Notes of Information Retrieval course

Marco Natali

INDICE

1 INTRODUCTION TO INFORMATION RETRIEVAL 4
1.1 Boolean retrieval model 8

ELENCO DELLE FIGURE

Figura 1 Altavista search engine web page 5
Figura 2 Google search engine on 1998 6
Figura 3 A Google search with knowledge graph 6
Figura 4 an example of transition from word to concept with the phrase "Leonardo is the scientist that has painted the Mona Lisa" 7
Figura 5 Google search evolution 7
Figura 6 A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise 8
Figura 7 an example of Inverted index for Brutus, Caesar and Calpurnia 8
Figura 8 Pseudocode of Intersection between two postings list 9
Figura 9 And query between Brutus, Calpurnia and Caesar 9

1 INTRODUCTION TO INFORMATION RETRIEVAL

In this course we will strongly consider *search engines*, but *information retrieval* is not only consider on search engines, so a definition of information retrieval is the following

Def. Information retrieval is finding material (usually documents) of unstructured nature that satisfies an information need from within large collections

As defined in this way, information retrieval used to be an activity that only a few people engaged in: reference librarians, paralegals, and similar professional searchers, but now the world has changed, and hundreds of millions of people engage in information retrieval every day when they use a web search engine or search their email.

Information retrieval systems can also be distinguished by the scale at which they operate, and it is useful to distinguish three prominent scales. In web search, the system has to provide search over billions of documents stored on millions of computers. Distinctive issues are needing to gather documents for indexing, being able to build systems that work efficiently at this enormous scale, and handling particular aspects of the web, such as the exploitation of hypertext

In between is the space of enterprise, institutional, and domain-specific search, where retrieval might be provided for collections such as a corporation's internal documents, a database of patents, or research articles on biochemistry. In this case, the documents will typi- cally be stored on centralized file systems and one or a handful of dedicated machines will provide search over the collection

Data available and considered can be from different nature, so we have the following classification:

STRUCTURED DATA: are data that tends to refer to tables and typically allows numerical range and exact match (for text) queries, like for example "Salary < 60000 AND Manager = Smith".

SEMISTRUCTURED DATA (XML/JSON): type of data where are available some structured aspects, like know the name of a document, chapter or paragraph, so it facilitate some semi-structured search like "Title contains data AND Bullets contain search".

UNSTRUCTURED DATA: Typically refers to free text, and allows keyword queries including operators and more sophisticated "concept" queries like find all web pages dealing with drug abuse.

Is the classic model for searching text documents, so we more concentrate on this type of data.

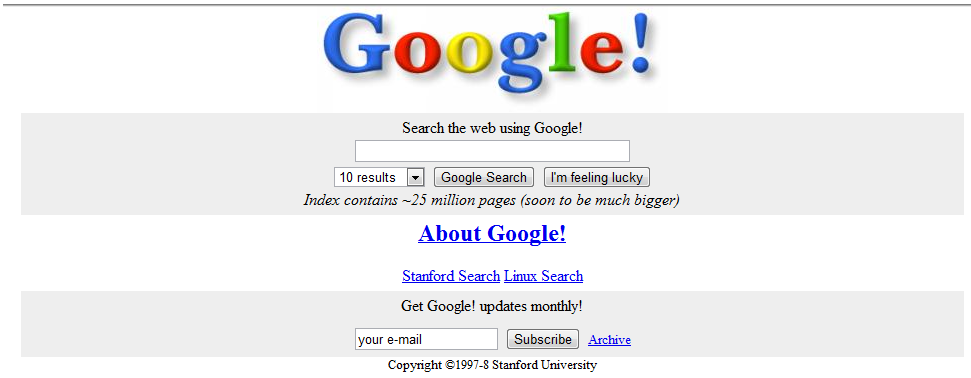
We consider now the search engines, that are defined as

Def. A search engine is a software system that is designed to carry out search, which means to search page in a systematic way for particular information specified in a textual search query.
The search results are generally presented in a line of results, often referred to as search engine results pages and the information may be a mix of links to pages, images, videos, infographics, articles, research papers, and other types of files.

Search engines are not only web search engine but contains also social network (Facebook), streaming site (Netflix), Maps (OpenStreetMap) and work finder (Linkedin) and we have 5 generations of search engines:

Figura 1: Altavista search engine web page

Figura 2: Google search engine on 1998



ZERO GENERATION: introduced on 1991, use only metadata added by users

FIRST GENERATION: introduced on 1995-1997 and use only on-page web-text data, as can be seen on figure 1.

