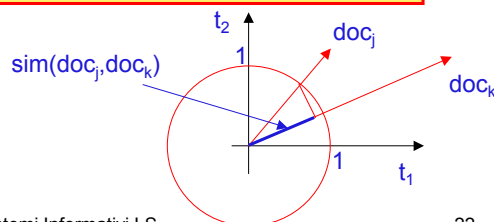


Cosine similarity

- The VSM model proposes to evaluate the similarity of two documents by measuring the **correlation** of the corresponding vectors
- A common way to measure correlation is computing the **cosine of the angle between the two vectors**:

$$\text{sim}(\text{doc}_j, \text{doc}_k) = \cos(\Theta_{j,k}) = \frac{\text{doc}_j \bullet \text{doc}_k}{\|\text{doc}_j\| \times \|\text{doc}_k\|} = \frac{\sum_{i=1}^V w_{i,j} \times w_{i,k}}{\sqrt{\sum_{i=1}^V w_{i,j}^2} \times \sqrt{\sum_{i=1}^V w_{i,k}^2}}$$

- Normalization is to avoid that documents' length becomes the dominant factor





Ranking the documents (1)

- In order to rank the documents it is sufficient to compute their similarities w.r.t the **query vector q**
 - Thus, the query is a vector as well! (we don't have Boolean operators anymore)
- If query terms are not weighted (alternatively, one could take into account their idf values) we have $q = (w_{1,q}, \dots, w_{v,q})$, $w_{i,q} \in \{0, 1\}$
- Let **QT** be the **index set of the query terms**, that is: $QT = \{i: w_{i,q} = 1\}$
- Then we can write

$$\text{sim}(\text{doc}_j, q) = \frac{\sum_{i \in QT} w_{i,j}}{\sqrt{\sum_{i=1}^v w_{i,j}^2} \times \sqrt{|QT|}} \propto \frac{\sum_{i \in QT} w_{i,j}}{\sqrt{\sum_{i=1}^v w_{i,j}^2}}$$

- Thus, to compute the similarity of a document w.r.t. to a query, we just need
 - 1) to accumulate its weights over the query terms, and
 - 2) to normalize by the length of the document vector



Ranking the documents (2)

- If documents' vectors have been normalized, i.e., $\|\text{doc}_j\| = \sqrt{\sum_{i=1}^v w_{i,j}^2} = 1$

then we are left with: $\text{sim}(\text{doc}_j, q) \propto \sum_{i \in QT} w_{i,j}$

- Note that, for normalized vectors, cosine similarity gives the same ranking as Euclidean distance:

$$\begin{aligned} L_2(\text{doc}_j, q) &= \sqrt{\sum_{i=1}^v (w_{i,j} - w_{i,q})^2} = \sqrt{\sum_{i=1}^v w_{i,j}^2 + w_{i,q}^2 - 2 \times w_{i,j} \times w_{i,q}} \\ &= \sqrt{1 + |QT| - 2 \times \sum_{i \in QT} w_{i,j}} \end{aligned}$$

- Ok, now that we know how to compute similarities, we can rank documents and just return only the k best documents
- How?



Computing the k best documents

- We do not present all the details, even because each system has its own undisclosed “tricks”
- The idea is:
 1. Take the posting lists of the query terms (remind, here we have frequency of terms, and in the vocabulary the inverse document frequencies)
 2. Merge (take the union of) such lists
 3. Sort by decreasing similarity values
 4. Return the first k documents
- The problem is that we have to completely process all the lists...
- ...however, notice that **cosine similarity is an aggregation (scoring) function of |QT| scores**
 - We can use something like TA algorithm!
 - In practice, systems often resort to the computation of an approximate result (e.g., just take the best k docs from each list)



VSM: an example

- Assume normalized vectors

q = computer science

2. Merge the lists

Term	N docs
computer	5
science	3

Doc #	w _{i,j}
3	0.17
5	0.2
8	0.25
10	0.03
13	0.08
2	0.31
5	0.12
8	0.05

Doc #	sim
2	0.31
3	0.17
5	0.32
8	0.3
10	0.03
13	0.08

Doc #	sim
5	0.32
2	0.31
8	0.3
3	0.17
13	0.08
10	0.03

1. Take the posting lists of the query terms

3. Sort by decreasing similarity values



Retrieval effectiveness

- How can we evaluate the “quality of results” of a system?
- There are two “standard” measures:

Precision (Prec): this is the fraction of **retrieved docs that are relevant**

- In probabilistic terms, it is $Prec = Prob\{doc \text{ is relevant} | doc \text{ is retrieved}\}$

