Università di Bergamo

Doctoral Thesis

Hints from the crowd

Author:
Paolo Fosci

Supervisor:
Prof. Giuseppe Psaila

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

in the

Meccatronica e tecnologie innovative
Department of Engineering

March 2015

Declaration of Authorship

I, Paolo Fosci, declare that this thesis titled, 'Hints from the crowd' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: *PaoloFosci*

Date: *27th February 2015*

"von Neumann gave me an interesting idea: that you don't have to be responsible for the world that you're in. So I have developed a very powerful sense of social irresponsibility as a result of von Neumann's advice. It's made me a very happy man ever since. But it was von Neumann who put the seed in that grew into my active irresponsibility!"

Richard Feynman - *Surely you're joking, Mr. Feynman!* (1985)

"Life is what happens to you while you're busy making other plans"

John Lennon - *Beautiful boy* (1980)

UNIVERSITÀ DI BERGAMO

Abstract

Meccatronica e tecnologie innovative
Department of Engineering

Doctor of Philosophy

Hints from the crowd

by Paolo Fosci

Can the crowd be a source of information? Is it possible to receive useful hints from comments, blogs and product reviews? In the era of Web 2.0, people are allowed to give their opinion about everything such as movies, hotels, etc.. These reviews are *social knowledge*, that can be exploited to suggest *possibly interesting items* to other people.

The goal of the *Hints From the Crowd* (HFC) project is to build a *NoSQL* database system for large collections of product reviews; the database is queried by expressing a natural language sentence; the result is a list of products ranked based on the relevance of reviews w.r.t. the natural language sentence. The best ranked products in the result list can be seen as the best hints for the user based on crowd opinions (the reviews).

The *HFC* prototype has been developed to be independent of the particular application domain of the collected product reviews. Queries are performed by evaluating a text-based ranking metric for sets of reviews, specifically devised for this system; the metric evaluates the relevance of product reviews w.r.t. a natural language sentence (the query).

Acknowledgements

My academic career has been like a *rollercoaster*. During the years, there have been periods of intense studies, followed by periods of... *other*. *John Lennon* was saying *life is what happens to you while your busy making other plans*. So, in life I had my long sharing of *other*.

It was thank to the *prof. Giuseppe Psaila* if, almost ten years after I got my degree at *Politecnico di Milano*, I decided to take the challenge of this *PhD*. A few years after graduation, *prof. Psaila*, who was my co-advisor, was looking for an assistant for his course of *Languages and Compilers* in this University, and got in touch with me to ask me if I can assist him. At that time I was working far away in Rome. But since the activity was interesting and not too much time consuming, I accepted with joy his proposal. For years, from march to may, I had to plan weekly transfer in order to take my lectures. It was tiring, stressful, but most of all, rewarding and fun! And so I got to know the University of Bergamo, and when I had the chance to take a sabbatical from work in order to attend this PhD program, well, I didn't miss the chance!

I've spent three years of studying, researching, working, teaching, doing the *cerberus* to the examination tests for students... in practice *fun*! Dealing with Prof. *Psaila* has always been interesting and mentally stimulating. And I learn a lot!

So I have just one thing to add: *Giuseppe, I stop swearing against LaTeX right now!*

I wish to dearly thank also *prof. Riccardo Riva*, who as PhD Co-ordinator was always helping me, not least in letting me to postpone my PhD final dissertation to the next year, as I was in trouble in writing this thesis after my returning back to old work at the formal ending of the PhD. I really hope his health troubles leaves him in peace. *Forza Professore!*

A special thank to prof. *Stefano Paraboschi* who was always available for a speech or a suggestion, and made me possible to attend the conferences and the summer schools where I have been during my PhD studies.

I can't forget all people, friends, in the computer science department:
Simone Mutti, *Steven Capelli*, *Paolo Vavassori*, *Davide Mora*, *Eros Magri*. Some of them had also the misfortune to have me as a teacher, and *still* they helped me when I was in troubles with servers! *C'mon Simone, we'll finish together, and you, Steven and Paolo, are the next! Follow Simone's way, not mine!*
And I want also to remember *Chiara Pisoni*, *Pasqualina Potena*, *Maurizio Toccu*, *Mario Arrigoni Neri*, prof.ssa *Patrizia Scandurra*, prof. *Angelo Gargantini*, prof. *Paolo Salvaneschi*, prof. *Tullio Caronna* (years ago he was my teacher in Chemistry at *Politecnico* but luckily for me he can't remember my exam!), and all the people working for the university. They all made my staying in department comfortable and relaxing, especially during lunch or coffee break. *Coffee! What a stunning invention!*
And also, I can't forget *Carlos Laorden* and *Javier Nieves* from University of Bilbao, who came in visit and made my staying in department extra special. *Javier, I still owe you an hangover :-D.*

A thought, to my PhD mates. Some of them had already got the PhD, others are about to finish! Even if our fields of studies were so different, it was interesting to have a different point of view when we were sharing the same room. And a special thank to *Rosalba Ferrari*, who as students representative made and excellent work!

Thanks to *Alfredo Cuzzocrea*, from University of Calabria, who co-authored with me and prof. *Psaila* some papers giving the project an higher resonance.

Surely I have to thank my *family*. And my friends as well, even if most of the time they were kidding me saying I was too old to get back to student life! So, after replying with a polite *tie'!*, I thank all them for their *support*.

A special thank to my friend, colleague, PhD mate (though in another university), co-author, inspirer, project web-interface designer, and *bla bla bla* (he hates *blas*), *Marcello*. He enlightens my mind with good taste and... knowledge! Knowledge about everything, beyond the work, beyond the research... beyond! ... and he helped me! He made the web interface! (But don't call him the *guy of colors*!)

Last but not the least, thanks to my friend *Mauro* for printing the cover of the *C:D*, and a special thank to *Sveta* for helping me to fix bugs with my english. Well, not only with my english! ;-)

PS.

Abbiamo fatto trenta, facciamo trentuno...

I want to thank my former advisor at *Politecnico di Milano*, prof. *Stefano Crespi-Reghizzi*, and my Computer Science teacher at high school, *prof. Roberto Bellavita*, because they *put the seed in that grew into my active irresponsibility.*

Contents

List of Figures

List of Tables

Abbreviations

APRV	**A**daptive **P**roduct **R**elevance **V**alue
BRV	**B**log **R**elevance **V**alue
HFC	**H**int **F**rom the **C**rowd
LSI	**L**atent **S**emantic **I**ndexing
PRM	**P**roduct **R**anking **M**etric
PRV	**P**roduct **R**elevance **V**alue
RTM	**R**ecursive **T**ermeset **M**ining (algorithm)
SVM	**S**pace **V**ector **M**odel
SW	**S**top**W**ords
TF-IDF	**T**erm **F**requency - **I**nverse **D**ocument **F**requency
VTC	**V**alid **T**ermeset **C**overage (algorithm)

Symbols

ad	termset average density
d	termset density
D_q^*	set of termsets derived from q^*
D_q	set of termsets derived from q
ES	term expansion set
ET	termset expansion set
p	product
q	query
q^*	expanded query / term-base
r	review
R	set of reviews
RD	set of relevant termsets
RES	term reverse expansion set
sc	semantic coefficient
t	generic term
T	termset
t^*	expanded term
T^*	expanded termset
w_q	termset weight according to a query

To the research of Light...

Chapter 1

Introduction

1.1 The context

A common *Web 2.0* user experience is to look for a specific product or service by means of specialized websites.

A very common situation is looking for a hotel room: the Internet is plenty of websites like *Booking*[1], *Venere*[2] or others that allow the user to search for a hotel in the desired period of the year and location, and as a result they return back the user a list of possible solutions. Usually it is also possible to filter the list of results according to predetermined parameters like the range of *price per night* of the room, or the *number of stars* of the hotel (e.g. *Venere*). More advanced websites (e.g. *Booking*) also allow hotel filtering according to the presence of certain facilities like *wifi-connection*, *air conditioning system* or *fitness centers* just to name a few.

But *what if* the user is arriving very tired on a Saturday late night and desires to sleep without being disturbed by rowdy people in the streets because there are clubs and bars very close to the hotel? or because the very next morning he/she's hearing

[1] http://www.booking.com
[2] http://www.venere.com

the bells from the near cathedral?
And *what if* the user is going abroad with his/her family and he/she desperately needs waiters that speak his/her language? or needs a room with painted shiny stars on the ceiling since otherwise his/her kids are not sleeping?
Nowadays all booking websites allow users to rate (usually with a number from 0 up to 10, or a certain number of *stars*) their staying in a hotel and write down a review about their experience. So, when a user is looking for a hotel, can also read comments made by other people about their experience in order to make his/her choice. *The only problem is that it could be a long and annoying task.*

Another common situation in *Web 2.0* is searching for a product to buy like a *camera* or a *car*. Again the Internet is plenty of websites for sales like *eBay*[3] or *Amazon*[4], which are probably the most famous at the moment.
In this case the user can ask for a certain product and the website *search-engine* answers back with a list of results that can be sorted by the price of the product or the distance of the seller. Often it is also possible to filter out products on the base of predetermined features that strongly depend on the kind of product the user is searching (e.g. the *size* for a dress, or the *number of megapixels* for a camera).
Again, also in this situation users are allowed to comment their experience. Usually comments are mostly related about the reliability of the seller, but also can report impression about the product that has been purchased.
So if a buyer wants to know if the *fancy webcam* that he/she wants to buy is easy to install, or wants to know if the instruction manual of the *microwave* oven he/she is interested into has a section in his/her language, he/she has the chance to retrieve the information by reading other people comments.

As final example let's consider the *Internet Movie Database* (*IMDb*).

[3] http://www.ebay.com
[4] http://www.amazon.com

IMDb is database with a website interface[5] that hosts cards related to every movie produced worldwide since 1880 (*"Sallie Gardner at a Gallop"*[6] which was actually produced before the *Lumière brothers* patented their first *cinèmatographe* in 1894) until present days. Every card holds a variety of information about the *director*, the *genre*, the movie *plot*, the *production year*, the *cast* that played in the movie, and in general whoever worked in the movie, just to name a few of the information that is possible to get from the *IMDb*. In addition, *IMDb* let (registered) users to review the movies they have seen, and hence even this information is available from a movie card.

IMDb also holds cards about people working in the movie system (mostly actors and directors) with their biography, the list of the movies in which they worked and other more or less interesting information.

All this information is linked together so it is possible for a user to browse the site: for instance from the card of the movie *"The Godfather"*[7] it is possible to reach the card of the director, that is *Francis Ford Coppola*[8], and then get the list of all the movies that Coppola has directed, and from this list reaching the card of the movie *"Apocalypse Now"*[9] where finally a user can discover that the actor *Harrison Ford*[10] played in this movie (yes, Harrison Ford was also in *"Apocalypse Now"*!).

In order to find out information, *IMDb* provides the user a search engine to directly access the card of the movie or the actor or director the user is interested into. The search engine can also be queried by setting a wide variety of parameters, in order to find out a sophisticated list of objects (movies or actors) like the list of all the *movies played in French language and shot in Africa between 1980 and 1995*, just to make an example.

But *what if* the user wants to know something about the *history of ancient Greece*

[5] http://www.imdb.com
[6] http://www.imdb.com/title/tt2221420
[7] http://www.imdb.com/title/tt0068646
[8] http://www.imdb.com/name/nm0000338
[9] http://www.imdb.com/title/tt0078788
[10] http://www.imdb.com/name/nm0000148

and the Persian Wars? And *what if* the user needs a movie with *funny, great, word jokes in simple English* in order to make it easy for his/her children to learn the language? Surely, the user can read reviews made by other people where probably the information he/she needs can reside. But there are movies like *The Matrix*[11] with more than 3000 reviews. And there are also so many movies...

1.2 Examples

Here are showed some examples taken from the Internet of the situations described in Section 1.1.

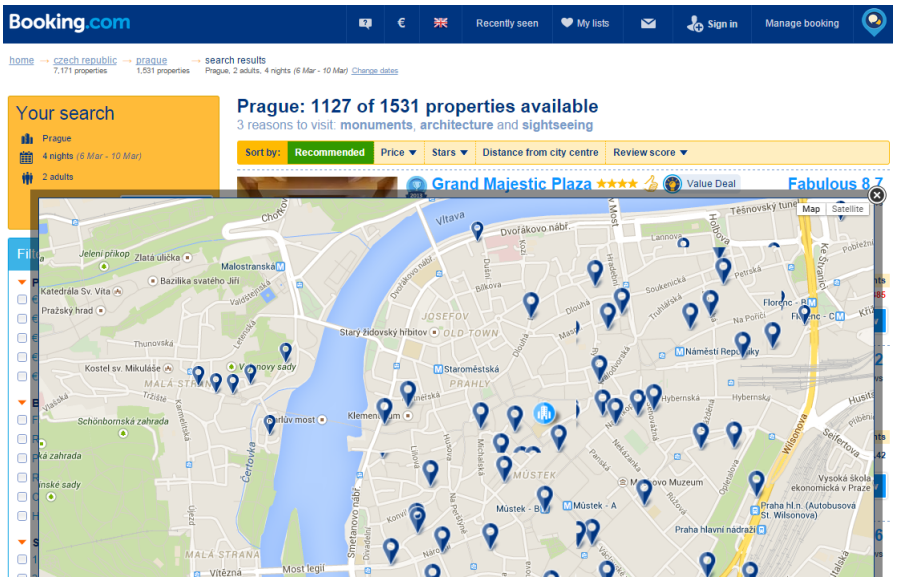

FIGURE 1.1: Booking.com - Map choosing hotel feature

The following screenshots, taken from the *Booking.com* website, will illustrate the typical user experience when dealing with the choose of a hotel. Figure 1.1 shows the *map-feature* to allow the user to pick a hotel from a map. Of course the website

[11] http://www.imdb.com/title/tt0133093/reviews

allows also the user to have a list of hotels, according to various parameters (such as *FreeWiFi*, or *price-range-per-night*) and ordered by price as shown in Figure 1.2.

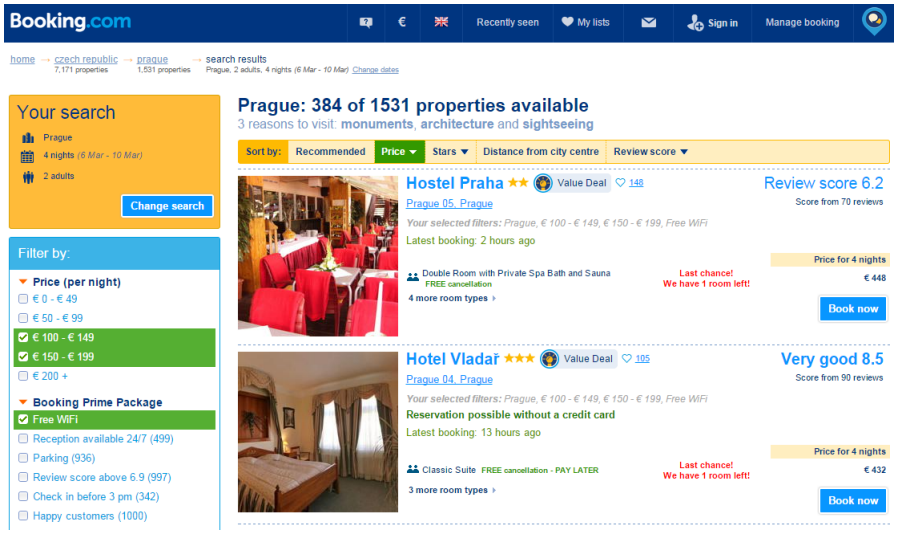

FIGURE 1.2: Booking.com - Hotel listing by price

The website has its own *automatic recommending system* in order to suggest the user which could be, according to its internal score, nice hotel solutions, as shown in Figure1.3.

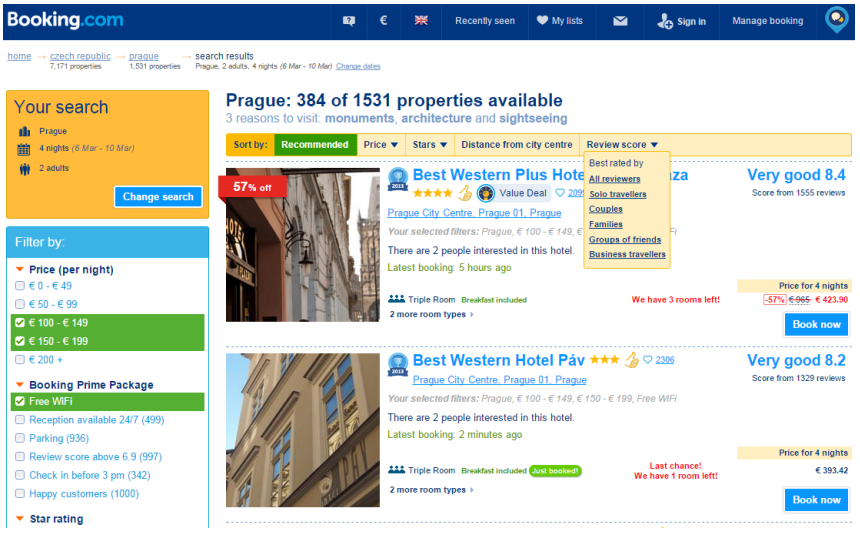

FIGURE 1.3: Booking.com - Hotel listing by website score

Hotels can also be order by reviewers's ratings, that can be an index built upon the average number of *stars* given very synthetically by the travelers, or a more complex index built upon various parameters as shown in Figure1.4. It's interesting to notice that the set of considered reviewers can be filtered according to those groups of travelers that better fit with the user (e.g. *families*, *solo travelers* or *couples*).

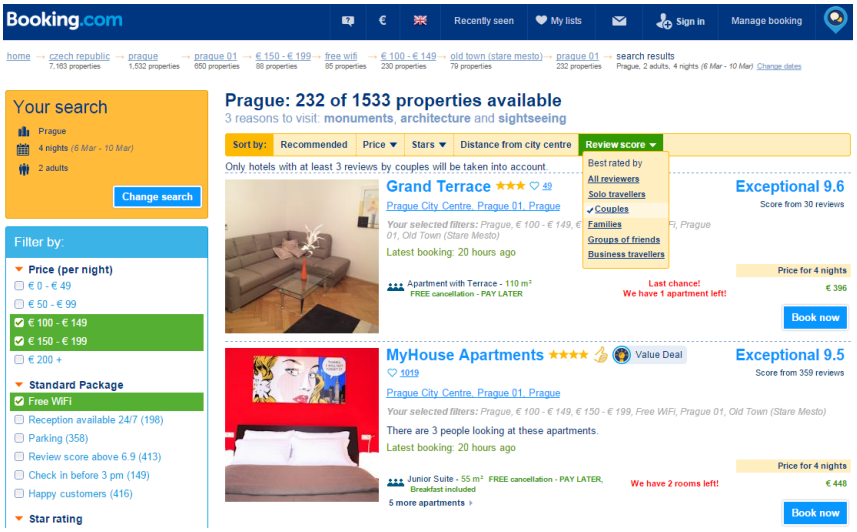

FIGURE 1.4: Booking.com - Hotel listing by travelers score

When choosing a hotel, the user can look the hotel card to check, by means of a dashboard and photographs, if the solution is suitable for his/her needs. Another feature the user can exploit, is reading people's reviews, that usually are divided into *PROs* and *CONs*. Figure 1.5 considers the case of a hotel with 124 reviews. As a hotel is frequented, such a number of reviews can represent a normal case. In the Figure (left side) is also shown the dashboard with the average value of the parameters (such as *cleanliness* or *comfort* to name a few) used to built the travelers average *score*.