SECOND GENERATION: introduced on 1998 by Google, use off-page, web-graph data, where it use *anchor texts* (how people refer to the page) and *links*, that strongly improve the usability and utility of a search engine. An example of second generation search engine can be viewed on figure 2.

THIRD GENERATION: introduced on 2005, it start to answer "the need behind the query" and are added more sources, like maps, images, news, wikipedia and so on.

FOURTH GENERATION: introduced on 2012, it strongly concern about *knowledge graph* defined as

Def. Knowledge graph is a knowledge base used by Google and its services to enhance its search engine's results with information gathered from a variety

Figura 3: A Google search with knowledge graph

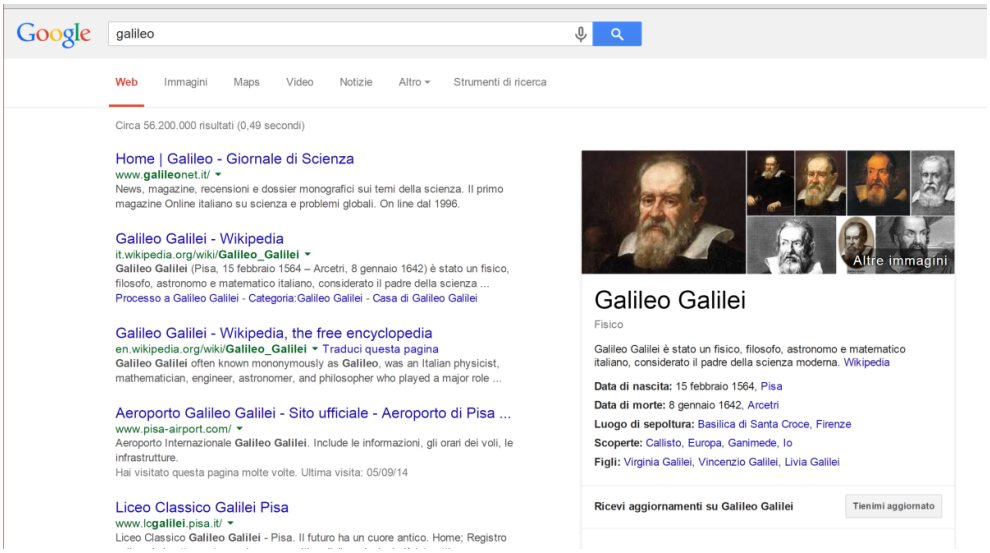


Figura 4: an example of transition from word to concept with the phrase "Leonardo is the scientist that has painted the Mona Lisa"

of sources.
The information is presented to users in an infobox next to the search results.

An example of knowledge graph can be viewed on figure 3 and knowledge graph also permit to transform word to concept, that can provide more information and provide more accurate results on user query, as we can see on figure 4 as we can viewed the phrase "Leonardo is the scientist that has painted the Mona Lisa".
With Knowledge graph is also possible to consider *polysemy* and *sinonimy* word, so at example is possible recognise that "Microsoft's browser" and "Internet explorer" represent the same concept.

In figure 5 is possible to notice which was the evolution on Google search engine.

Now are available "devices 2.0", that have their IDs, communication capacity, computing and storage, like for example car with maps, where the driver can asks (using text or voice audio) the route for a place.