Recall (Rec): this is the fraction of **relevant docs that are retrieved**

- In probabilistic terms, it is $Rec = Prob\{doc \text{ is retrieved} | doc \text{ is relevant}\}$

- For a given query, let

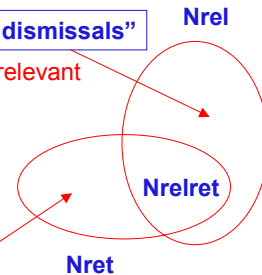
- N_{rel} be the number of **relevant** docs
- N_{ret} the number of **retrieved** docs, and
- N_{relret} the number of **retrieved** docs that are also **relevant**

- It is:

$$Prec = \frac{N_{relret}}{N_{ret}} \quad Rec = \frac{N_{relret}}{N_{rel}}$$

“false drops/hits/alarms”

“false dismissals”



How to measure?

- To measure the performance of a system one resorts to so-called “**test collections**”, which come with a set of queries, for each of which the set of relevant documents is known
 - In practice, relevance is assessed by a set of experts
- The reference test collections for information retrieval systems are the ones from **TREC**:
 - TREC stands for “Text Retrieval Conference”: it is an annual conference dedicated to experiments over large text collections ($N > 1$ million, several GBytes of text)
- The collections come from different sources (e.g.: Wall Street Journal, Federal Register, US Patents, LA Times)
- As a final remark:
 - if you don’t have a test collection, you can still evaluate precision of results, but you cannot evaluate recall!**

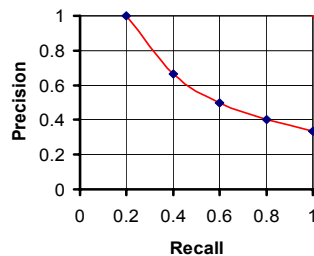


Precision-recall curve

- For a given query, it is customary to **measure the precision at several levels of recall** and then to plot (Prec,Rec) values, thus leading to a so-called **Precision-Recall curve**

Doc#	20	37	2	19	26	87	11	5	4	54	12	36	81	42	27
Relevant?	✓		✓			✓				✓					✓

Assume there are 5 relevant documents for our query



(1,1) is the "ideal" system

- The retrieval performance of a system can then be evaluated by **averaging precision values over a set of sample queries**



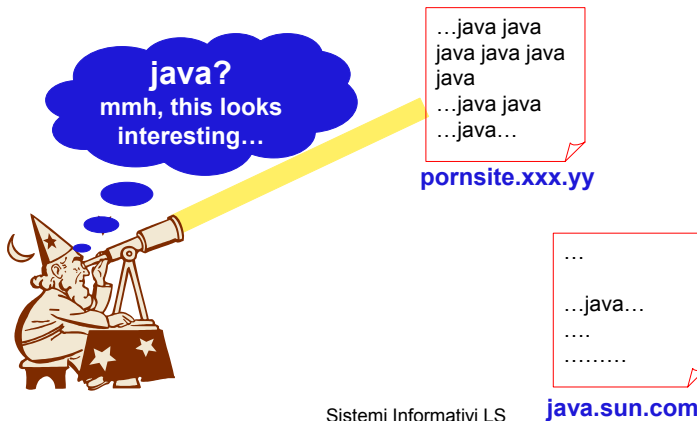
Some considerations

- The **vector space model** is undoubtedly the **most widely used** text retrieval model
- It is commonly used by TREC participants, as well as by **most Web search engines**
 - However this is not the full story about Web search engines...
- Its basic concepts are also adopted, possibly with some variations, by systems other than IR ones, such as:
 - Information Filtering** systems
 - Recommendation** systems
 - Multimedia** systems
- With respect to the Boolean model, **VSM usually achieves a better precision**, at any given recall level, for simple queries (a few search terms)
- The **Boolean model**, on the other hand, is **still preferred by experienced users** who know how to precisely formulate their queries
- Other models (e.g., based on probability) exist, but they are not as popular as VSM



Web search engines

- Early search engines on the Web adopted the VSM or some variant of it
- However, **since VSM relies on term frequencies, it is prone to spamming**
Example: if you want your site to become top-ranked for the query “java”, just insert in the home page site the word “java” n times!



Sistemi Informativi LS

java.sun.com

31



What's new about searching the Web?

Web pages, unlike “ordinary text documents”, are:

- widely distributed on many servers ⇒ need to **gather** them
- extremely dynamic/volatile ⇒ need to **refresh** their content
- highly structured (HTML tags) ⇒ need to **look at terms' positions**
- extensively inter-linked ⇒ need to **analyze links**
- ...