If the user main goal is to find a hotel far from belfries, since during his/her staying, on sunday morning, he/she likes to sleep, well, he/she has to read a lot of reviews!

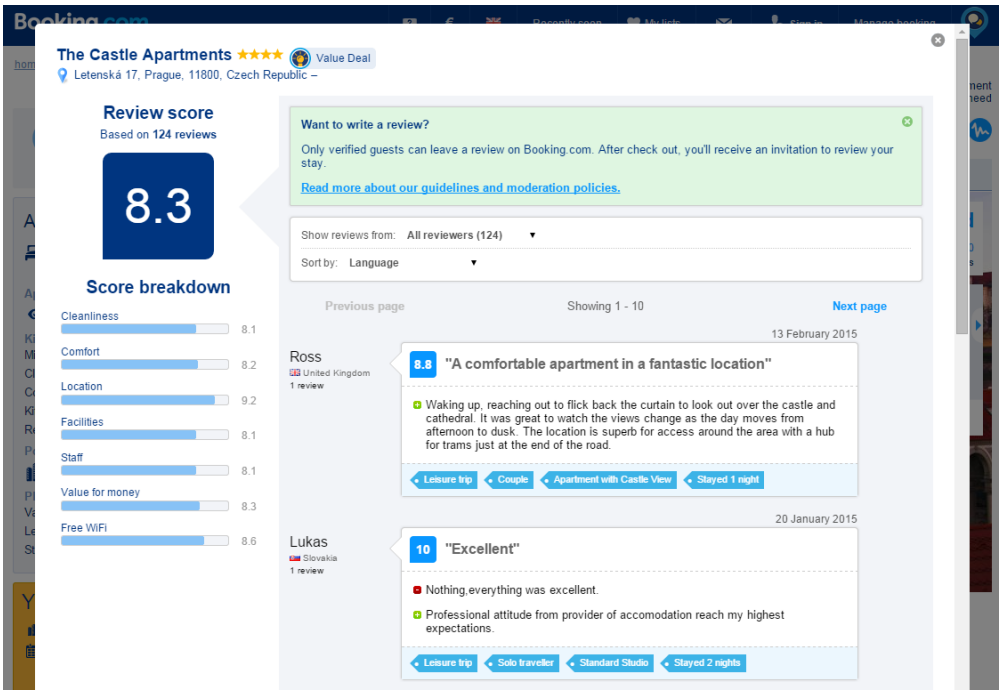

FIGURE 1.5: Booking.com - User reviews per hotel

Venere.com is a website similar to Booking.com. It's interesting to analyze Venere.com since it is one of the few examples of an *Italian website* that had a worldwide success.

Similarly to Booking.com website, also Venere.com has its own map-feature to allow the user to pick a hotel according to the site location he/she prefer (Figure 1.6). In this case as well, there is the chance for the user to filter hotels on the base of various parameters, such as *Budget* (i.e. *price-per-night* on Booking), *stars*, or the presence of several hotel services like *Free WiFi* or *Breakfast*. The user can have the list of matching hotel (see Figure 1.7) sorted by price, or according to a synthetic *star* rating, or a more refined *guest* (travelers) rating or even according to web-site rank (*Our Favorites*).

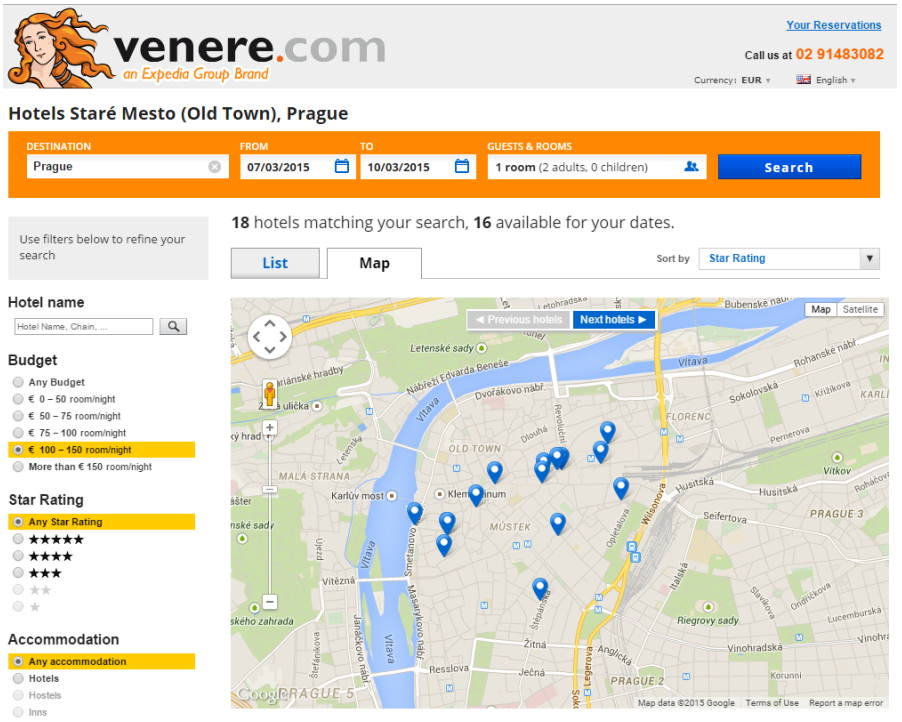

FIGURE 1.6: Venere.com - Map feature

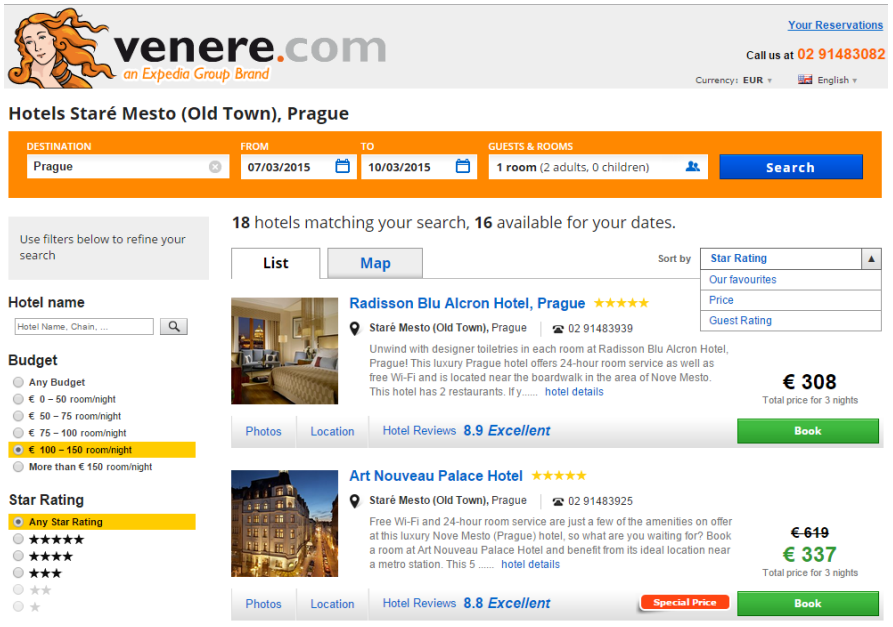

FIGURE 1.7: Venere.com - User listing choices

When a hotel seems interesting to the user, he/she can take a look at the hotel card, where still, can be found a dashboard with quality indexes. And, of course, travelers reviews!

Venere has a connection with the popular *TripAdvisor*[12] web site. Hence the source of reviews can be very wide. Figure 1.8 shows the case of a hotel with a total amount of *1432* traveler reviews! Reviews are just plain text and are not structured in *PROs* and *CONs*. But if the user goal is always to avoid *belfries*... well, *good luck*!

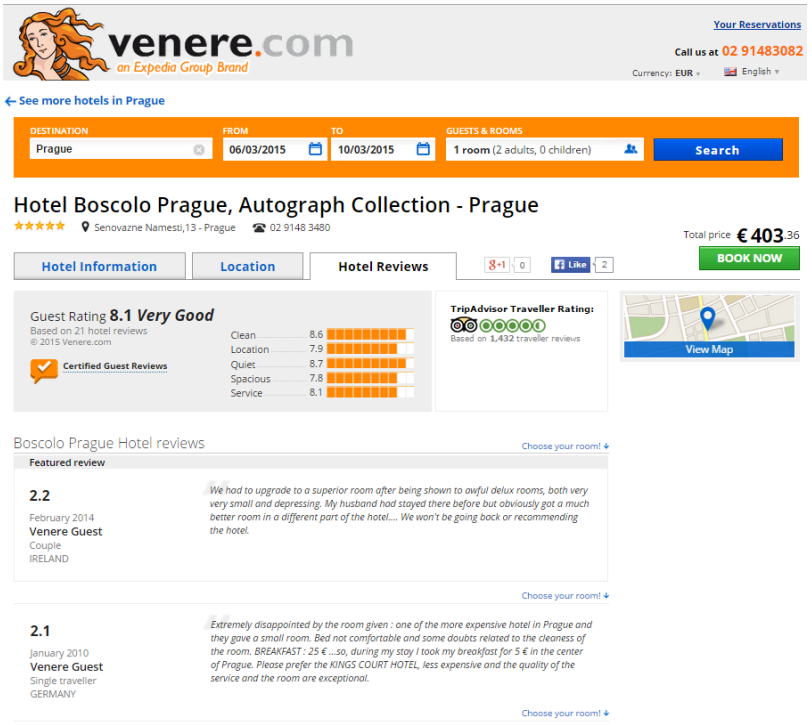

FIGURE 1.8: Venere.com - User reviews

Changing point of view, *Epinions.com*[13] *helps people make informed buying decisions. It is a premier consumer reviews platform on the Web and a reliable source*

[12] http://www.tripadvisor.com
[13] http://www.epinions.com

for valuable consumer insight, unbiased advice, in-depth product evaluations and personalized recommendations.[14]

Epinions.com is connected to *eBay.com*, but it must not be confused with an e-commerce website. Epinions.com is a repository of people's reviews, were registered users can talk about their experience with a *product* (or a *service*). Products are organized into *areas* of interest (e.g. *Electronics*, *Computers*, *Media* to name a few). Every area can recursively be structured, and the user can browse it, into sub-areas until reaching a quite defined class of homogeneous products. If the user has clear in mind what he/she is looking for, he/she can also fill a search-box with the desired product.

Figure 1.9 shows the case when the user is looking for a generic *digital camera*.

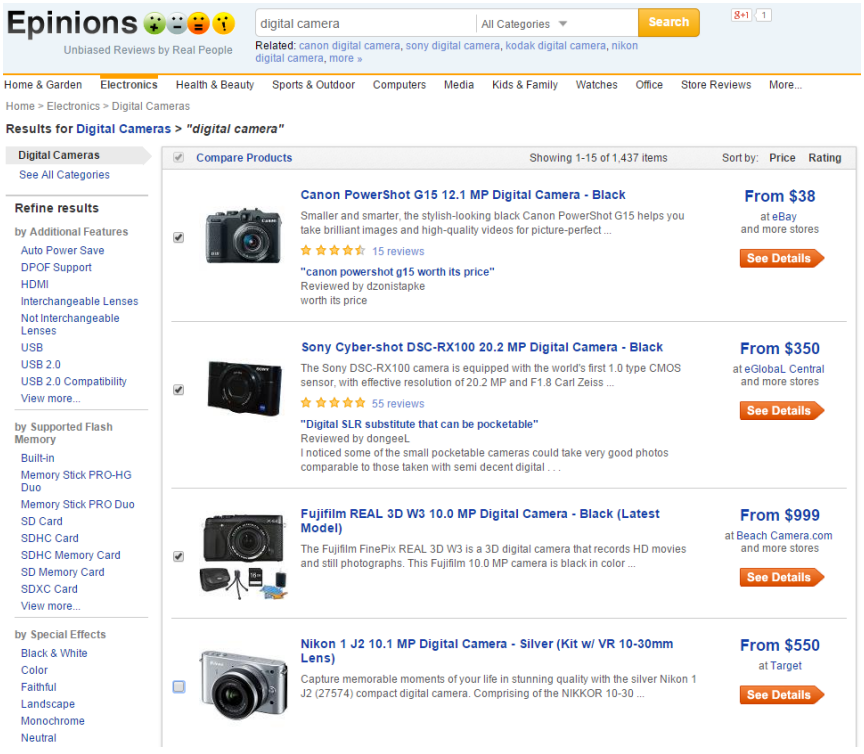

FIGURE 1.9: Epinions.com product listing

[14]From Epinions.com *About* page: http://www.epinions.com/about

The criterion by which products are sorted in the listing is explained by Epinions.com itself in the screenshot in Figure 1.10.

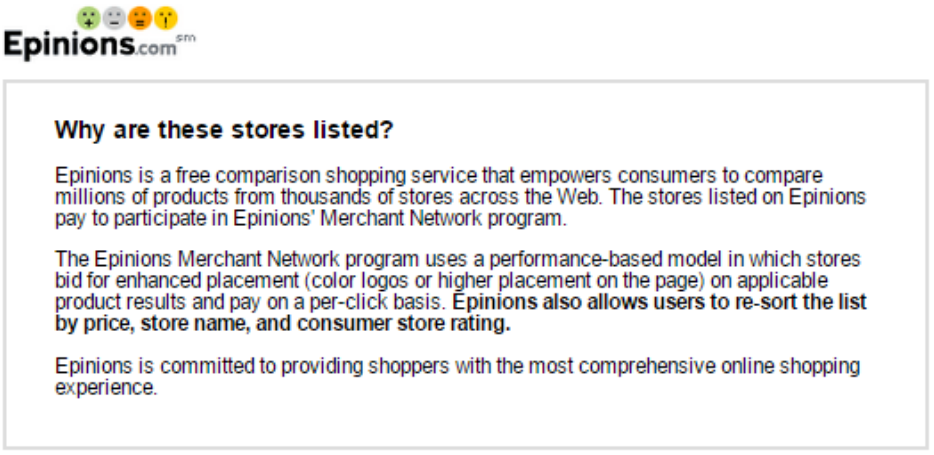

FIGURE 1.10: Epinions.com - Why are these stores listed?
Epinions.com criterion in sorting product listing

Epinions.com provides the user the chance to pick several (two or more) products up, and compare them, as shown in Figure 1.11, in order to let the user to have deep awareness about what to buy, as it is in Epinons.com's scope. The set of parameters (product features) used for the comparison is product-depending. The set can include *mega-pixel resolution* and *display dimensions* for a digital camera, whereas for a car it is important to know the *fuel consumption per km* and the *number of seats for passengers*.

Once the user picks from the listing a well defined product, he/she can see the product card, where in addition to all product features, the user can find reviews given by other peoples. As shown in Figure 1.12, reviews includes a synthetic evaluation with a certain number of *stars*, *PROs* and *CONs*, and a brief summary written in natural language.
In this case as well, if the user is looking for a digital camera with a *fancy design for left-handed people*, he/she has to read reviews.

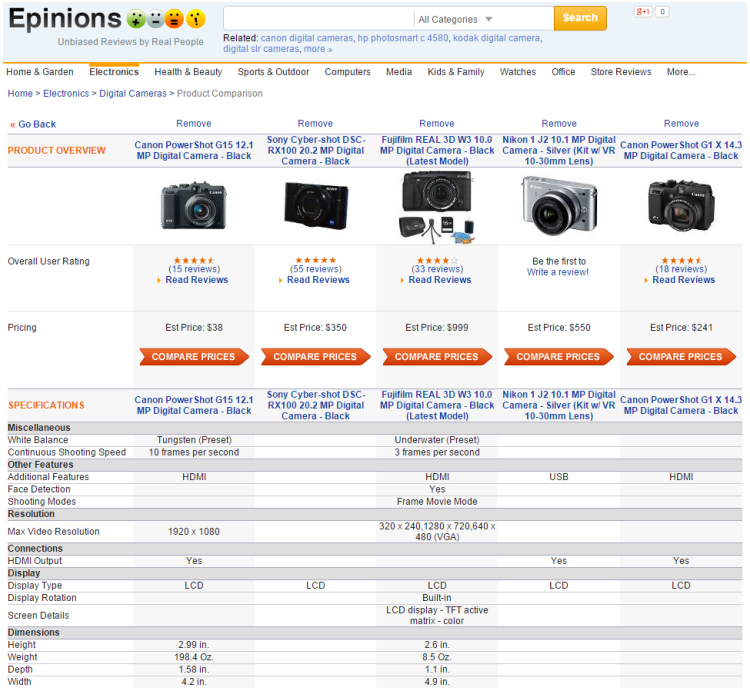

FIGURE 1.11: Epinions.com - product comparison

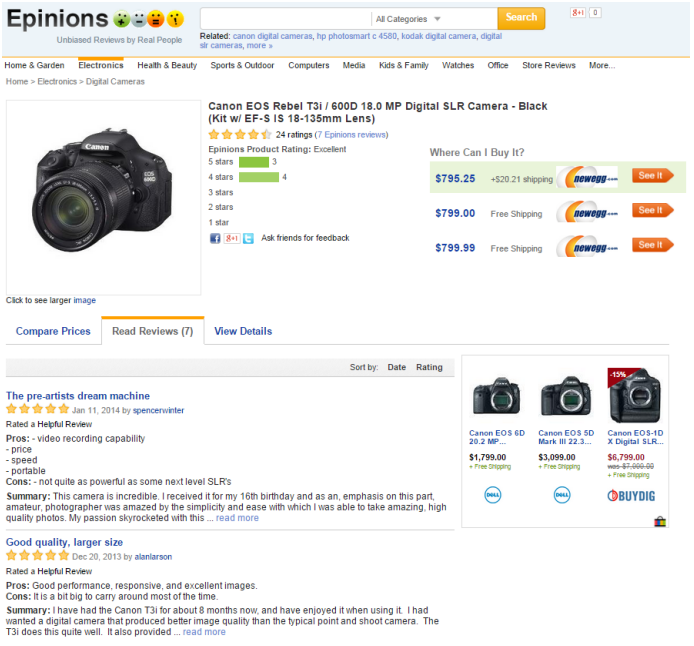

FIGURE 1.12: Epinions.com product reviews

To conclude this overview, Figure 1.13 shows the typical card for a movie on the *IMDb* web site. The IMDb case has been already analyzed in Section 1.1, and will be resumed later. Here is shown the card of a 2014 movie (*The imitation game*), about *Alan Turing*'s life, that has been aired in theaters less than one month ago (by the time when this thesis has been written). The movie card holds a wide variety of different information, such as *movie plot*, *director*, *stars* (in this case with the meaning of main actors), the complete *cast list*, and so on. Further more, photos and videos related to the movie are available for user consumption, and there is also a small list of similar movies, besides a set of other interesting information. All these information is linked together to allow the user to browse them.
In the particular case of this *Alan Turing* related movie, it's interesting to notice that since its release, the movie has been rated by *133.217* users, got *353* specialized magazine's critics, and has been already reviewed by *367* IMDb registered users.