Figura 5: Google search evolution

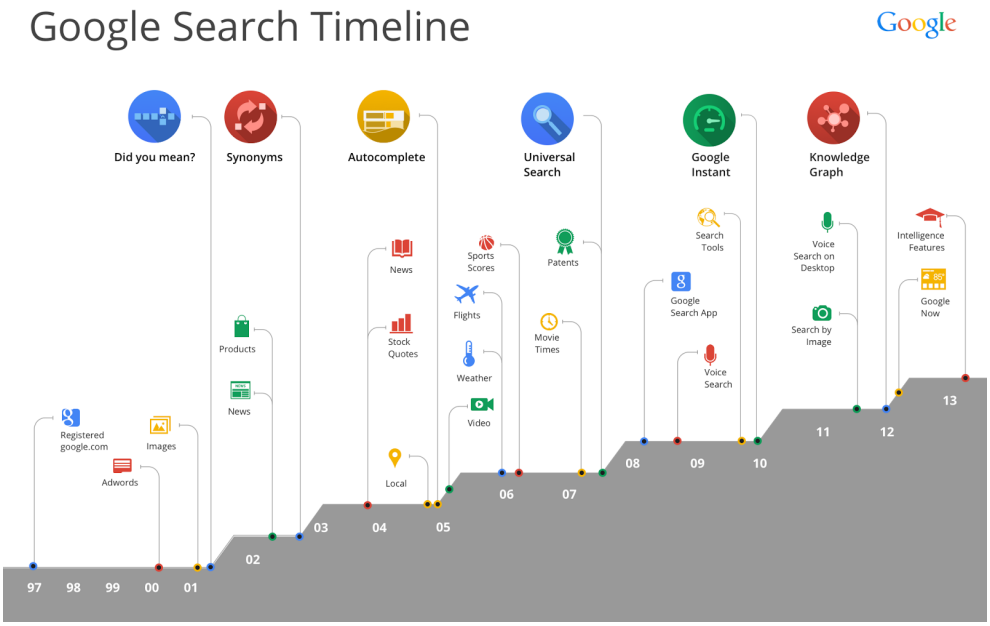


Figura 6: A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise

	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	
...							

1.1 BOOLEAN RETRIEVAL MODEL

We consider now the first model of Information retrieval, that was mainly used on first generation of search engine, but are also used currently in email, library catalog and so on.

Def. The boolean retrieval model is a model able to asks query formed by a boolean expression (query where we use AND, OR, NOT operators to join terms) where we Views each document as a set of words and it is a precise model, because document matches condition or not.

We consider as example to determine which plays of Shakespeare contain the words "Brutus AND Ceaser AND NOT Calpurnia" and the simplest form is do a sort of linear scan, but this type of text processing does not enable to rank results, consider some similarity measures and so on, so we use our boolean retrieval model defined above

In figure 6 is possible to note a term-document incident matrix, where we record for each document, here a play of Shakespeare's, whether it contains each word out of all the words Shakespeare used (Shakespeare used about 32,000 different words).

The problem of Term-document incident matrix is that the matrix could be very big and it is also a *sparse* matrix, so we waste a lot of spaces to represent useless data.

Figura 7: an example of Inverted index for Brutus, Caesar and Calpurnia

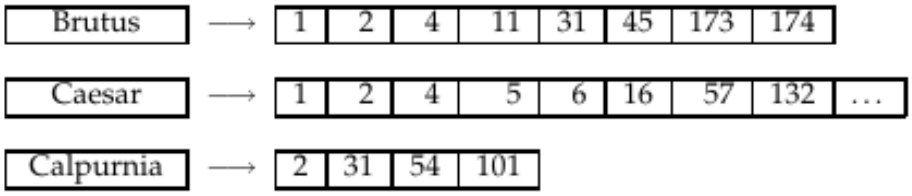


Figura 8: Pseudocode of Intersection between two postings list

```
INTERSECTION(p1, p2)
1  answer = <>
2  while p1 ≠ NIL and p2 ≠ NIL
3       if DOCID(p1) == DOCID(p2)
4            ADD(answer, DOCID(p1))
5            p1 = NEXT(p1)
6            p2 = NEXT(p2)
7       elseif DOCID(p1) < DOCID(p2)
8            p1 = NEXT(p1)
9       else p2 = NEXT(p2)
10      return answer
```

To solve this problem we introduce *inverted index*, where for each term t, we must store a list of all documents that contain t and we identify each by docID, a document serial number.
In figure 7 is possible to note an example of inverted index, where we only store the docID where a word appear in a document.

The advantages of inverted index is that query requires just a scan and also we can store smaller integers, using *gap coding*, that will enable to use less amount of memory.

To execute an AND query the first approach consist to check each element of the two postings list that will cause $n * m$ operations, where n and m are the length of the two postings list, and we can achieve a better result if we sort the two postings list, so at least if we compare 1 and 2 we can avoid to consider to compare the element 1 with element that are greater than 2, so we need only $n + m$ comparison that will improve the performance of AND query.

The pseudocode to intersect two postings list in a AND query can be found on figure 8 and in case we have to compute an AND query between 3 operands, it is better to first compute an AND query between operands with smallest length and then compute an AND query with the other operand, so to compute the query "Brutus AND Caesar AND Calpurnia" in figure 9 we first compute "Brutus AND Calpurnia" and then compute "(Brutus AND Calpurnia) AND Caesar"

Figura 9: And query between Brutus, Calpurnia and Caesar