Web users are:

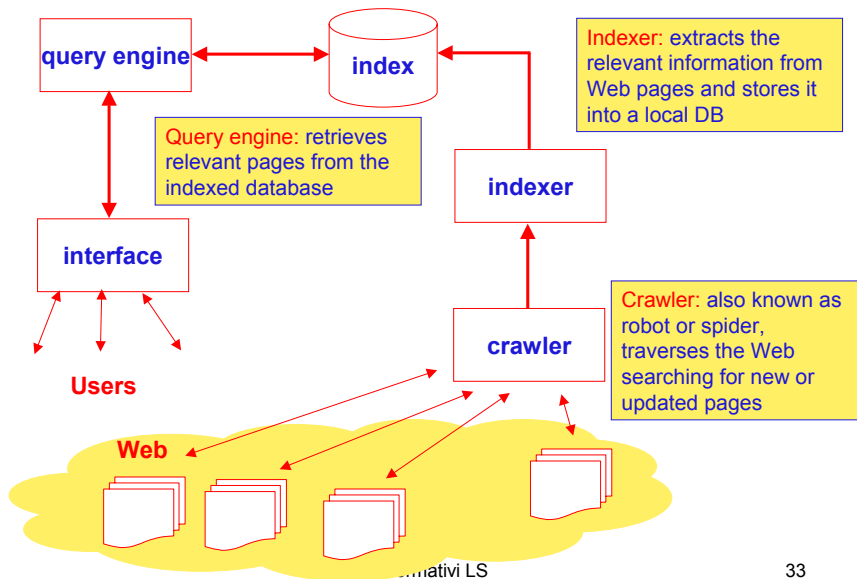
- ordinary people, without any special training on querying
- many

Sistemi Informativi LS

32



The components of a Web search engine

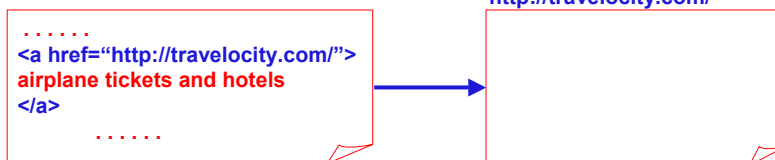


33



Using tag information (sketch)

- It is possible to exploit HTML tags to improve retrieval effectiveness
- The two basic ideas are:
 - Associate different importance to term occurrences in different tags
 - E.g., `<title>java...</title>` more relevant than `<p>java</p>`
 - Used by Altavista, HotBot, Yahoo, Lycos
 - Look at anchor text to index referenced documents
 - Used by Google



- Problems:
 - relative importance of tags not clear
 - lacks rigorous performance study



Link analysis: basic idea

- Starting from 1998 [BP98,Kle99], several techniques have been proposed to exploit the **link structure of the Web**, so as to improve the precision of results of search engines
- The basic intuitions can be summarized as follows:

1. If the owner of a page P1 creates a link to a page P2, then this is an evidence that P1 is assigning some “authority” to P2

An “authoritative page” (authority, for short) is thus a page with many in-coming links (backlinks)

- On the other hand, this just gives authority (thus, relevance) to popular pages (again, **spamming can become a problem! Do you see why?**)

2. If P1 is an authority and it links to P2, then P2 is likely to be an authority as well

This leads to a recursive definition of authority

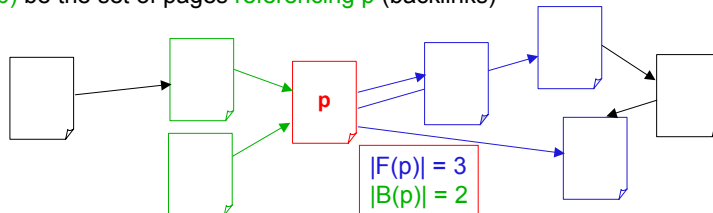


Page Rank [BP98]

- The PageRank method, which is the distinguishing feature of **Google**, assigns, **independently of the specific query q**, to each page p a rank **PR(p)**
- For a given query q, Google evaluates the “overall score” of a page p by means of a **weighted sum of text-based relevance** (based on a VSM-like model) and authority ranking:

$$\text{Score}_q(p) = w_{\text{VSM}} \times \text{sim}(p,q) + w_{\text{AUTH}} \times \text{PR}(p)$$

- To describe Page Rank, we need a couple of definitions; let
 - $F(p)$ be the set of pages **referenced by page p** (forward links)
 - $B(p)$ be the set of pages **referencing p** (backlinks)