If *cinema* is a passion, people's reviews can be sometimes an interesting reading. *Sometimes*.

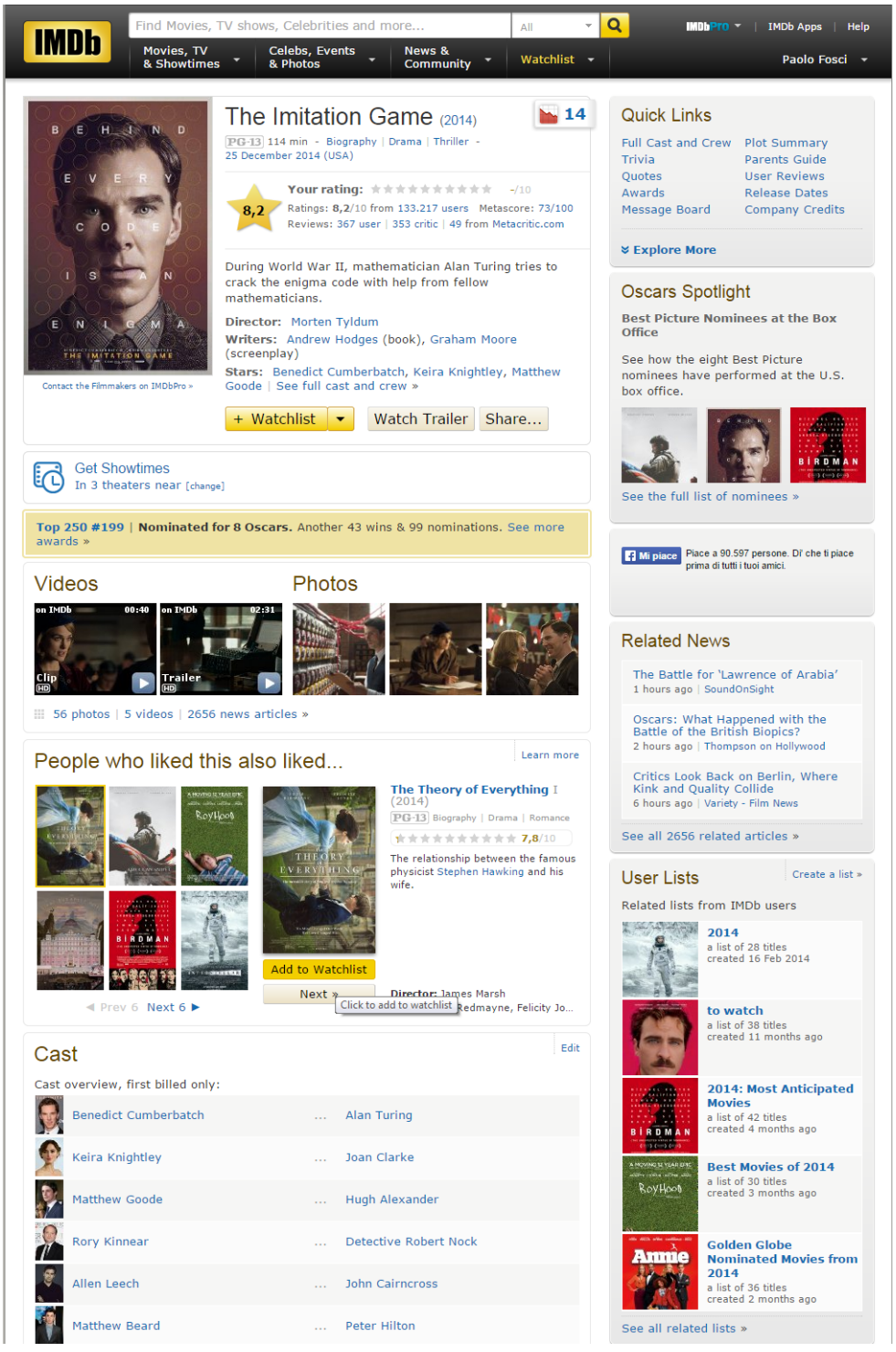

FIGURE 1.13: IMDb.com - *The Imitation game* card

1.3 Purpose of the project

Situations described in Section 1.1, and carefully analyzed in Section 1.2 are quite common user experiences in everybody's life. Often a person is looking for something, a product or a service, and his/her needs go beyond the usual criterion given by an automatic search engine in a specialized website. Often the answer to a very specific user requests resides in other people comments and reviews. In the real world this would be the case of asking an *opinion* or a *hint* to someone else, a friend or an acquaintance. But, while in the real world the issue might be the small number of people to ask an opinion and also their probable lack of a complete knowledge related to the request, in the Internet the issue is the opposite: there's an overload of information that makes quite impossible for a user to get the information he/she needs unless reading a few comments randomly, since reading all comments is a really severe task for a human being.

The purpose of this thesis is to present a *search engine* that starting from a generic user request expressed in natural language, like could be English or Italian or Russian, read all the other people comments and reviews in order to find out those products or services whose comments are closer to the user request.
An important aspect to consider is that a user request can express a wide variety of meanings that are included into, or close to, the original request. As an example, consider how close are the concepts represented by an *old oak* and a *green ancient tree*. So, an important task of the search engine is not only discovering in the other user comments the original meaning of the request, but also the discovery of all the similar meanings or part of them.

Talking about *search engines* might be quite improper, as the concept can be easily misconfused with the classic concept of general purpose web-search-engine like *Google*, *Yahoo* or *Bing* are.

A closer concept is the well-known concept of *recommendation system*. But in this case as well, the combination might be improper, since usually a *recommendation system* works automatically on the base of a predetermined criterion.
What would be better to say, is that the preconized system is something in between a general purpose search engine, since for the user it works according to the same way, and a versatile product recommendation system, as this is its real scope.

The basic idea underlying the system is related to the classic concept in information retrieval of *frequent itemset extraction*, and from this starting point a *ranking model* has been developed together with a technology that in a few words allows to read all the comments and then rank the products found in an acceptable execution time for the user.

Since the task behind the search engine is the same as asking *hints* to people, and considering that on the Internet the people to whom ask a hint can be really great, a *crowd*, the project has been called **Hints From the Crowd** (*HFC*).

1.4 Structure of the thesis

The rest of this thesis is structured as follow:
Chapter 2 describes the state of the art in information retrieval showing the main technologies that influenced the *HFC* system and analyze also a few examples of real recommending system in well known sites on the Internet. Chapter 3 introduces the retrieval model that stands at the base of the *HFC* system, and gives also a glance to the historical reasons that justify model design choices. Chapter 4 shows the most significant algorithms used in order to grant *HFC* system acceptable performance. Chapter 5 presents the architecture of the *HFC* system and the data model used. Chapter 6 describes the datasets used for testing and discuss the performance of the

HFC system. Finally Chapter 7 summarizes the objectives achieved by the *HFC* system,

The appendices briefly show the main external tools or libraries used in the project. Specifically: Appendix A is related to the *pos-tagger* of the *Stanford Natural Language Processing Group*; Appendix B describes *Wordnet*, an ontology used to expands the meaning of words in a user request; Appendix C presents *Apache Lucene* that is a library for information retrieval based on the *Space Vector Model* used at the beginning of the project as a benchmark to compare results during early tests; Appendix D is about *Entity Recognition* & *Linked data* that represent the next evolutive step of this project; and finally Appendix E enlists the queries used for the main tests.

Chapter 2

State of the art

Before describing the object of this work, it is necessary to have a glance about the state of the art about the mostly used technologies concerning information retrieval. This Chapter will present a quick overview about the methodologies that mostly influenced this work. In particular will be treated:

- *Vector Space Model*
- *tf-idf* model
- *Latent Semantic Indexing*
- *PageRank* algorithm
- *Itemset mining*
- *Amazon Recommendation System*
- *Youtube Recommendation System*
- *IMDb Recommendation System*

Vector Space Model, *tf-idf* model and *Latent Semantic Indexing* are related to the initial issues when addressing the task of designing a search engine dealing with natural language analysis. *PageRank* is Google's base algorithm, and it is not possible

not talking about it here, mostly to explain why this algorithm can't be applied for *HFC*'s goal. *Itemset Mining* is a technique widely used in *recommendation systems* and stands at base of the HFC project. *Amazon* and *Youtube* recommendation systems are just examples of real and successful applications of the techniques underlying the concept of *itemset*, and implicitly they reveal what they don't perform by showing what they do perform[1]. From the comparison between Google, on one side, and the recommendation systems cited before, on the other side, emerges that the *HFC* project stands in the middle of the two approaches as it effectively acts like a search engine, but its technology is more similar to a recommendation system. *IMDb* case study yet is an another recommendation system, but still it is interesting to investigate as it represents the source of the main application-data of the *HFC* project.

2.1 Vector Space Model

Vector Space Model[2] (*VSM*) is a very classic and wide used model in information retrieval. *VSM* is the base of a large variety of applications and indexing tools such as *Apache Lucene* (see Appendix C).

In *VSM* documents and queries are represented as vectors.

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j}) \tag{2.1}$$

$$q = (w_{1,q}, w_{2,q}, \ldots, w_{t,q}) \tag{2.2}$$

Each dimension corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. Several different ways of computing these values,

[1] As a reminder, the goal of the *HFC* project is to search and rank products on the base of people's comments after a query expressed by a user in natural language.

[2] main source: *Wikipedia*

also known as (term) *weights*, have been developed. One of the best known schemes is *tf-idf* weighting.

The definition of term depends on the application. Typically terms are single *words*, *keywords*, or *longer phrases*. If the words are chosen to be the terms, the dimensionality of the vector is the number of words in the vocabulary (the number of distinct words occurring in the corpus).

Vector operations can be used to compare documents with queries.

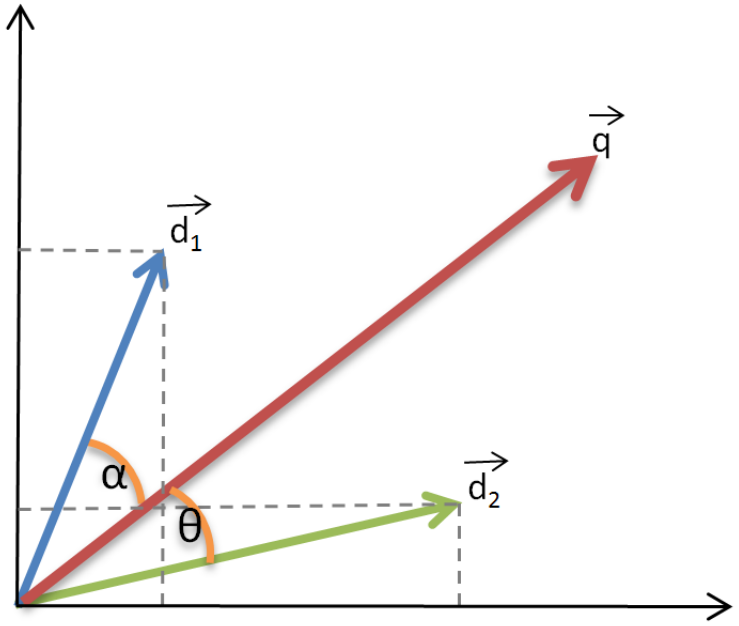

FIGURE 2.1: Documents and query represented as vectors
source: *Wikipedia*

Relevance rankings of documents in a keyword search can be calculated, using the assumptions of document similarities theory, by comparing the deviation of angles between each document vector and the original query vector where the query is represented as the same kind of vector as the documents (see Figure 2.1). In practice,

it is easier to calculate the cosine of the angle between the vectors, instead of the angle itself:

$$\cos\theta = \frac{d_2 \cdot q}{\|d_2\| \, \|q\|} \tag{2.3}$$

where $d_2 \cdot q$ is the intersection (i.e. the dot product) of the document d_2 and the query (q in the figure) vectors, $\|d_2\|$ is the norm of vector d_2, and $\|q\|$ is the norm of vector q. The norm of a vector is calculated as such:

$$\|q\| = \sqrt{\sum_{i=1}^{n} q_i^2} \tag{2.4}$$

As all vectors under consideration by this model are element wise nonnegative, a cosine value of zero means that the query and document vector are orthogonal and have no match (i.e. the query term does not exist in the document being considered).

The main known limitations of *VSM* are:

- long documents are poorly represented because they have poor similarity values (a small scalar product and a large dimensionality).
- *semantic sensitivity*; documents with similar context but different term vocabulary won't be associated, resulting in a *false negative match*.
- the order in which the terms appear in the document is lost in the vector space representation.

Many of these difficulties can, however, be overcome by the integration of various tools and lexical databases such as *WordNet* (see Appendix A).

2.2 *tf-idf* model

As stated in Section 2.1, several different ways of computing (term) *weights* for the *Vector Space Model* have been developed. The most immediate is the *Boolean Model* where for each term in a document vector a 1 value indicates the presence of the term in the document, while instead a 0 value indicates the absence of the term in the document. A more advanced model is the *Frequency model*, where for each term in a document vector is reported the number of occurrences of the term in the document.

Both of these models suffer from the problem of how important is a term to a document in a collection or corpus.

Tf-idf[3], acronym for *term frequency-inverse document frequency*, is a numerical statistic that reflects how important a word is to a document in a collection or corpus. *Tf-idf* is the product of two statistics, *term frequency* (tf) and *inverse document frequency* (idf). Various ways for determining the exact values of both statistics exist. In the case of the *term frequency* $tf(t, d)$, the simplest choice is to use the raw frequency of a term in a document, i.e. the number of times that term t occurs in document d. The *inverse document frequency* (idf) is a measure of whether the term is common or rare across all documents. *Idf* is obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2.5}$$

where N is the total number of documents in the corpus and $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears (i.e., $\mathrm{tf}(t, d) \neq 0$). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust

[3]main source: *Wikipedia*

the formula to $1 + |\{d \in D : t \in d\}|$. Mathematically the base of the log function does not matter and constitutes a constant multiplicative factor towards the overall result. Then *tf-idf* is calculated as

$$\text{tf} - \text{idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \tag{2.6}$$

A high weight in $tf - idf$ is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms. Since the ratio inside the *idf*'s log function is always greater than or equal to 1, the value of *idf* (and $tf - idf$) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the *idf* and $tf - idf$ closer to 0.

2.3 Latent Semantic Indexing

Retrieval methods suffer from two well known language related problems called *synonymy* and *polysemy*[4].

Synonymy describes that a object can be referred in many ways, i.e., people use different words to search for the same object. An example of this are the words *car* and *automobile*.

Polysemy is the problem of words having more than one specific meaning. An example of this is the word *jaguar* which could mean a well known car type or an animal. *Latent Semantic Indexing* (*LSI*) offers a dampening of synonymy. By using a *Singular Value Decomposition* (*SVD*) on a term by document matrix of term frequency. The dimension of the transformed space is reduced by selection of the highest singular values, where the most of the variance of the original space is. By

[4]source:
http://cogsys.imm.dtu.dk/thor/projects/multimedia/textmining/node10.html

using *SVD* the major associative patterns are extracted from the document space and the small patterns are ignored. The query terms can also transform into this subspace, and can lie close to documents where the terms does not appear. The advantage of *LSI* is that it is fully automatic and does not use language expertise and the positive side effect is that the length of the document vector becomes much shorter. Empirical studies of LSI have been good. LSI has also been examined analytically. By comparing *LSI* to multidimensional scaling it has been shown that *LSI* preserves the document space optimally when using the inner product similarity function. The same article implies that this applies also to other similarity measures. By using *Bayesian regression model* it is shown that by removing the small singular values, statistically dubious information are being removed and also specification errors are reduced.

2.4 PageRank

When addressing the task of developing a search engine the natural comparison is with *Google*. *Sergej Brin* and *Larry Page* introduced Google in their 1998 famous article *The anatomy of a large-scale hypertextual web search engine* ([7]) where they described the *PageRank* algorithm that stands at the base of Google:

> *We assume page A has pages* $T_1 \ldots T_n$ *which point to it (i.e., are citations). The parameter d is a damping factor which can be set between* 0 *and* 1. *We usually set d to* 0.85. *There are more details about d in the next section. Also* $C(A)$ *is defined as the number of links going out of*

page A. The PageRank *of a page A is given as follows*[5]*:*

$$PR(A) = (I - d) + d(\frac{PR(T1)}{C(T1)} + ... + \frac{PR(TI)}{C(TI)}) \tag{2.7}$$

Note that the PageRanks form a probability distribution over Web pages, so the sum of all Web pages' PageRanks will be one.
PageRank or $PR(A)$ *can be calculated using a simple iterative algorithm, and corresponds to the principal* eigenvector *of the normalized link matrix of the Web.*

As claimed by Brin and Page, the basic idea behind PageRank[6] is a revised form a the calculation of the eigenvectors of a matrix, a well known concept wide used in networks analysis ([9]). The idea can be simplified with the adage:

a page is as much relevant as relevant are the pages that link it.[7]

PageRank can then be applied to all those scopes where there is a network.
The situations described in Section 1.1 are not presenting then the ideal conditions to apply a *PageRank* approach, as from those web-sites it is not possible to infer any network useful for *HFC* goal. Nevertheless, in the premises of this thesis was discussed about the analysis of the natural languages and, as stated in Section 2.1, to avoid *semantic sensitivity* issues it is necessary to use lexical databases like *Wordnet* where still the influence of *PageRank* can be felt, as reported by *Wikipedia*[8]

[5]Actually as reported by *Wikipedia* in *https://en.wikipedia.org/wiki/PageRank#Damping_factor* *the correct form of the Formula 2.7 should be:*

$$PR(A) = \frac{(I - d)}{N} + d(\frac{PR(T1)}{C(T1)} + ... + \frac{PR(TI)}{C(TI)})$$

[6]PageRank is deeply analyzed by Brin&Page in their following 1999's article [8]

[7]The same concept can be adapted to *Social Networks* like *LinkedIn* for instance, where the relevance of a professional profile is as much important as important are the people linked to the professional profile

[8]http://en.wikipedia.org/wiki/PageRank

In lexical semantics it has been used to perform Word Sense Disambiguation and to automatically rank WordNet synsets according to how strongly they possess a given semantic property, such as positivity or negativity.

A dynamic weighting method similar to PageRank has been used to generate customized reading lists based on the link structure of Wikipedia.

2.5 Itemset Mining

Frequent Itemset mining is a method developed for market basket analysis. It aims at finding regularities in the shopping behavior of customers of supermarkets, mail-order companies, on-line shops etc. More specifically *frequent Itemset mining* find *sets of products that are frequently bought together*[9].

A formal description is the following: let $B = \{i_1, \ldots, i_m\}$ be a set of items. This set is called the *item-base*. Items may be products, special equipment items, service options etc. Any subset $I \subseteq B$ is called an item set. An item set may be any set of products that can be bought (together).
Let $T = (t_1, \ldots, t_n)$ with $\forall k, 1 \leq k \leq n : t_k \subseteq B$ be a vector of *transactions* over B. This vector is called the *transaction database*. A transaction database can list, for example, the sets of products bought by the customers of a supermarket in a given period of time.

Let $I \subseteq B$ be an *itemset* (i.e. item set) and T a transaction database over B. A transaction $t \in T$ *covers* the itemset I or the itemset I is contained in a transaction $t \in T \iff I \subseteq t$.
The set $K_T(I) = \{k \in \{1, \ldots, n\} | I \subseteq t_k\}$ is called the *cover* of I w.r.t. T, that is the index set of transactions in T that contain the itemset I.

[9] Main source Christian Borgelt: http://www.borgelt.net/slides/fpm4.pdf

The value $\sigma_T(I) = \frac{1}{n}|K_T(I)|$ is called the *support* of I w.r.t. T. In other words the support $\sigma_T(I)$ represents the fraction of transactions in T that contain the itemset I.

The extraction of all possible itemsets in a transaction database is called *Itemset Mining* operation. Itemset mining is a potentially hard problem and brute force approach in real case applications are not feasible as the number of possible itemsets grows exponentially with the number of items.
Indeed, given $m = |B|$ the number of items in the item-base B, the number of all possible itemset is $2^m - 1$, that is the cardinality of the power set of B minus the empty set. Notice that given $l = |I|$ with $1 \leq l \leq m$ as the cardinality of an itemset I, the number of all itemsets of l cardinality is given by the *Newton binomial coefficient* $\binom{m}{l}$.

Another important property of itemsets is $\forall I : \forall J \supseteq I : K_T(J) \subseteq K_T(I)$, that is the *cover* $K_T(J)$ of every itemset J contained in an itemset I is a superset of the *cover* $K_T(I)$. From this property follows the fundamental $\sigma_{\mathbf{T}}(\mathbf{J}) \geq \sigma_{\mathbf{T}}(\mathbf{I})$ $\forall J \subseteq I$, that is the *support* $\sigma_T(J)$ of every itemset J contained in an itemset I is greater than the *support* $\sigma_T(I)$.
On the base of the previous properties it is possible to partially order and depict itemsets in a (direct) acyclic graph which is called *Hasse diagram*. Figure 2.2 shows the *Hasse diagram* of a an item-base of five elements $\{a, b, c, d, e\}$. Figure 2.2 also shows the number of itemset for every cardinality l.

In real applications only *frequent* itemsets are interesting. In other words, given $\overline{\sigma}$ as *minimum support*, only itemsets with a support $\sigma_T \geq \overline{\sigma}$ should be considered.
Under this observation, various efficient algorithms have been proposed for Itemset Mining. The most famous algorithms are *aPriori*[10], *Eclat*[11] and *FP-Growth*[12]. All algorithms start from the dual consideration that *no superset of an infrequent itemset can be frequent*, or *all subsets of a frequent itemset are frequent*, thus mine

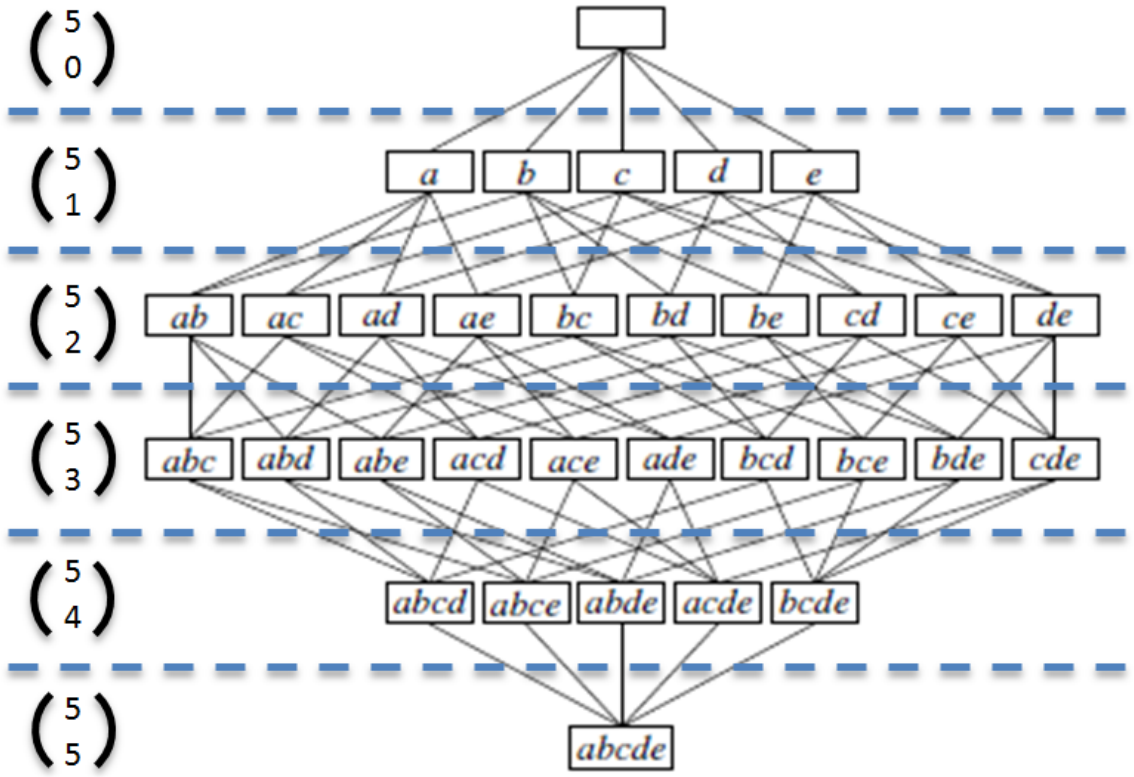

FIGURE 2.2: Hasse diagram related to an *item-base* of 5 items $\{a, b, c, d, e\}$

itemset starting from the smallest trying to build bigger itemset by adding new item, or trying to find the maximums itemsets and then derive the others as subsets.

2.6 *Amazon* Recommendation System

Amazon recommendation system is described by its authors *Linden*, *Smith* and *York* in their article published on *IEEE Internet Computing* in 2003 [13] and it loosely depends on the concept of itemset mining.

Amazon uses *recommendation systems*[10] to recommend new products to its users. It wants to serve its users with products it believes the user will be interested in seeing and then purchasing. Amazon's method of recommendation is complex, but it is centered around the products a user views and purchases. They utilize a system known as *item-to-item collaborative filtering*.

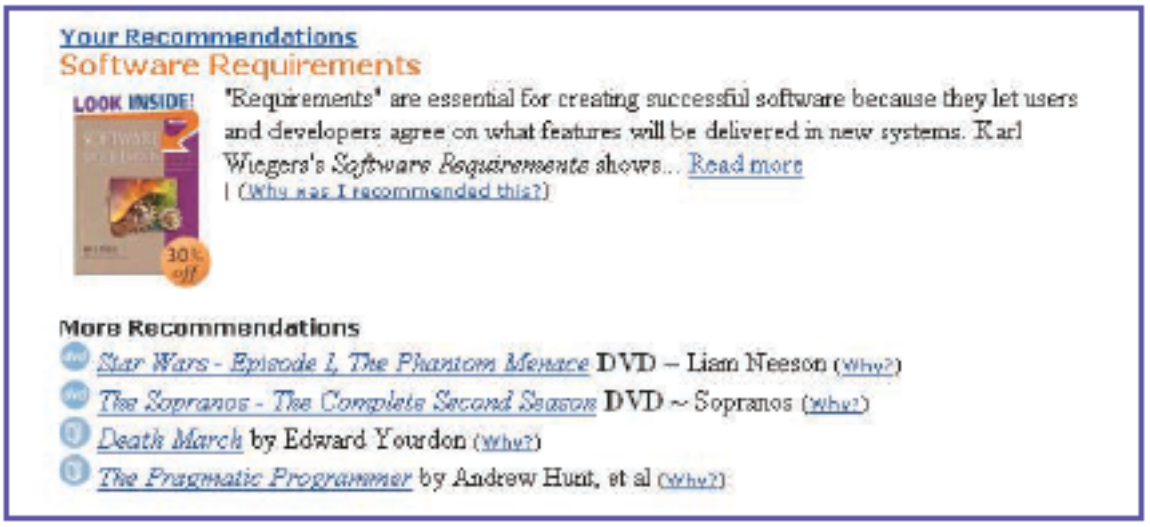

FIGURE 2.3: Amazon recommendations based on user profile

Item-to-Item Collaborative Filtering. Amazon's Item-to-Item Collaborative Filtering focuses on the *items* rather than the *users*. It does not try to match the user to similar customers. Instead, it matches each of the user's purchased and rated items to similar items, and then it merges the similar items (*itemset*) among them into one recommendation list unique to the user. In order to determine the match for a given item with the greatest amount of similarity, the algorithm builds a table of similar items. It does that by looking for items that customers tend to purchase together. For example, people would likely buy a bicycle along with a helmet and

[10] A brief summary can be found on this Princeton page:
http://scenic.princeton.edu/network20q/wiki/index.php?title=Amazon_and_Youtube%27s_Recommendation_Systems

padlock. Then, the recommendation system builds a product-to-product matrix by iterating through all item pairs and computing a similarity metric for every single pair. There are any number of ways to compute a similarity metric. One common way is to use the cosine measure method (see 2.1), where each vector corresponds to an item rather than a customer, and the vector's M dimensions correspond to customers have bought that item. This is a very time-intensive operation - the running time is $O(N^2 * M)$ in the worst case, though it's usually $O(N * M)$ in practice. This is because most customers don't have many purchases. With the similar-items table, the algorithm then searches for items similar to each of the user's purchases and ratings, brings those items together, then recommends the items that are the most popular or have the highest similarity metric. Since the values are already calculated, not much time is spent in this phase.

FIGURE 2.4: Amazon recommendations based on the items in the customer's cart

Scalability. Amazon.com has more than 29 million customers (*in 2003*[11]) and several million catalog items. This means that there is an absolutely massive amount of data that must be worked with. Unfortunately, this breaks the back of many algorithms designed for something far smaller. Indeed, almost all existing algorithms

[11] The number of customers raised up to 244 million in 2014 as reported by GeekWire: http://www.geekwire.com/2014/amazon-adds-30-million-customers-past-year

were evaluated over small data sets. However, Amazon's Collaborative Item-To-Item filtering is very scalable. The key to its scalability and high performance is that it computes the similarity metric and creates the table of similar items, a very expensive operation, offline. As a result, the algorithm has to do relatively little online. It merely looks up similar items for the user's purchases and ratings using data it has already created. Unlike traditional collaborative filtering algorithms, Amazon's also does well just a very limited amount of user data. It can even product quality results with just two or three products

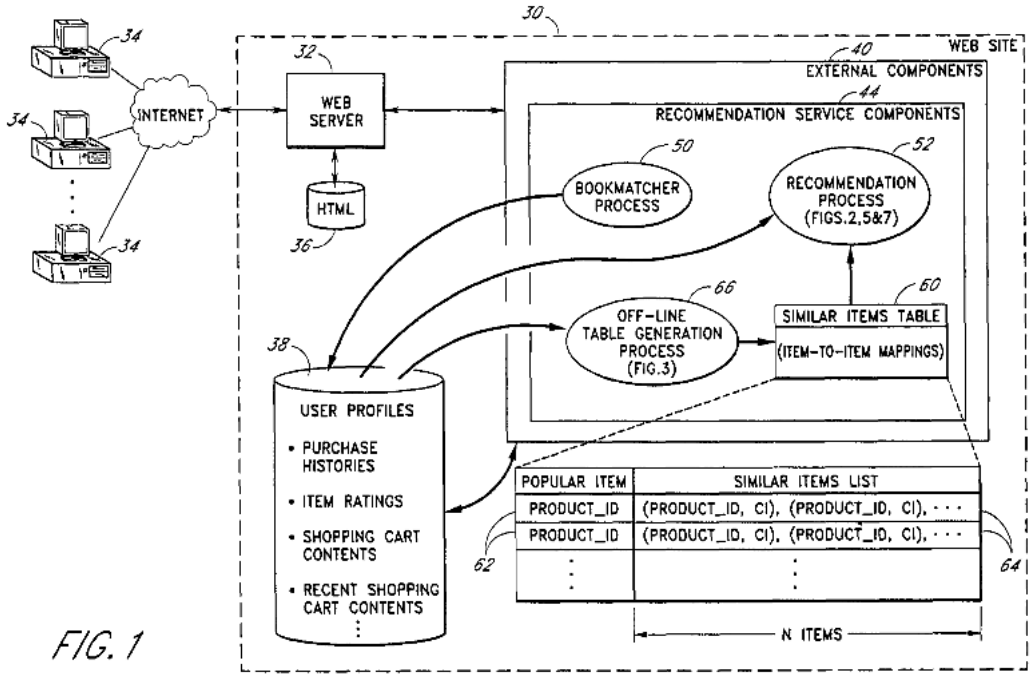

FIGURE 2.5: Amazon Recommendation System Architecture

Just for reader information, with no intention or need to explain it, Figure 2.5 shows Amazon Recommendation system's architecture and it is taken from is taken from *Linden* (et al.)'s Recommendation System US Patent [14].

2.7 *Youtube* Recommendation System

Youtube recommendation system is described by *Davidson et al.* in [15].

Youtube utilizes recommendation systems to bring videos to a user that it believes the user will be interested in. They are designed to increase the numbers of videos the user will watch, increase the length of time he spends on the site, and maximize the enjoyment of his Youtube experience.

User Activity In order to obtain personalized recommendations, Youtube's recommendation system combines the related videos association rules with the user's personal activity on the site. This includes several factors. There are the videos that were watched - along with a certain threshold, say by a certain date. After all, a user doesn't want to count videos watched from 2 years ago if the user has watched enough videos, most likely. Also, Youtube factors in with emphasis any videos that were explicitly *liked*, added to favorites, given a rating, added to a playlist. The union of these videos is known as the *seed set* (that still refers to the *itemset concept*). Then, to compute the candidate recommendations for a *seed set*, Youtube expands it along the related videos.

Introducing Relevance and Diversity Limiting factors are utilized as well. For example, Youtube will only allow a certain number of videos from the same uploader in the related videos set. Furthermore, Youtube checks certain elements related to each video and scores and ranks the videos according to these values. Youtube uses user specificity signals to boost videos that are closely matched with the user's unique taste and preferences. The recommendation engine looks at the user's watch history, view count, and time of watch for the video in the *seed set* to obtain a comprehensive view. Then, of the videos it is able to obtain that are both

relevant and in-tune with user preferences, a small number is taken - enough to fill the related videos section. The signals can be categorized into three groups corresponding to three different stages of ranking: 1) video quality, 2) user specificity and 3) diversification. Then, there are overall video quality signals that suggest that the user will appreciate the video regardless of his individual tastes. These values include the view *count/popularity*, ratings, number of *likes* and *dislikes*, commenting, favoriting, sharing activity, and how long it has been on the site.

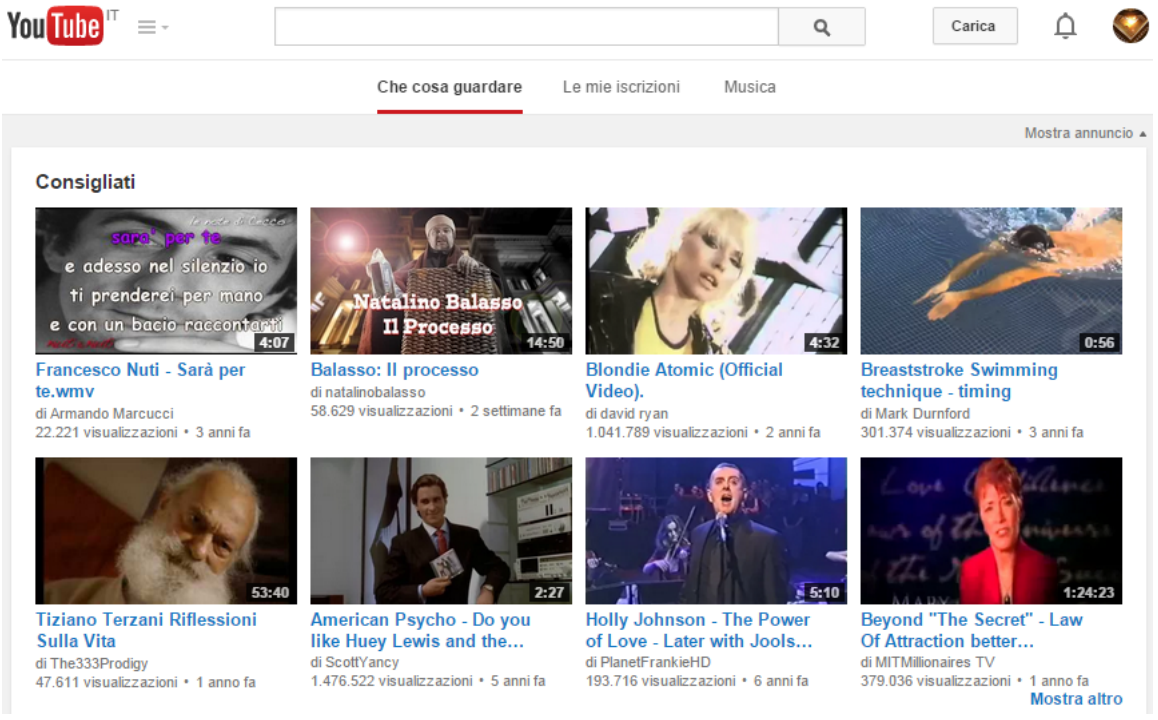

FIGURE 2.6: Example of Youtube recommendation system

2.8 *IMDb* Recommendation System

IDMb has been already introduced in 1.1. Just to complete the site profile, nowadays *IDMb* is an *Amazon.com* subsidiary since 1998[12] that Amazon use as an advertising resource for selling DVDs and videotapes.

This section is not intended to discuss IMDb recommendation system in deep: IDMb briefly describes its recommendation system on its FAQ page on the internet[13], and actually there's only a little literature on this subject as it is treated as a trade secret.

> *IMDb makes personalized recommendations to help you discover movies and TV shows that you will love.*
>
> **Where can I find my Watchlist and my ratings?**
> *Find your* Watchlist *using the* Your Watchlist *link in the top right of the nav bar. Find your ratings by using the* Your Ratings *menu item under the* Community *menu in the nav bar.*
>
> **How does IMDb choose personalized recommendations?**
> *First, we take all of the movies and TV shows that you've either rated or added to your* Watchlist. *Then, we compare your data to ratings made by other users. We can then find movies and TV shows that people with similar tastes to you. For each recommendation, you can see a list of the movies or TV shows upon which the recommendation was based. You have either rated these titles highly, or added them to your* Watchlist.

[12] IMDb was launched on the Internet in 1990
[13] http://www.imdb.com/help/show_leaf?personalrecommendations

How does IMDb know what I *showed interest in*?

When you give a movie a positive rating or add a movie to your Watchlist, *we track that as a movie that you are interested in.*

Figure 2.7 shows an example of IMDb recommending system.