Computing the Page Rank: basic version

- The basic version of Page Rank is simply:
from which we see that:

$$PR(p) = \sum_{s \in B(p)} \frac{PR(s)}{|F(s)|}$$

- Each page s “distributes” its page rank among all the pages it references
 - $PR(p)$ is the sum of such contributions from all pages referencing p
- The computation is performed iteratively, by starting from an initial (equal) vectors of ranks

As soon as we update $PR(p)$, we use it

Then, we can normalize page ranks: $\sum_p PR(p) = 1$

page	PR(p)	F(p)
p1	1	2
p2	1	1
p3	1	1

$PR(p1) = 1/1 = 1$
 $PR(p2) = 1/2 = 0.5$
 $PR(p3) = 1/2 + 0.5 = 1$

page	PR(p)	F(p)
p1	1	2
p2	0.5	1
p3	1	1

$PR(p1) = 1/1 = 1$
 $PR(p2) = 1/2 = 0.5$
 $PR(p3) = 1/2 + 0.5 = 1$

page	PR(p)
p1	0.4
p2	0.2
p3	0.4

Sistemi Informativi LS 37



The rank sink problem

- The procedure may fail to converge in case of “sink pages”, i.e., pages with no out-going links that “absorb” and do not “redistribute” ranks

p2 is a sink page

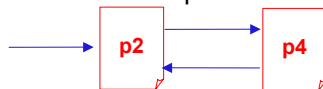
page	PR(p)	F(p)
p1	1	1
p2	1	0
p3	1	2

$PR(p1) = 1/2 = 0.5$
 $PR(p2) = 1/2 = 0.5$
 $PR(p3) = 0.5$

page	PR(p)	F(p)
p1	0.5	1
p2	0.5	1
p3	0.5	1

$PR(p1) = 0.5/2 = 0.25$
 $PR(p2) = 0.5/2 = 0.25$
 $PR(p3) = 0.25$

- The problem also arises in the presence of *cyclic dead-ends*



- To obviate the rank-sink problem, the actual Page Rank algorithm considers the so-called “random surfer” model...



The random surfer model

- We can view the basic Page Rank equation as a **model of user behavior**:
 - If, at a certain time, a user is visiting a page s , the probability that it will move to a page p among the ones in $F(s)$ is $1/|F(s)|$
 - Thus, $PR(p)$ is a sort of sum of probabilities (thus, a probability as well)...
 - Indeed, it can be proved that

the (normalized) page rank of a page p is the stationary probability of being in p if one takes a “random walk” over the Web

- This can be formally analyzed using Markov chains
- ...however, if one enters a sink page the model breaks down!
- The problem can be solved if one considers that, besides following links, a user might occasionally “jump”, with probability ε , to another, randomly chosen, page
- This leads to re-define the Page Rank equation as:

$$PR(p) = \varepsilon + (1 - \varepsilon) \times \sum_{s \in B(p)} \frac{PR(s)}{|F(s)|} \quad \varepsilon \approx 0.15$$



A different approach

- Page Rank assumes that the importance of a page is independent of the specific query q , thus it needs to be combined with text-based relevance
- A different approach has been pioneered by J. Kleinberg [Kle99]
- The basic idea now is:

1. If we have a query q , we can use text-based relevance to isolate a set of “good pages”, and then to apply link analysis to this set, so as to understand which of them are the most authoritative
Such “good pages” constitute the so-called “root-set” of query q

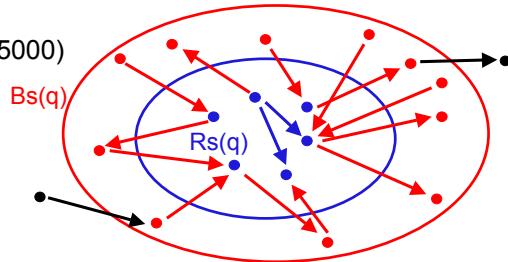
- On the other hand, by limiting link analysis only to the root-set, we might miss some good authorities that do not contain the search keyword(s)

2. Link analysis is performed on a so-called “base-set”, which is obtained by adding to the root-set some pages that are connected to those in the root-set



Root-set and base-set

- The root-set of a query q , $Rs(q)$, is determined by taking the best k , say $k = 200$, pages according to a classical text-based ranking (e.g., using tf.idf and VSM)
- The base-set, $Bs(q)$, is then defined as follows:
 1. For each page p in the root-set, also called a **root-page**:
 $Bs(q) = Bs(q) \cup F(p)$ (the pages referenced by p)
 2. For each root-page p :
 $Bs(q) = Bs(q) \cup B(p)$ (the pages that reference p)
 If p is referenced by too many pages, add only some of them (up to m)
- Typically, $Bs(q)$ consists of a few thousands pages (1000-5000)