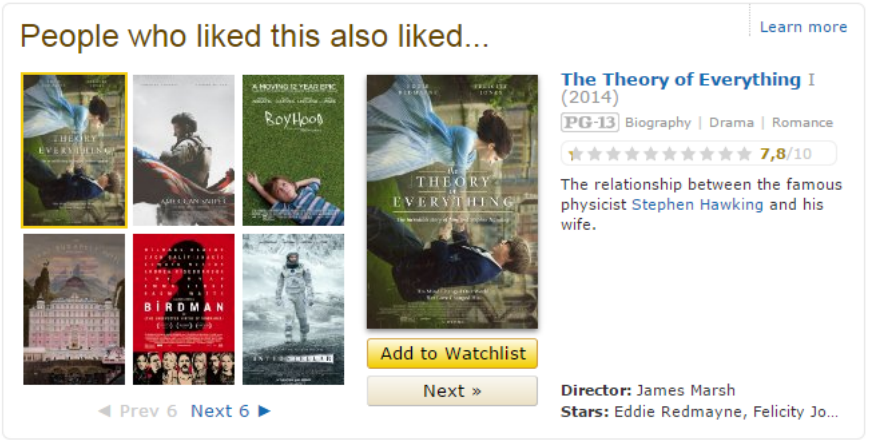

FIGURE 2.7: Example of IMDb recommendations

But this is not the only recommendation system provided by the site. In order to perform its commercial purpose IMDb propose the user a wide set of predetermined *recommended* lists, as shown in Figure 2.8, such as *Oscar Winners* listing, *Box Office* listing (Figure 2.9), or *Top250* that is a listing of the top rated 250 films of all-time, based on ratings given by the registered users (Figure 2.10).

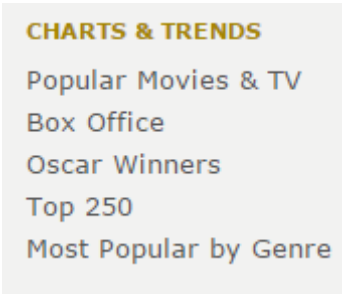

FIGURE 2.8: IDMb recommended listings

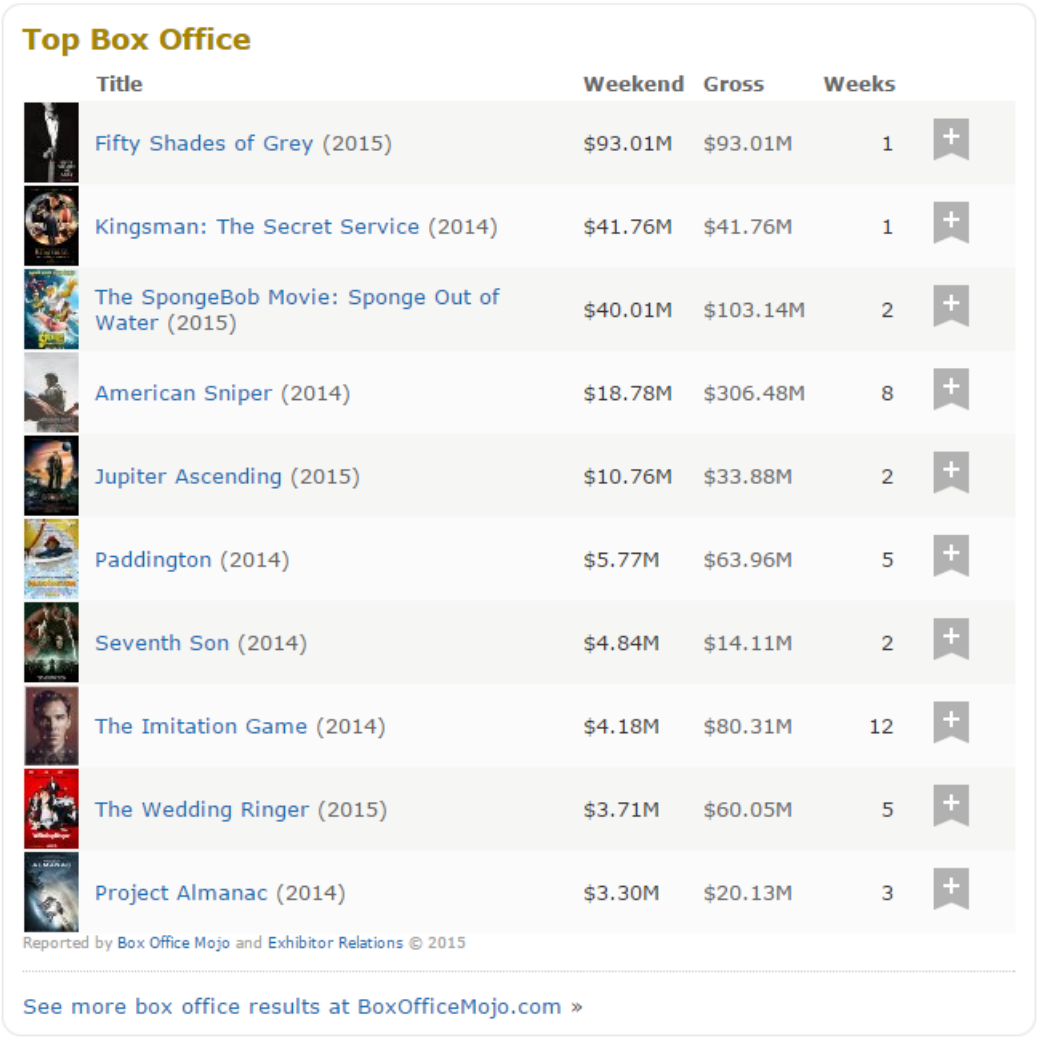

FIGURE 2.9: [IDMb Box Office listing

To balance Top250 listing, the user can also have access to a *Bottom100* listing, that is a listing with the worst (according by registered users) movies ever made, as shown in Figure 2.11.

In general, the concept of Top250 has been widely exploited giving access to the user to a generic *Top250 by genre.* as shown in Figure *a)* in Table 2.1)

IMDb provides also a wide set of browsing options (Figure *b)* in Table 2.1) in order to let users to achieve different kind of listing according to their desiderata.

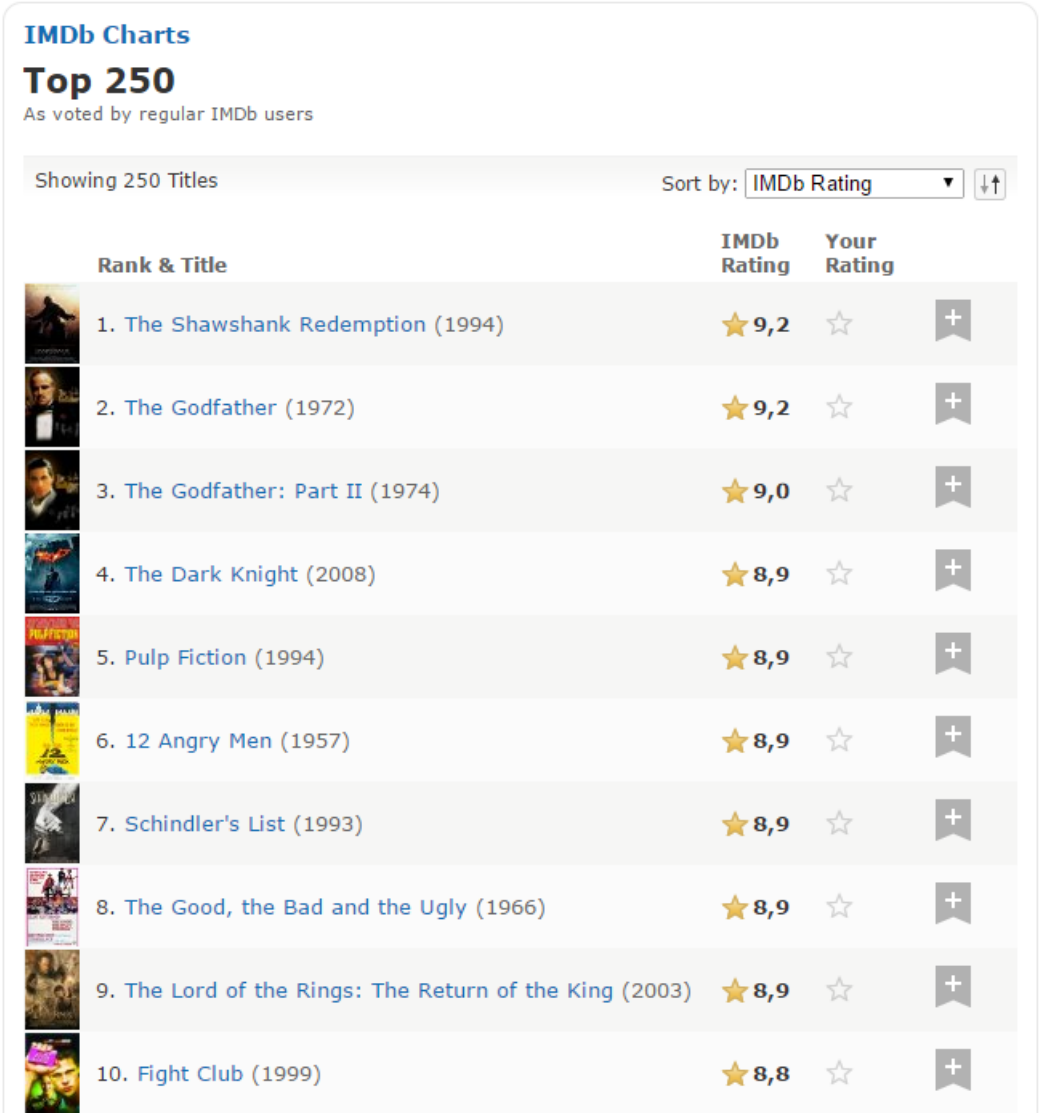

FIGURE 2.10: IDMb Top250 listing

More over there is a search engine by which the user can query the database in order to find those movies or actors he/she's looking for. The search engine can be activated by filling a *search-box*, as shown in Figure 2.12, to find a specific movie or a director or an actor/actress. If the user is intended to perform a more refined search, such as *all movies shot in 1960, in Milan, in french language*, or *all dramas and biographies rated at least 6.5* (whatever it means) *shot by* 20_th *Century Fox*,

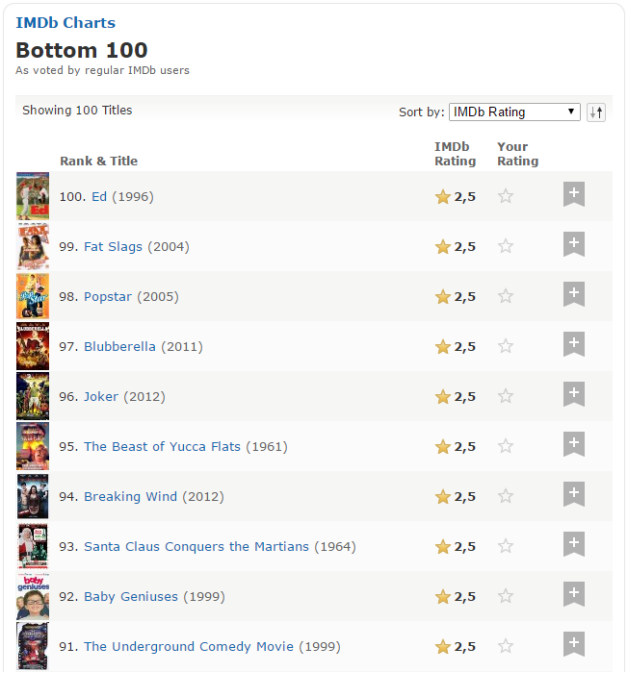

FIGURE 2.11: IMDb Bottom 100 listing

a) Top movies by genre b) IMDb browsing options

TABLE 2.1: IMDB listing

he/she can use the *advanced title search* shown in Figure 2.13, where in a guided mode the user can submit a complex query involving information of different kind.

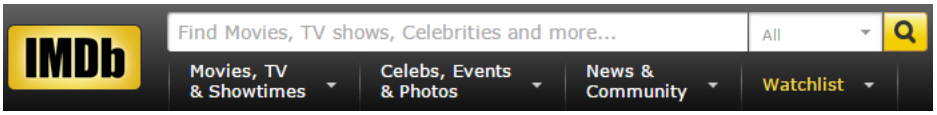

FIGURE 2.12: IMDb search box

As seen, there's a wide variety of recommendation systems and searching modes on the IMDb web site. What is missing, is the chance for a user to look for movies that talks about *ancient Greece and Persian's wars*, or even to look for a *funny movie with great word jokes in order to make my kids spend a nice afternoon with their friends*. In normal life it would natural asking a *hint* to a friend or an acquaintance. On the IMDb web site, or on, generally speaking, the Internet, it would be nice doing the same, but a text box where a user can insert his/her desiderata, hopefully in natural language is completely missing.

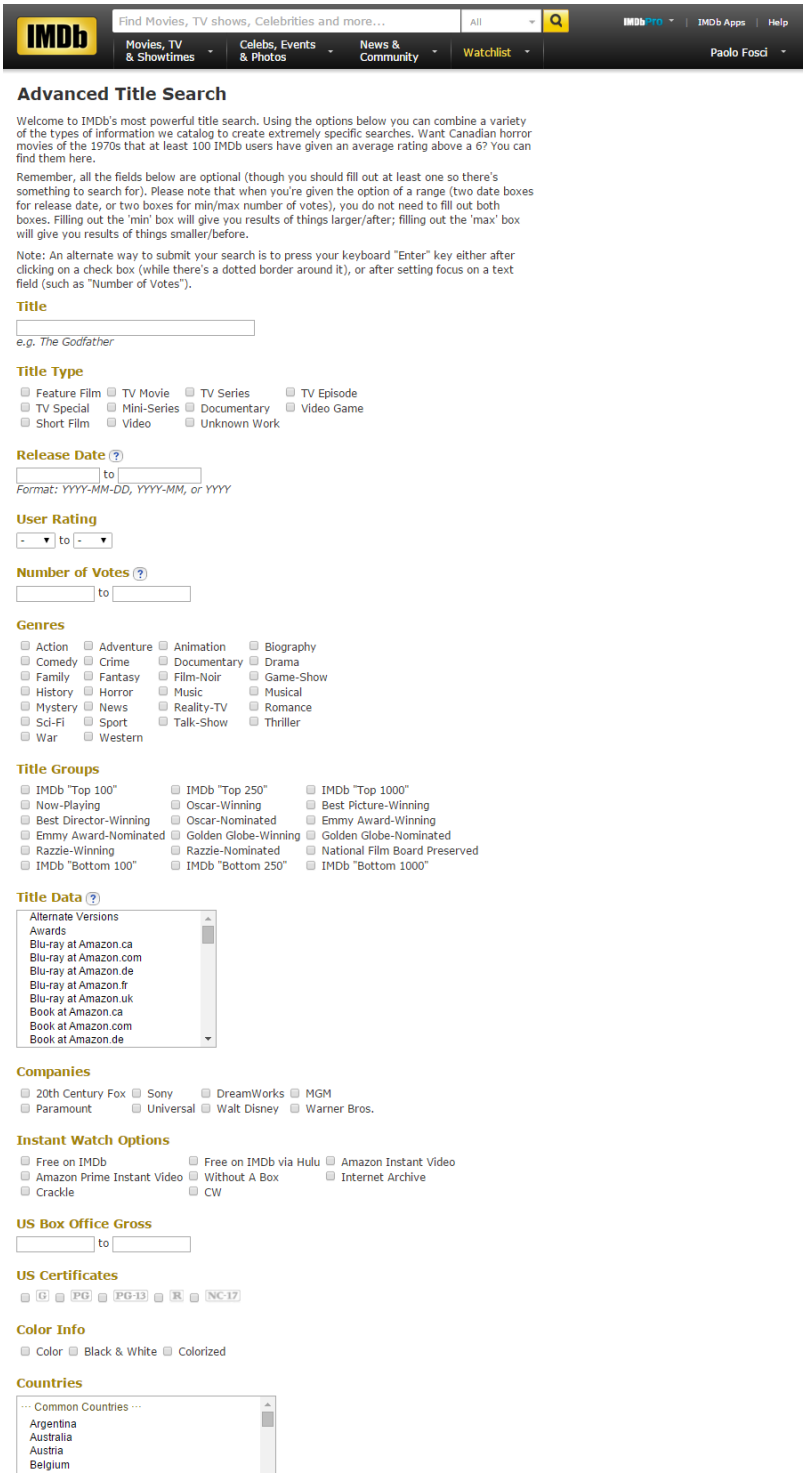

FIGURE 2.13: Part of IMDb advanced searching options

Chapter 3

Problem definition

This Chapter is to define the scope of this thesis illustrating the idea behind the design of the *HFC* system, and presenting the theoretical model developed to retrieve and rank products from reviews after a user submit a query expressed in natural language.
The Chapter will introduce first the basic idea behind the *HFC* ranking model and the origins where the model come from. Then the *HFC* ranking model will be explained in its details. To finish, the intermediate versions of the *HFC* ranking model will be briefly shown in order to understand how and why designing choices were made.

3.1 Basic idea

Section 1.2 reported examples of people needs in every day life. Modern web sites for sales, or booking, allow people to comment the product or service they get, and this information can be read by other users in order to make their choice.

Era una notte buia e tempestosa... during experiments related to a research about the comparison between (social) *network analysis* and *itemset mining techniques*, both applied to the analysis of a corpus of reviews about movies got from *Epinions.com* web site[1], where the same set of words in a group of reviews were considered as an *itemsets* (i.e. a *wordset*), it has been notice that the biggest itemsets mined in comments for a specific movie, often describe the movie quite well. The reason behind this behavior can be explained by the consideration that in reviews, where people often express a wide range of different concepts about the product, there are strong ideas about the product that tend to be common between people. It's very likely so, that different people tend to use the same set of words when talking about features that strongly characterize a product. From this point of view, an itemset can be imagined as a *carrier of meaning* (and after what is the purpose of words if not to carry meaning?), and the bigger the itemset the bigger the meaning carried.

Just as an example, comments about a movie like *300*[2] can report the following sentences among the others: *a good movie about ancient Greece and the Persian wars*, or an *it was interesting the historical part where king Leonidas fights against the Persians in the Termopilis battle in Greece* or even *I was quite sad when I saw the Greek king Leonidas dying in the battle killed by the Persians.*
In such sentences the words *Greece*, *battle*, *Persian* are quite common between comments. More over there are words like *wars* and *fights*, or *dying* and *killed* which have a very close meaning each other. Some words are quite close together, other instead are more distant, but in both cases they are present in the same comment, and these words characterize the movie.

[1] *Epinions.com* has been introduced in Chapter 1
[2] http://www.imdb.com/title/tt0416449

3.2 Ranking Model

The ranking model described in the following has been presented for the first time in a previous work [3] exposed during the **Dexa Conference** held in Prague in August 2013. During the conference has been also stated that the query engine of the *HFC* system can be assimilated to a *NoSQL database* system
The model has been also the core of the paper [4] presented during the *Medi Conference* held in Amantea (Italy) in September 2013, and also the core for the paper [5] presented during the 3_{rd} *International Workshop on Semantic Search over the Web (**SSW 2013**)* held in Riva del Garda (Italy) on August 2013 in conjunction with the 39_{th} *International Conference on Very Large Databases (**VLDB 2013**)*. In 2014 this model has been taken as an inspiration for the paper [6] presented during the ***SoCPaR*** conference held in Tunis in August.

The ranking model described in the following is the key component of the query engine of the *HFC* system. The purpose of the *HFC* system is to retrieve products and rank them according to a natural language sentence (the query) submitted by a user. The query engine extracts those products whose reviews are mostly relevant for the query. *Relevance* is evaluated by means of a *ranking metric*; retrieved products are returned as a list sorted in reverse order of relevance. Hereafter, it will be described how the ranking metric is defined.

3.2.1 Termsets

In this work, a query q is considered as a *set of terms* (or briefly, a *termset*). Thus, a query containing a number n of terms can be described as:

$$q = \{t_1, \ldots, t_n\}$$

Obviously will be investigated only those queries where $n > 1$ or, in other words, those queries q where $|q| > 1$.
With T is denoted a generic termset that is a subset of q for which applies $|T| > 1$. With D_q is denoted the set of termsets T derived from q. The reader should notice that the cardinality of D_q is:

$$|D_q| = 2^n - (n + 1) \tag{3.1}$$

i.e. D_q is the power set of q without the empty set and the n single terms that compose q.

The notation T_l denotes an l-termset of q, that is a termset composed by l terms, i.e. $|T_l| = l$. The notation $D_{q,l}$ denotes the set of l-termsets T_l. Notice that the cardinality of $D_{g,l}$ is:

$$|D_{q,l}| = \binom{n}{l} \tag{3.2}$$

3.2.2 Termset Weight

The concept of *weight* for a termset will be defined here.

Definition 1: The weight of a $l-$termset is a function of the length l of the $l-$termset and the length of the query q ($|q| = n$); the *weight* is denoted as $w_q(l)$. For $n = 2$, according to Formula 3.1 there is only one 2-termset and its weight by definition is:

$$w_q(2) = 1 \qquad\qquad n = 2 \tag{3.3}$$

For $n > 2$ instead,

$$w_q(l) = 0.5 \qquad l = n \tag{3.4}$$

$$w_q(l) = \frac{w_q(l+1)}{\binom{n}{l} + 1} \qquad 2 < l < n \tag{3.5}$$

$$w_q(l) = \frac{w_q(3)}{\binom{n}{2}} \qquad l = 2 \tag{3.6}$$

□

The rationale behind Definition 1 is the following. The topmost termset, corresponding to the whole query, is the most important one, as it is carrying the whole meaning of the termset, and its weight is equal to the overall weight of all the shorter termsets. The same principle is valid recursively for any generic termset T_l (with $2 < l < n$), whose weight is equal to the overall weight of all lower cardinality termsets (even those that are not subset of T_l). In this way, reducing the size of termsets, the contribution of each level of cardinality quickly decreases.

Figure 3.1 shows an example of a query containing the words *funny*, *great*, *hilarious* and *jokes*. For each admissible termset length l, all available termsets that is possible to derive from the query-terms are listed, and their number and weight is displayed.

Notice, that the overall weight of all termsets is exactly 1

$$\sum_{T \in D_q} w_q(|T|) = 1. \tag{3.7}$$

l	*number of* T_l	*weight*	*l-termsets* (T_l)
4	1	0.5000	{funny, great, hilarious, jokes}
3	4	0.1000	{funny, great, hilarious}
			{funny, great, jokes}
			{funny, hilarious, jokes}
			{great, hilarious, jokes}
2	6	0.0167	{funny, great} {funny, hilarious}
			{great, hilarious} {great, jokes}
			{funny, jokes} {hilarious, jokes}

FIGURE 3.1: Termsets for a query containing *great, funny, hilarious, jokes* and corresponding weights.

3.2.3 Query Expansion and Semantic Coefficient

Once the concept of termset weight has been defined, lets take a step back, and make focus on the query. As stated previously, a user must have the chance to submit his/her query in natural language. *This implies a lot of things.*
First of all, not all words (i.e. terms) have the same importance. Some words are full of meaning according to their position in the sentence, while in practice other words are completely useless (e.g. *conjunctions*). Moreover, words can be connected to other words, that are not necessarily in the query, because they all share the same meaning, or concept.
On the base of these considerations, and others more, a query must be handled thru several operations before the query-engine can be activated. In the following, these operations will be shown.

Pos-tagging. *Pos-tagging* is the operation to mark each word with its logical role in the sentence. As an example the phrase *I hope this thesis is not too long and boring* can be marked as $I_{pronoun}$ $hope_{verb}$ $this_{adverb}$ $thesis_{noun}$ is_{verb} $not_{negation}$ $too_{adjective}$ $long_{adjective}$ $and_{conjunction}$ $boring_{adjective}$.

In the *HFC* system this operation is performed by means of *Stanford Parser*[3] that tags (i.e. mark) each word in a user query with an attribute that denotes the word grammar role (*verb*, *noun*, *adjective* to name a few) in the query more or less in the way explained in the previous example.

Stopwords filtering. *Stopwords* are those words that are *too* common in texts (such as *articles*, *conjunctions* or common verbal forms like *is* or *have*). These words hold a very small semantic meaning, since their popularity is useless to connote the text, while instead they contribute to create noise. So after pos-tagging operation, *stopwords* are discarded from the query.
Stopwords can include also some very context-dependent words such as the word *actor* in a movie context, or the word *car* in an automotive context.

Thus, denoting with SW the set of possible stopwords, in the rest of the thesis the notation:

$$q = \{t_1, \ldots, t_n\}$$

includes only those terms $t_i \notin SW$, and, as stated in Section 3.2.1, only those queries q such that $|q| > 1$ (actual length without stopwords) will be considered.

Term expansion. *Term expansion* is the operation to link a word to similar ones. This operation can be performed according to different approaches:
the expansion can be realized by means of an *ontology* that, very briefly, thru a *stemming* operation can retrieve the root of a word and then associated the word to all the derived words from the root according to various relationships. This approach is strongly grammar and vocabulary guided. For example to the word *white* can be linked to its plural form *whites*, to its *synonym pale*, and also to its *hypernym color*.
The other main approach, is guided by semantic, and it's related to the concept of

[3]See Appendix A

entity recognition and *linked data*. By means of these concepts, in a sentence containing the sequence of words *Paul McCartney*, the famous singer can be recognized, and then it is possible to associate him to *The Beatles*, *John Lennon*[4] and *Liverpool* just to make a short list.
Both this approaches could be used in combination. Unfortunately, by the time this thesis has been written, only the *ontology* approach has been implemented in the *HFC* system, leaving the *linked data* approach to a future development.

In the *HFC* system term expansion is performed by means of *WordNet*[5] ontology. After pos-tagging, each tagged term $t_i \in q$ is expanded with all those terms directly associated to t_i based on the grammar tag given by the pos-tagger. The rationale is the following: especially in English language (that is at the moment the language on which the research is focused), the same word can assume different roles depending on its position in a sentence. As an example the word *blue* can be either an *adjective* or a *verb* or even a *noun*. Wordnet allows to link words according to various relationships. While all nouns, verbs and adjectives can have *synonyms*, only nouns can have *hyponyms*[6], and only verbs can have *troponyms*[7]. There are actually, a total of 15 possible different relationships, according to *Wordnet*, between a tagged term and its expanded words. If the system was left free to take all possible expansions, there would be two kinds of problems: first of all, too many words could be associated together making the search engine doing too much extra work[8]; second, words unrelated with the query could be associated to it creating noise and making the result of the query engine inaccurate.

[4] And hopefully not to *Yoko Ono* :p
[5] See Appendix B
[6] Y is a hyponym of X if every Y is a (kind of) X. The reverse relationship is *hypernymy*. As an example, *oak* is a hyponym of *tree*, and *tree* is the hypernym of *oak*.
[7] the verb Y is a troponym of the verb X if the activity Y is doing X in some manner. As an example, *to chat* is a troponym of *to talk*)
[8] Performance has been a great issue, and will be discussed in Chapter 6.

Pos-tagging operation is motivated precisely by these issues. By means of pos-tagging is possible to filter in only those word expansions that can be considered appropriate for the query.

More formally, t_i^* denotes the generic expanded term of t_i, and $ES(t_i)$ denotes the expansion set of t_i (i.e. the set of all expanded terms for term t_i). By definition, $t_i \in ES(t_i)$ with an *identity-term* relation link, thus

$$|ES(t_i)| \geq 1$$

Notice that, given a generic expanded term t^*, it can happen that t^* can stand at the same time in both the expansion set of two (or even more) different terms, that is, $t^* \in ES(t_i)$ and $t^* \in ES(t_j)$, with $i \neq j$. In other words, it's not possible to stand *a priori* that $ES(t_i) \cap ES(t_j) = \emptyset$ with $i \neq j$. As an example, the term *colour* can be an *hypernym* expansion for both terms *red* and *black*.

Query expansion. From the concept of *expanded term* can be easily derived the concept of *expanded query* as a set of *expanded terms*. Formally, an *expanded query* q^* is given by a combination of $(t_1^*, \ldots, t_n^*)$ where $q^* = \bigcup t_i^*$, where $t_i^* \in ES(t_i)$ and $1 \leq i \leq n$.
Since an expanded term can reside in different expansion sets, an expanded term t^* can be included more than once in q^*. In other words, it happens that $|q^*| \leq |q|$.
Those expanded queries q^* for which hold $|q^*| = |q|$ are considered *valid*.
Notice that the original query q is a particular q^* itself, and it is *valid* by definition.

Expanded Termsets. The previous considerations about a query q and its expansions q^*, can be applied to each termset T. Thus, given a termset $T = \{t_1, \ldots, t_l\}$,

T^* denotes an *expanded termset* derived from[9] T, and given by a combination of expanded terms $(t_1^*, \ldots, t_l^*)$ for which $T^* = \bigcup t_i^*$, where $t_i^* \in ES(t_i)$ and $1 \leq i \leq l$.
Still holds the relation $|T^*| \leq |T|$ and the following definition:

Definition 2: T^* is *valid* $\iff$ $|T^*| = |T|$.
□

Similarly, $ET(T)$ denotes the *expansion set of termset* T (i.e. the set of all possible expanded termset T^* that can be derived from T). The cardinality of ET(T) is

$$|ET(T)| = \prod_{t \in T} |ES(t)| \tag{3.8}$$

that is the number of all possible combinations of the expanded terms of those terms that compose T. Notice that T itself is part of $ET(T)$, that is $T \in ET(T)$.

In Definition 2 the *validity* of a termset has been addressed thru a path that, starting from a termset T, generates expanded terms and then wonder if a candidate termset T^* is *valid* or not. The implementation of *HFC* system taught that it is better to consider the reverse situation: starting from a candidate termset T^*, finding out the query terms that generate the expanded terms that compose T^* and finally controlling if T^* is *valid*.
Since has been stated that an expanded term t^* can be derived by different query terms t_i and t_j, that is, it can happen $ES(t_i) \cap ES(t_j) \neq \emptyset$, it is useful to introduce the concept of *reverse expansion set* $RES(t^*)$, that returns the subset of query terms t that can generate t^*. Still, it is easy to recognize that is not possible to stand *a priori* that $RES(t_i^*) \cap RES(t_j^*) = \emptyset$ with $i \neq j$.
Considering a termset $T^* = \{t_1^*, \ldots, t_l^*\}$, it is possible to get a set $QT(T^*)$ of termset T combining terms picked up from all reverse expansion sets $RES(t_j^*)$ of expanded

[9] A similar locution is: ... *generated thru term expansion by*...

terms $t_k^* \in T^*$. In a formal way, given $T^* = \{t_1^*, \ldots, t_l^*\}$, then $T = \bigcup t_i$ where $t_i^* \in RES(t_i^*)$ and $1 \leq i \leq l$ and holds the relation $|T| \leq |T^*|$.
From this point of view, the problem to validate a termset T^* becomes the problem to find at least one termset $T \in QT(T^*)$ for which $|T| = |T^*|$[10].

Finally, D_q^*, denotes the set of all valid expanded termsets that are included in q and all its valid expansions q^*. By definition $D_q \subseteq D_q^*$.

Semantic coefficient. After having introduced the concepts of *expanded termset* T^* and *expansion set of termset* $ET(T)$, it's necessary introducing the concept of *semantic coefficient*. The idea of semantic coefficient derives from the naive consideration that a termset T is a *carrier of meaning*. and every expanded termset T^* is a carrier of a similar meaning, even if it's not properly the same[11]. From the this point of view, the expansion set of a termset $ET(T)$ can be seen has a whole *meaning container*, where its *meaning* is given by the contribution of every expanded termset T^*. As the original termset $T \in ET(T)$ is natural to think that T contribution must be more relevant than the others T^*.

In a formal way semantic coefficient is defined as follow:
starting from a single term t, each expanded term $t^* \in ES(t)$ has a *semantic coefficient* $sc_t(t^*)$, with $0 < sc_t(t^*) \leq 1$, that depends on the cardinality of $ES(t)$.

Definition 3: For each $t^* \in ES(t)$

$$sc_t(t) = 0.5 + \frac{0.5}{|ES(t)|} \tag{3.9}$$

$$sc_t(t^*) = \frac{0.5}{|ES(t)|} \tag{3.10}$$

□

[10] Later on, in Section 4.2, there will be a further explanation.
[11] As an example, the meaning of *big tree* or *huge tree* is very close to *big oak*.

The rationale of semantic coefficient, is the following. A term describes a whole *meaning* that is mostly expressed by the term t itself, but receives a small contribution from expanded terms t^*: the greater the number of expansions, the smaller the semantic contribution of a single expanded term[12]. Notice that according the previous Definition:

$$\sum_{t^* \in ES(t)} sc_t(t^*) = 1 \tag{3.11}$$

Table 3.1 shows how sc_t varies with $ES(t)$ cardinality. The Table consider both the value of semantic coefficient for the original term t and for all its expanded terms t^*.

$\lvert ES(t)\rvert$	$sc_t(t^*)$ $(t^* = t)$	$sc_t(t^*)$ $(t^* \neq t)$
1	1.0000	—
2	0.7500	0.2500
3	0.6667	0.1667
4	0.6250	0.1250
5	0.6000	0.1000
10	0.5500	0.0500
20	0.5250	0.0250

TABLE 3.1: Term semantic coefficient w.r.t. the cardinality $|ES(t)|$.

Figure 3.2 shows the trends of semantic coefficient $sc_t(t)$ and $sc_t(t)$ according to $|ES(t)|$ variation[13]. From the graphic can be observed that $1 \leq sc_t(t) < 0$ and $0.25 \leq sc_t(t^*) \leq 0$.

From the term semantic coefficient descends the concept of *termset semantic coefficient*, as the semantic relevance of a generic termset T^*[14] that derives from the single term semantic coefficient of he particular combination of expanded term t_i^*

[12] Criterion of *expansion inflation*.

[13] Excessive values of $|ES(t)|$ are only hypothetical. In 99% $|ES(t)|$ of case is supposed to be less than 10 ($|ES(t)| < 10$)

[14] Remember that even a termset T can be considered as a special expanded termset T^*.

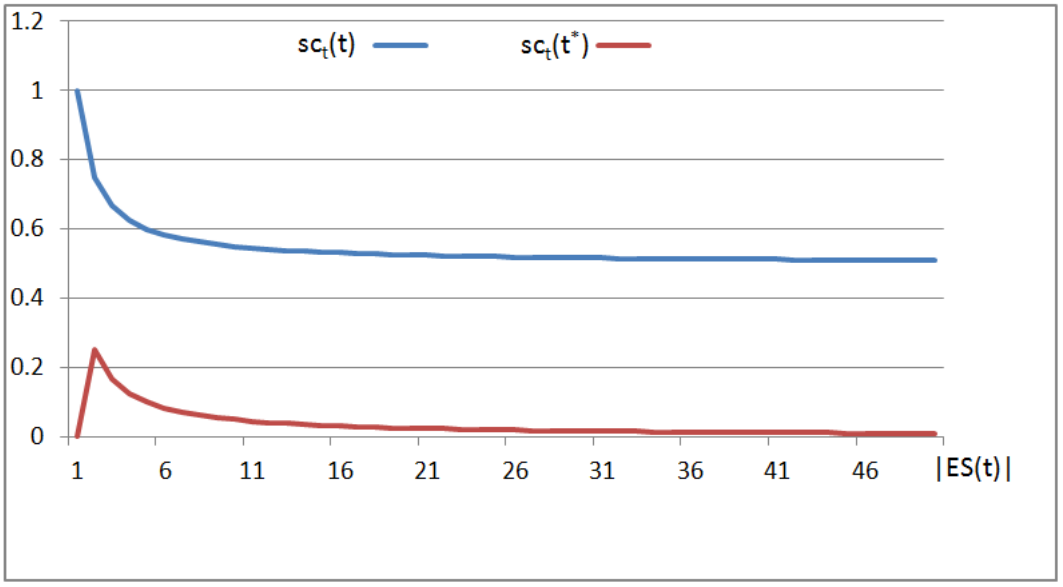

FIGURE 3.2: Term semantic coefficient trends

that compose T^*. Hence, $sc_T(T^*)$ denotes the semantic coefficient for an expanded termset T^* derived from T. $sc_T(T^*)$ is defined as follow:

Definition 4: Given an expanded termset $T^* = \{t_1^* \dots t_l^*\}$ derived for a termset $T = \{t_1 \dots t_l\}$, it is

$$sc_T(T^*) = \prod_{t_i^* \in T_l^*} sc_{t_i}(t_i^*) \tag{3.12}$$

□

That is, $sc_T(T^*)$ is given by the product of all $sc_t(t^*)$ of the terms t^* that compose a termset T^*. In this way, a termset that contains only original terms gives the highest semantic contribution, while augmenting the number of expanded terms in the termset, the semantic contribution decreases.

Table 3.2 gives an example of termset semantic coefficient according to the structure

of its composition. In the example, it is considered a 2-termset $T = \{t_1, t_2\}$. Term t_1 has two expansions, so that $|ES(t_1)| = 3$, and term t_2 has three expansions, so that $|ES(t_2)| = 4$. The number of possible combination for $T^* = \{t_1^*, t_2^*\}$ is 12. In the table is considered all the possible structure to compose a termset T^* according to a term is used in its original form, or in an expanded form. For each kind of structure is reported the number of T^* that satisfied it, and their semantic coefficient.

T^* (structure)	$\#T^*$	$sc_t(T^*)$
$\{t_1, t_2\}$	1	0.4167
$\{t_1, t_2^*\}$	3	0.0833
$\{t_1^*, t_2\}$	2	0.1041
$\{t_1^*, t_2^*\}$	6	0.0208

TABLE 3.2: Termset semantic coefficient
for a generic 2-termset $\{t_1^*, t_2^*\}$ where $|ES(t_1)| = 3$ and $|ES(t_2)| = 4$
.

Notice that, according to Definition 4

$$\sum_{T^* \in ET(T)} sc_T(T^*) = 1 \tag{3.13}$$

3.2.4 Product Reviews and Termsets

Consider now a product p (i.e. a movie, a camera, a hotel room) and its reviews in a web site as it has been addressed in Chapter 1.