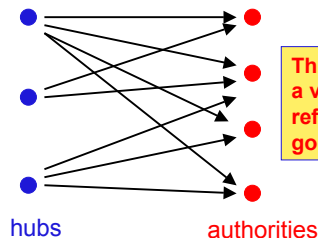


Hubs and authorities

- Link analysis, restricted to the base-set, aims to retrieve the most important authorities in $Bs(q)$
- Unlike Page Rank, authorities are now defined as follows:

- A page is a **good authority** w.r.t. q if it is referenced by many (good hub) pages that are related to the query
- A page is a **good hub page** w.r.t. q if it points to many good authorities for q
- Good authorities and good hubs **reinforce each other**

Think of a hub as a good entry point to valuable resources




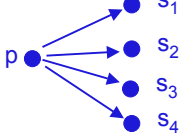
Think of an authority as a valuable resource referenced by many good entry points



The HITS algorithm

- **HITS** (Hyperlink-Induced Topic Search) is the algorithm proposed in [Kle99] to iteratively compute authorities and hubs
- Let $B(p)$ and $F(p)$ be the set of pages in $Bs(q)$ that, respectively, reference and are referenced by p
- The **hub** and **authority score**, $H(p)$ and $A(p)$, respectively, of a page p are defined as:

$$A(p) = \sum_{s \in B(p)} H(s)$$


$$H(p) = \sum_{s \in F(p)} A(s)$$


- After each iteration, $H(p)$ and $A(p)$ values are normalized ($\sum_p H(p)^2 = 1$, $\sum_p A(p)^2 = 1$)
- The process converges after about 20 iterations



What is HITS computing?

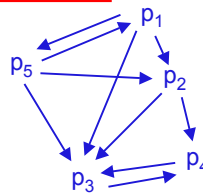
- HITS is an iterative way to compute the principal eigenvector (i.e., the one with the largest eigenvalue) of a matrix M
- Let us define the link matrix L , such that $L(i,j) = 1$ if $p_i \rightarrow p_j$ and 0 otherwise

▪ In vector notation:

$$A = L^T \times H = L^T \times L \times A$$

$$H = L \times A = L \times L^T \times H$$

- A converges to the principal eigenvector of $M_{AUTH} = L^T \times L$
 $M_{AUTH}(i,j)$ is the no. of pages pointing to both p_i and p_j
- H converges to the principal eigenvector of $M_{HUB} = L \times L^T$
 $M_{HUB}(i,j)$ is the no. of pages to which both p_i and p_j point



L	p1	p2	p3	p4	p5
p1		1	1		1
p2			1	1	
p3				1	
p4			1		
p5	1	1	1		

L ^T	p1	p2	p3	p4	p5
p1					1
p2	1				1
p3	1	1		1	1
p4		1	1		
p5	1				



Some considerations about HITS

- HITS individuates the largest eigenvector of the “co-citation matrix” M_{AUTH} , then it can return the k best authorities
- A limitation of HITS is that it does not consider text-relevance scores at all
 - These are just used to define the root-set
- Further, it suffers the so-called problem of the “multiple communities”

A community over the Web is a set of tightly related pages
- In case of multiple communities (e.g., “jaguar”), HITS just returns results from the “primary”/largest one (since its pages are likely to be prevalent in the base-set)
- Indeed, it can be shown that the other eigenvectors of M_{AUTH} correspond to such “secondary” communities
- Needless to say, in the last few years a lot of proposals have appeared to solve above problems...
- [DHH+03] presents a unified framework for analyzing ranking algorithms for Web pages



The “jaguar” query

.370 <http://www2.ecst.cschico.edu/jschlich/Jaguar/jaguar.html>
.347 <http://www-und.ida.liu.se/t94patsa/jserver.html>
.292 <http://tangram.informatik.uni-kl.de:8001/rgehml/jaguar.html>
.287 <http://www.mcc.ac.uk/dlms/Consoles/jaguar.html>

Results of HITS (principal eigenvector); community: Atari Jaguar Production

.255 <http://www.jaguarsnfl.com> Official Jaguars NFL Web site
.137 <http://www.nando.net/nfl/> Jacksonville Jaguars Home Page
.133 <http://www.ao.net/brett/jaguar/> Brett's Jaguar Page
.110 <http://www.usatoday.com/football/> Jacksonville Jaguars

2nd largest eigenvector; community: Jacksonville Jaguars

.227 <http://www.jaguarvehicles.com> Jaguars Cars Global Home Page
.227 <http://www.collection.co.uk/> The Jaguar Collection
.211 <http://www.moran.com/sterling/>
.211 <http://www.coys.co.uk>

3rd largest eigenvector; community: Jaguar Cars