The set of reviews of a product p is denoted by $R(p) = \{r_1, \ldots, r_k\}$. Each review r_i is a text, or in other words, a sequence of term occurrences $r_i =< t_1, \ldots, t_s >$, where terms can be repeated more than one time.

$ST(r_i)$ denotes the set of terms appearing in a review $r_i \in R(p)$, while $ST(R(p))$ denotes the set of terms appearing in all reviews for product p.
Thus, it is $ST(R(p)) = \bigcup ST(r_i)$, $\forall r_i \in R(p)$.

Now, consider again the query q and the set D_q of all termsets T derived from q.

Definition 5: A termset T is said *relevant* for product p if $\exists r_i | T \subseteq ST(r_i)$. $\square$
The set of relevant termsets T for product p is denoted as $RD_{p,q}$.

In an analogous way, remembering that D_q^* is the set of all expanded termsets T^* derived directly or by means of termset expansion from q.

Definition 6: A termset T^* is said *relevant* for product p if $\exists r_i | T^* \subseteq ST(r_i)$. $\square$
The set of relevant expanded termsets T^* for product p is denoted as $RD_{p,q}^*$.

Notice that $RD_{p,q} \subseteq D_q$, and also $RD_{p,q}^* \subseteq D_q^*$.

3.2.5 Termset Average Density

Very briefly, given a set of objects, in *Information Retrieval* the parameter *support* is defined as the fraction of objects, e.g. *documents*, that contain a determined item, e.g. *word* (see Section 2.5).

The situation addressed in this thesis is similar. In a preliminary work [1], it has been assumed that every termset occurrence in product reviews contributes to the *support* of the termset with the same weight, i.e. 1, since the *support*, by definition, is the number of reviews containing the termset on the total amount of reviews. In a formal way, given $R_T(p)$ as the subset of $R(p)$ (i.e. $R_T(p) \subseteq R(p)$) of those reviews containing termset T, support of termset T, denoted as $\sigma_p(T)$, is defined as:

$$\sigma_p(T) = \frac{|R_T(p)|}{R(p)} \tag{3.14}$$

After some tests, it has become evident that this assumption was not adequate, since every review containing the termset was considered in the same way, that is, every review had the same weight.

Instead, given a termset T, terms $t_i \in T$ can be very dense or very sparse in the single review. Under this consideration it can be easily guessed that a review in which the occurrences of terms t_i are dense is more relevant for the query than a review where occurrences are sparse. Thus, it has been introduced the concept of *Termset Density*, that is the density of terms $t_i \in T$ in a single review[15].

Definition 7: Consider a product p, a review $r \in R(p)$, and a termset T. The *Termset Review Density* $d_r(T)$ is defined as:

$$d_r(T) = \frac{|T|}{minWin_r(T)} \tag{3.15}$$

where $minWin_r(T)$ is the size of the minimal window in review r that includes all the terms $t_i \in T$.

□

Notice thus, for *Termset Review Density*, it holds that $0 < d(T_l, r) \leq 1$.

The next step is to define a *Termset Average Density* for a generic termset T w.r.t. a product p.

Definition 8: Consider a product p and its set of reviews $R(p)$. Consider also a termset T and the set $R_T(p)$ of reviews containing T. The *Termset Average Density* for product p, denoted as $ad_p(T)$, is defined as:

$$ad_p(T) = \sum_{r \in R_T(p)} \frac{d_r(T)}{|R(p)|} \tag{3.16}$$

□

The Termset Average Density is analogous to termset support, with the difference that the contribution to *support* of each review is not 1 anymore, but its density

[15] In this moment, there's no need to distinguish between termset T and expanded termset T^*

$d_r(T)$[16].

Notice that $ad_p(T) \leq s_p(T) \leq 1$.

Termset Average Density becomes equal to support when density of each review containing the termset T is 1, that is, each review contains the terms $t_i \in T$ all together in a window with no extra terms in between.

3.2.6 Product Ranking Metric

Once every piece of the ranking model, *termset weight*, *semantic coefficient* and *average density* has been defined is finally possible to define the *Product Ranking Metric* ***PRM***, by which it is possible to rank products on the base of its reviews after a user query q.

Definition 9: Consider a query q, the set of termsets[17] D_q^* derived from q, the weights $w_q(|T^*|)$ and the semantic coefficients $sc_q(T^*)$ for each termset $T^* \in D_q^*$. Consider also a product p, the set of reviews $R(p)$ and the set of relevant termsets $RD_{p,q}^*$ that can be actually extracted from $R(p)$. Given, for each $T^* \in RD_{p,q}^*$, the average termset density $ad_p(T^*)$, the *Product Ranking Metric* for product p is defined as

$$\mathbf{PRM}_q(p) = \sum_{T^* \in RD_{p,q}^*} (w_q(|T^*|) \times ad_p(T^*) \times sc_q(T^*)) \tag{3.17}$$

□

The rationale of the above definition is the following. For each termset T^* derived from the query q and actually relevant in the reviews, its contribution to the overall

[16]Indeed Formula 3.14 can be written in different way as $s_p(T) = \sum_{r \in R_T(p)} \frac{1}{|R(p)|}$

[17]From now on, when there is no need to distinguish from a termset T, directly derived from q, and a termset T^*, derived by termset expansion, the adjective *expanded* will be omitted not to burden the text.

ranking value is given by its weight $w_q(|T^*|)$ (that depends solely on its size) multiplied by its semantic coefficient $sc_q(T^*)$ (that depends solely on its structure) and by its *average density* $ad_{p,q}(T^*)$ (that depends solely on its impact in reviews).

The system of weights and semantic coefficients has been designed to obtain a $PRM_q(p) = 1$ for an *ideal* set of reviews for product p, where each review contains every expanded termset T^* that can be derived from q with a density $d_r(T^*) = 1$, and every expanded termset T^* is *valid*.
In other words, $PRM_q(p) = 1$ can be obtained when a user asks for something that every reviewer said in the same way of the user.

3.3 Model Evolution

Once the evaluation model has been described it can be interesting to understand its genesis.
In Section 3.1 it has been told that the first inspiration came from the comparison between social network analysis and itemset mining techniques. Under this boost, a primarily idea of ranking model started to grow, even if not in the definitive form as it has been exposed in Section 3.2. Starting from the first draft, the ranking model has been refined more and more times till reaching its final form. Each main step of this model evolution has been the subject for a conference paper. Not every paper presented has been accepted to the conference due to several motivation. Sometimes the rejection was caused due to an actual weakness of the specific model presented that didn't while the worth to be investigated anymore.
In the followings will be introduced the ranking models (only the ranking model, as the papers were more comprehensive) presented in two different accepted paper exposed in as many conferences.

3.3.1 DATA 2012 - Ranking model

The following is the first version of the ranking model of *HFC* system that has been exposed in a preliminary work [1] presented during the *Data Conference* held in Rome in July 2012.
The text is copied verbatim so it may happen that the notation is different from the one exposed before in Section 3.2.
Notation apart, the main differences are concerning the weighting system, the complete absence of the concept of semantic coefficient, and the usage of termset support instead of average density.
Not the least, two different ranking metric called **PRV** and **APRV** are introduced.

> *Our approach to product retrieval is based on the widely exploited notion of* itemset mining. *Originally [10], itemset mining is the basic and computationally difficult step for association rule mining. Here, we are not interested in association rule mining, but only in itemset mining for analyzing sets of words that frequently occur together in reviews. In this Paragraph, we first present the basic notions of itemset mining. Then, we adapt the concept to our context.*
>
> **Basic Notions on Itemset Mining**
>
> *The notion of* itemset mining *was introduced as a fundamental part of the process for mining association rules [10, 16]. In a* transaction *database* $Z = \{z_1, \ldots, z_n\}$*, a transaction* z *is a set of items* $z = \{i_1, \ldots, i_k\}$ *sold together in transaction* z*.*
>
> *A* $k-$*itemset* $I = \{i_1, \ldots, i_k\}$ *is a set of* k *items. The support of* I*, denoted as* $s(I)$ *is the frequency with which the itemset occurs in the transaction database, i.e.,* $s(I) = \frac{|Z(I)|}{|Z|}$ *(where* $Z(I) \subseteq Z$ *is the set of*

transactions $z \in Z$ such that $z \cap I = I$, i.e., the transactions that contain itemset I).

Given a minimum support threshold $\overline{s}$*, an itemset I is said* large *if its support is no less than the threshold $\overline{s}$, i.e., $s(I) \geq \overline{s}$.*

The problem known as itemset mining *is the problem of extracting large itemsets from the transaction database Z, provided that a minimum threshold for itemset support is defined.*

Notice that this problem has been widely and extensively studied in the last two decades. As far as the main results about efficient algorithms are concerned, we just cite the fundamental paper [10], in which the Apriori *algorithm was defined, that has enabled itemset mining on large databases.*

Product Reviews and Itemsets

Consider now a product p (a movie, a camera, etc.) and a set of reviews $R(p) = \{r_1, \ldots, r_k\}$, where each review is a text, i.e., a sequence of term occurrences $r_i =< t_1, \ldots, t_s >$.

A set of reviews for a product p can be seen as a transaction database, where each review $r_i \in R(p)$ can be seen as a transaction, and terms t_j corresponds to items. Given a minimum threshold for support, a large $k-$itemset $I = \{t_1, \ldots, t_k\}$ summarizes a relevant number of reviews in $R(p)$. Consequently, $R(p)$ can be represented (and summarized) by means of the set of large itemsets $IS(R(p))$ extracted from within reviews, and composed of terms in reviews.

Moving from the general idea described above, we now define the way to decline the concept for our context.

Definition 10: Consider the set of terms $T(R(p)) = \{t\}$ appearing in reviews $r \in R(p)$ and the set SW of stopwords. Consider also a minimum

support threshold for single terms $\overline{s}_t \in (0, 1)$.
The set $\overline{T}(R(p)) \subseteq T(R(p))$ is the set of *Relevant Terms*, such that $\forall t \in \overline{T}(R(p))$, it is $s(t) \geq \overline{s}_t$, $s(t) \times |R(p)| \geq 3$ and $t \notin SW$.
□

In other words, we consider relevant a term if it is not a stopword and it is reasonably frequent in the reviews. Notice that the set SW of stopwords is prior knowledge for our approach.
Notice that condition $s(t) \times |R(p)| \geq 3$ discards terms that are present in less than 3 reviews, even though their support is greater than $\overline{s}_t$; in fact, terms that appear in only one single review are sporadic and not sufficiently mediated by the community to characterize the product (typical situation with a small number of reviews for a product).

Definition 11: Consider the set $\overline{T}(R(p))$ of *relevant words*, and a minimum support threshold for compound itemsets $\overline{s}_c$.
A $k-$itemset I with length $k \geq 2$ is said *relevant* if $I \cap \overline{T}(R(p)) = I$ and $s(I) \geq \overline{s}_c$.
The set of relevant itemsets for product reviews $R(p)$ is denoted as $\overline{IS}(R(p))$.
□

In other words, we focus on itemsets composed only of relevant words whose support is greater than a minimum support threshold $\overline{s}_c$ that is possibly different w.r.t. the minimum support threshold $\overline{s}_t$ used for determining relevant terms.

The reason why we consider two different minimum support thresholds for single terms and compound itemsets is the following. Terms

must be relevant alone, thus they must be sufficiently representative of community opinions. But compound itemsets may be rarer than single terms in reviews, so a different minimum support threshold for compound itemsets is considered.[18]

Therefore, we can state that given two minimum support thresholds $\overline{s}_t$ *and* $\overline{s}_c$ *for, respectively, single terms and compound itemsets, a set* $R(p)$ *of product reviews for a product* p *is summarized by the set of single relevant terms* $\overline{T}(R(p))$ *an by the set of relevant compound itemsets* $\overline{IS}(R(p))$.

Retrieval Model

The key element of our proposal is the Retrieval Model, *since it is used to rank products in order to let most significant products to emerge, i.e., the products whose reviews better match the query terms.*

Consider a query $q = \{t_1, \ldots, t_n\}$ *containing a number of terms* $n \geq 1$. *The query* q *itself is a* $n-$*itemset.*

With $I_l(q)$ *we denote a* $l-$*itemset composed by* l *terms in* q *(it is* $1 \leq l \leq n$*). We denote with* D_q *the set of all itemsets that can be obtained with terms in* q*, and with* $D_q(l)$ *the subset of itemsets of length* l*; notice that cardinality of* $D_q(l)$ *is* $|D_q(l)| = \binom{n}{l}$ *and notice that* $|D| = 2^n - 1$*, i.e.,* D *is the power set of* q *without the empty itemset.*

In order to define a rank for a product based on its reviews, we define the concept of weight *for an itemset.*

[18] In principle, we could think about different minimum thresholds for compound $k-$itemsets that somehow depend on itemset length k. We will consider this aspect in our future work.

Definition 12: The weight of a $l-$itemset is denoted as $w_q(l)$. The weight of the single $n-$itemset q is, by definition, $w_q(n) = 1$, while is

$$w_q(l) = \frac{w_q(l+1)}{\binom{n}{l}} \qquad for \quad 1 \leq l < n.$$

□

The rationale behind Definition 12 is the following. The topmost itemset, corresponding to the whole query, is the most important one, and it gives the full contribution to rank the product. In contrast, lower levels must contribute a little, but not as much as the topmost itemset; in particular, each non-topmost level contributes as one of the itemset in the upper level.

Notice, that this way, the overall weight of itemsets of level $(n-1)$ *is* 1*, exactly as the single topmost itemset; reducing the size of itemsets, the contribution of each level quickly decreases.*

Example 1: To illustrate, consider the following example query over a movie reviews collection: *great funny hilarious jokes*. Based on the previous definitions, the weights for itemsets are determined. In particular, the 4-*itemset* that coincides with the whole query is assigned weight 1. For each of the four 3-*itemsets*, the assigned weight is

$$w_q(3) = \frac{w_q(4)}{\binom{4}{3}} = \frac{1}{4} = 0.2500.$$

#	*weight*	*l*	*itemsets*
1	1.0000	4	{funny,great,hilarious,jokes}
4	0.2500	3	{funny,great,hilarious} {funny,great,jokes} {funny,hilarious, jokes} {great,hilarious, jokes}
6	0.0417	2	{funny,great} {funny,hilarious} {funny,jokes} {great,hilarious} {great,jokes} {hilarious, jokes}
4	0.0104	1	{funny} {great} {hilarious} {jokes}

FIGURE 3.3: Itemsets levels for query *great funny hilarious jokes* and corresponding weights.

For each 2-*itemset*, the weight is

$$w_q(2) = \frac{w_q(3)}{\binom{4}{2}} = \frac{0.2500}{6} = 0.0417$$

the weight at this level dramatically decreases, so that the presence of only two terms together in the review gives a very little contribution to the overall product ranking.

Finally, 1-*itemsets* are assigned with weight

$$w_q(1) = \frac{w_q(2)}{\binom{4}{1}} = \frac{0.0417}{4} = 0.0104$$

. Notice that even though single terms are less than 2-*itemsets*, their contribution to the overall ranking is very low, in order to compensate the possible very high supports of single terms.

Figure 3.3 graphically illustrates the itemsets levels.

□

We are now ready to define the ***Product Relevance Value (PRV)****.*

Definition 13: Consider a product p, the set of reviews $R(p)$ and the set of relevant terms and compound itemsets $F(p) = \overline{T}(R(p)) \cup \overline{IS}(p)$

that it is possible to extract from $R(p)$. Consider now a query q and the set of itemsets D_q included in q.

If we denote with $\overline{D}_q(p) = F(p) \cap D_q$ the set of itemsets included in q that can be actually extracted from within reviews of product p, the *Product Relevance Value (PRV)* for product p is defined as

$$\mathbf{PRV}_q(p) = \sum_{I \in \overline{D}_q(p)} (w_q(|I|) \times s(I)) \tag{3.18}$$

□

The rationale of the above definition is the following. For each itemset included in the query q and actually relevant in the reviews, its contribution to the overall relevance value is its support value multiplied by its weight. Notice that the longer the matched itemset, the stronger its contribution; the shorter the matched itemset, the smaller its contribution.

However, the definition of PRV *does not take the number of reviews for each product into account. For this reason, we introduce the* ***Adaptive Product Relevance Value (APRV)****.*

Definition 14: Consider a product p, the set of reviews $R(p)$, the query q and the *Product Relevance Value* $PRV_q(p)$. The *Adaptive Product Relevance Value APRV* is defined as

$$\mathbf{APRV}_q(p) = PRV_q(p) \times log_{10}(|R(p)|) \tag{3.19}$$

□

The rationale behind this variation of the relevance measure is that the same itemset I with a given support $s(I)$ is more effective in summarizing

> *a product with a large number of reviews than a product with a very small number of reviews. The logarithm allows us to avoid that products with very large sets of reviews excessively dominate the other ones.*
>
> *Consequently, based on this concepts, the goal of the retrieval task is to provide a ranked list of products, based either on the* Product Relevance Value *or on the* Adaptive Product Relevance Value.

The ranking model was applied to a small dataset of movie reviews downloaded from Epinions.com web site (the dataset is described later on in Section 6.1.2). After the presentation at the conference, the audience raised remarks regarding the smallness of the dataset to make the attempt relevant as there were no benchmark to compare the results, and regarding also the ranking model asking how much it was different from a *Vector Space Model* application (see Section 2.1).

3.3.2 KES 2012 - Ranking Model

This version of the ranking model of *HFC* system has been exposed in a previous paper [2] presented during the *KES Conference* held in San Sebastian (Spain) in September 2012.

As in the model illustrated in the section before, the text is copied verbatim so it may happen that the notation is different from the one exposed in Section 3.2.

Notation apart, the main differences are concerning the evolution of the weighting system, but still is not the same adopted in Section 3.2; again, the complete absence of the concept of semantic coefficient; the introduction of average density instead of termset support, but its usage is slightly different.

The ranking model has been applied to a dataset of people's blogs. Each blog was composed by several posts. So in this case the word *blog* is used instead of *product*,

and similarly, the word *post* is used for *review*.
Not the least, the ranking metric is called **BRV**.

We now present the Retrieval Model. We got inspiration by the concept of itemset mining [10], adapted to our context, using the concept of Termset *in place of* Itemset*.*

Posts and Termsets

Consider now a blog b and a set of posts $P(b) = \{p_1, \ldots, p_k\}$*, where each post is a text, i.e., a sequence of term occurrences* $p_i =< t_1, \ldots, t_s >$*.*

A termset I *is a set of terms* $I = \{t_1, \ldots, t_k\}$*. The support of a termset* I *w.r.t. blog b is the number of posts* $p_i \in P(b)$ *in which* I *occurs* $(I \subseteq p_i)$ *on the total number of posts, i.e.,*

$$s(I, b) = |\{p_i \in P(b) | I \subseteq p_i\}| / |P(b)|$$

A single term t *is the simplest form of termset, i.e.,* $I = \{t\}$*. Given a minimum threshold for support* $\overline{s}$*, a large* $k-$*termset* $I = \{t_1, \ldots, t_k\}$ *summarizes a relevant number of posts in* $P(b)$*. Consequently,* $P(b)$ *can be represented (and summarized) by means of the set of large termsets* $IS(P(b))$ *extracted from within posts, and composed of terms in posts.*

Moving from the general idea above described, we now define the way we decline the concept for our context.

Definition 15: Consider the set of terms $T(P(b)) = \{t_j\}$ appearing in posts $p \in P(b)$ and the set SW of stopwords. Consider also a minimum support threshold $\overline{s} \in (0, 1)$.
The set $\overline{T}(P(b)) \subseteq T(P(b))$ is the set of *Relevant Terms*, such that

$\forall t \in \overline{T}(P(b))$, it is $s(t,b) \geq \overline{s}$, $s(t,b) \times |P(b)| \geq 3$ and $t \notin SW$.

□

In other words, we consider relevant a term if it is not a stopword and is reasonably frequent in the posts. Notice that the set SW of stopwords is prior knowledge for our approach.
Also notice that condition $s(t,b) \times |P(b)| \geq 3$ discards terms that are present in less than 3 posts, even though their support is greater than $\overline{s}$; in fact, terms that appear in only one single post are sporadic and not sufficiently mediated by the community to characterize the blog (typical situation with a small number of posts for a blog).

Definition 16: Consider the set $\overline{T}(P(b))$ of *relevant words*, and the minimum support threshold $\overline{s}$.
A $k-$termset I with length $k \geq 2$ is said *relevant* if $I \cap \overline{T}(P(b)) = I$ and $s(I,b) \geq \overline{s}$. The set of relevant termsets for blog posts $P(b)$ is denoted as $\overline{IS}(P(b))$.
With $F(P(b)) = \overline{T}(P(b)) \cup \overline{IS}(b)$, we consider the full set of compound termsets $\overline{IS}(P(b))$ extended with the set $\overline{T}(P(b))$ of *relevant words* that are actually 1-*termsets* (i.e. termset composed by a single term).

□

In other words, we focus on termsets composed only of relevant terms whose support is greater than a minimum support threshold $\overline{s}$.[19]

[19] In principle, it could be possible to consider two different minimum support thresholds for single terms and compound termsets, based on the following reason. Terms must be relevant alone, thus they must be sufficiently representative of community opinions. But compound termsets may be rarer than single terms in posts, so a different and possibly lower minimum support threshold for compound termsets might be considered.

Therefore, we can state that given a minimum support threshold $\overline{s}$ *for termsets, a set* $P(b)$ *of posts for a blog* b *is summarized by the set of relevant termsets* $F(P(b))$.

Blog Relevance Measure

Consider a query $q = \{t_1, \ldots, t_n\}$ *containing a number of terms* $n \geq 1$. *The query* q *itself is a* $n-$*termset.*

With $I_l(q)$ *we denote a* $l-$*termset composed by* l *terms in* q *(it is* $1 \leq l \leq n$*). With* D_q *, we denote the set of all termsets that can be obtained with terms in* q*, and with* $D_q(l)$ *the subset of termsets of length* l*; notice that the cardinality of* $D_q(l)$ *is* $|D_q(l)| = \binom{n}{l}$ *and notice that* $|D_q| = 2^n - 1$*, i.e.,* D_q *is the power set of* q *without the empty termset.*

In order to define a rank for a blog based on its posts, we define the concept of weight *for a termset.*

Definition 17: The weight of a $l-$termset is denoted as $w_q(l)$. The weight of the single $n-$termset q is, by definition, $w_q(n) = 0.5$, while for $1 < l < n$ it is $w_q(l) = \frac{w_q(l+1)}{\binom{n}{l}+1}$ and for $l = 1$ it is $w_q(1) = \frac{w_q(2)}{\binom{n}{1}}$.
□

The rationale behind Definition 17 is the following. The topmost termset, corresponding to the whole query, is the most important one, and its weight is equal to the overall weight of all the shorter termsets. The same principle is valid for any generic termset I_l *(with* $1 < l < n$*), whose weight is equal to the overall weight of all lower levels termsets (even those that are not subset of* I_l*). In this way, reducing the size of termsets, the contribution of each level quickly decreases.*

Notice, that the overall weight of all termsets is exactly 1.

Example 2: To illustrate, consider the following example query over a collection of blogs: *best music Saturday party*. Based on the previous definitions, the weights for termsets are determined. In particular, for the unique 4-*termset*, that coincides with the whole query, the assigned weight is 0.5. For each of the four 3-*termsets*, the assigned weight is

$$w_q(3) = \frac{w_q(4)}{\binom{4}{3} + 1} = \frac{0.5}{4+1} = 0.1$$

For each of the six 2-*termset*, the weight is

$$w_q(2) = \frac{w_q(3)}{\binom{4}{2} + 1} = \frac{0.1}{6+1} = 0.0143$$

the weight at this level dramatically decreases, so that the presence of only two terms together in the post gives a very little contribution to the overall blog ranking.
Finally, the four 1-*termsets* are assigned with weight

$$w_q(1) = \frac{w_q(2)}{\binom{4}{1}} = \frac{0.0143}{4} = 0.0036$$

. Notice that even though single terms are less than 2-*termsets*, their contribution to the overall ranking is very low, in order to compensate the possible very high supports of single terms.
□

Definition 18: Consider a blog b, a post $p \in P(b)$, and a termset I composed by $|I|$ terms. The *Termset Post Density* $d(I, p)$ is defined as

$$d(I, p) = \frac{|I|}{minWin(I, p)} \tag{3.20}$$

where $minWin(I, p)$ is the size of the minimal window in post p that includes all the terms in termset I.
□

Notice that for Termset Post Density*, it holds that* $0 < d(I, p) \leq 1$*. Also notice that, by definition,* Termset Post Density *for a* $1-$*termset* I_1 *is always* $d(I_1, p) = 1$*.*

We define also the concept of relevance *for a termset.*

Definition 19: Consider a blog b, its set of posts $P(b)$ and a query q ($|q| \geq 1$). With $P_I(b)$ we call the subset of $P(b)$ of those posts containing termset I. The *Relevance* of a termset I for a blog b is denoted as $rel_b(I)$ and is defined as follows:

$$rel_b(I) = |q|^{(|I|-|q|)} \times \frac{\sum_{p \in P(b)} d(I, p)}{|P(b)|} \tag{3.21}$$

Relevance is a specific termset feature that summarizes the importance of a termset w.r.t. to a set of posts.
□

The reader can easily guess that the longer the termset, the more sparse the occurrences of the termset in reviews. So, there is the risk that the relevance value of significant termsets, i.e, the long ones, becomes too low w.r.t. the relevance value of short termsets, that are more likely to be dense in posts. For this reason, we inserted the term $|q|^{(|I|-|q|)}$*, in order to boost long termsets and compensate the above discussed effect.*

We are now ready to define the ***Blog Relevance Value (BRV)****.*

Definition 20: Consider a blog b, the set of posts $P(b)$ and the set of relevant termsets $F(P(b))$ that is possible to extract from $P(b)$. Consider now a query q and the set of termsets D_q included in q.

If we denote with $\overline{D}_q(b) = F(P(b)) \cap D_q$ the set of termsets included in q that can be actually extracted from posts of blog b, the *Blog Relevance Value* for blog b is defined as

$$\mathbf{BRV}_q(b) = log_{10}(|P(b)|) \times \sum_{I \in \overline{D}_q(b)} (w_q(|I|) \times rel_b(I)) \tag{3.22}$$

□

The rationale of the above definition is the following. For each termset I included in the query q and actually relevant in the posts (i.e., with support greater than or equal to the minimum threshold $\overline{s}$), its contribution to the overall relevance value is given by its weight $w_q(|I|)$ (that depends on its length) multiplied by its relevance *measure $rel_b(I)$.*

The $\log_{10}(|P(b)|)$ factor is a boosting factor to rank better those blogs with a larger number of posts, among the ones with the same $\overline{D}_q(b)$ set of termset, in case also rel_b values are the same. The logarithm allows us to avoid that blogs with very large sets of posts excessively dominate the other ones.

As stated before, this version of the ranking model was applied to a blog dataset, described later on in Section 6.1.1. After the presentation at the conference, the audience was wondering how it was possible to verify the quality of the results unless to read every single blog, since, except for a few rough blog attributes, it was not possible to distinguish a blog from another. From considerations emerged afterwards, raised the idea of semantic coefficient.

Chapter 4

Algorithms

The research for reaching the best performance has been the main challenge of the project, and the quest for performance will be the center of a long dissertation in Chapter 6. In order to achieve the best performance, all components of the *HFC* system had to be designed optimally. Particular attention has been given to the implementation of the most exploited algorithms.

This Chapter will be addressed to describe the three most exploited algorithms:

- Recursive termset mining
- Termset valid query-coverage
- Termset minimum window

4.1 Recursive Termset Mining algorithm - RTM

In Chapter 3 has been shown that *termsets* are the key elements to determine the ranking value *PRV* of a product according to a user *query*. As explained in

Chapter 3, termsets are mined starting from the occurrences, in product reviews, of the searching terms derived from the analysis of the query. The searching terms constitute the so-called *term-base*, a set that is composed by the terms that are in the query[1], and also by those terms that derive from the query thru the *term-expansion* operation performed by means of *Wordnet* as explained in Section 3.2.3.
Once the *term-base* has been determined, all occurrences of terms[2] are retrieved from the storage system[3], and kept in central memory.
In central memory, for each product, there is an instance of the *term-base*. Linked to each term of the *term-base*, there is the list of *reviews*[4] in which occurrences has been retrieved[5].
From this starting point, for each product for which were retrieved terms occurrences, the algorithm for **R**ecursive **T**ermsets **M**ining (*RTM* in the following) is activated.

The *RTM* algorithm is a recursive algorithm that take inspiration from the *FP-Growth* algorithm [12], already mentioned in Section 2.5.

In a few words, for each product review, the strategy of the *RTM* algorithm is to consider the *term-base* $\{t_1, \ldots, t_n\}$ as a sequence $< t_1, \ldots, t_n >$ ordered according to a given criterion[6]. Thus, a k-termset T is denoted by an ordered list $< \bar{t_1}, \ldots, \bar{t_k} >$[7] of k terms taken from the *term-base*. In order to find every *(k+1)*-termset that share the same k terms, the algorithm tries to enrich the k-termset with those terms in the *term-base* that follow the $\bar{k}$*-th* term (of the k-termset) according to the sorting criterion. As soon as a *(k+1)*-termset is mined, the algorithm immediately uses

[1]Stopwords are not included.
[2]From now on, the word *term* will be used to point out only terms in *term-base*.
[3] Every architecture detail is afterwards demanded to and Chapter 5.
[4]Actually it is only the *review ID*.
[5]Linked to each review (ID), there is the list of position of term the occurrences retrieved. For *RTM* algorithm this information is not useful.
[6]The specific criterion does not matter. It's important there's an sorting criterion so that can be stated that $t_i < t_j \ \forall i < j$, that is, the term t_i precedes the term t_j when $i < j$.
[7]Be aware that $\bar{t_i} \neq t_i$, but as terms $\bar{t_i}$ in termset T are taken from the *term-base*, the ordering property $\bar{t_i} < \bar{t_j} \ \forall i < j$ is still valid.

the *(k+1)*-termset in order to recursively mine *(k+2)*-termsets. The recursion stops after the attempt to enrich the k-termset with the n-th term of the *term-base*.

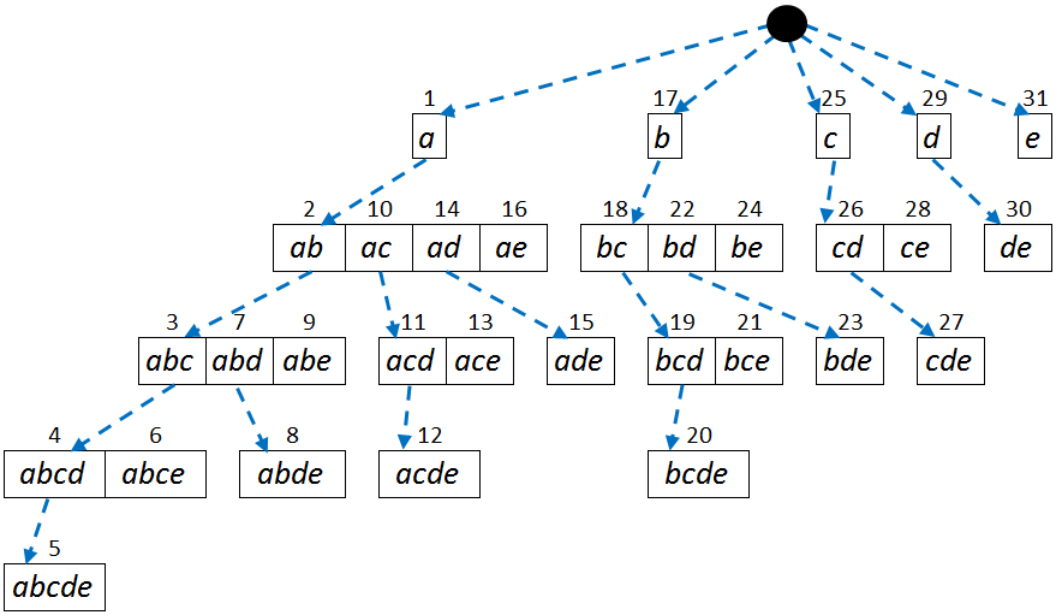

FIGURE 4.1: Termset mining order for a *term-base* $< a, b, c, d, e >$.
Numbers over termsets show the mining order.
Arrows indicate algorithm recursive calls.

Since terms have a sorting order, it should be clear that a *(k+1)*-termset can be derived only from the sole k-termset that represents a prefix for it. The reader should notice also that a k-termset containing the last (n-th) term of the *term-base* cannot be further enriched. Figure 4.1 shows the dependency relationship between k-termsets and *(k+1)*-termsets. The Figure 4.1 also shows all the possible recursive calls and the mining order in the case that every possible termset can be mined.

For *RTM* algorithm a termset is a list of terms $< t_1, \ldots, t_n >$ that is linked to the list of product reviews in which the termset can be retrieved. So, when the algorithm tries to enrich a k-termset with a *(k+1)*-th term, controls that there is at least a

review in which the candidate *(k+1)*-termset can be retrieved. After that, the *k+1*-termset must be *valid* according to Definition 2 of Chapter 3.
In this way, the *RTM* algorithm found all *valid* termset in reviews of a product p.

The *RTM* algorithm is shown in Figure 4.2.

```
a.  termBase: listOf⟨Term⟩
b.  termsetSet: setOf⟨Termset⟩
c.  termsetSet = RTMalgorithm (termBase, ∅, 0)

1.  function RTMalgorithm (termBase: listOf⟨Term⟩, kt: Termset, start)
                                  : setOf⟨Termset⟩
      begin
2.      t: Termset                  // candidate (k+1)-termset
3.      tSet: setOf⟨Termset⟩ = ∅
4.      for [i=start; i < ‖termBase‖; i++] do
5.        if [cover(kt) ⋂ cover(termBase(i)) ≠ ∅] then
6.          t = kt ⋃ termBase(i)
7.          if [validCoverage (t)] then
8.            tSet = tSet ⋃ t
9.            tSet = tSet ⋃ RTMalgorithm (termBase, t, i + 1)
            endIf
          endIf
        endFor
10.     return tSet
    end
```

FIGURE 4.2: Recursive Termsets Mining algorithm

In details:

- Line *a* represents the *term-base*. It is given after query analysis and term expansion by Wordnet.
- Line *b* defines the variable set where the mined termsets are kept.
- Line *c* shows the first activation of *RTM* algorithm.

- **Function** *RTMalgorithm* receives the *term-base* (parameter *termBase*), a *k*-termset (parameter *kt*) as base to mine *(k+1)*-termsets, the index[8] (parameter *start*) of the first term in the *term-base* useful to enrich the *k*-termset. The function returns the set of all termsets mined from the *kt* termset in input parameters. The reader should notice that the first activation of the algorithm performed in Line *b* reports as parameters the *term-base* as result of the expansion by *Wordnet*, an empty termset[9], and the index of the first term in the *term-base* from which start the mining.

- Two local variable are defined. Variable *t* at Line *2* represents the candidate *(k+1)*-termset. Variable *tSet* at Line *3* represent the local set where termsets recursively derived from *kt* are kept.

- The **for** loop at Line *4* is responsible for the generation of all *(k+1)*-termsets that can be derived from the *k*-termset *kt*. In detail, the **for** loop scans the *term-base* from the term following the k-th term in *kt*, denoted by the index *start*, till the end of the the *term-base*, since these are the only terms of the *term-base* that are able to enrich the termset *kt*.

- Inside the **for** loop, the **if** statement at Line *5* checks if a new *(k+1)*-termset can be mined. The mining condition is that the *(k+1)*-termset has a *support*, that is, there is a non-empty intersection between the *cover*[10] of the termset *kt* and the *cover* of the current term taken from the *term-base* given by the **for** loop. The function *cover* returns, indeed, the *cover* of termset or single term, that is the index sets of reviews in which can be retrieved the termset or the term. A non-empty intersection between the two *cover* functions means there

[8] It is used the *Java* convention by which indexes start from 0.

[9] The *RTM* algorithm consider *empty*-termset and 1-termset as actual termsets. Improperly, since by definition a termset has at least two elements. After the algorithm has finish its execution these improper termsets are filtered out.

[10] The concept of *cover* has been introduced in Section 2.5, and in a few words, is the list of reviews that contain a term or a termset.

is at least one review of the current product that contains the new *(k+1)*-termset: when this case is verified a new *(k+1)*-termset is build and assigned to the variable t in Line *6*.

- The **if** statement at Line *7* deals with the issue about *valid* termsets explained in Definition 2 of Chapter 3. The function *validCoverage* is a boolean function that indicates if a termset is *valid* or not. The operation of the function *validCoverage*[11] is explained in the next Section 4.2 since there is a special algorithm in this case as well.

  When the termset t is valid the function *validCoverage* returns a *true* value, and thus in Line *8* the termset t is inserted into the local set $tSet$ of termsets mined. After this operation, in Line *9*, the local set $tSet$ is update with all the termsets that can be mined from t by means of a *recursive call* of function *RTMalgorithm*. The parameters for function *RTMalgorithm* are the *term-base* *termBase*, the current *(k+1)*-termset t just mined, and the index of the term in the *term-base* following the term that allow to mine the current termset t. If the current term in the *term-base* is already the last one, the next recursive call terminates since the condition to enter in the **for** loop at Line *4* is not valid anymore.

- Finally, Line *10* returns overall set of termsets generated from the termset kt.

This algorithm works completely in main memory and, exploiting the order showed in Figure 4.1, is really fast[12] as treats only those termsets that could be effectively mined, and discards in advance all those termsets that cannot be surely mined as derived from a discarded termset (because it has *no support* or because it is *not-valid*). Figure 4.3 shows a possible execution of the algorithm over a *term-base* composed

[11] Please, be aware that the word *coverage* deals with *termset validity* and has a different context respect to the word *cover*, that deals with the list of reviews that hold a terms or termset.

[12] Details about execution time performance and the number of termset mined will be shown in Chapter 6

by 5 elements $< a, b, c, d, e >$. The Figure shows the mining order of *valid* termsets, and blue arrows shows the actual algorithm recursive calls performed.

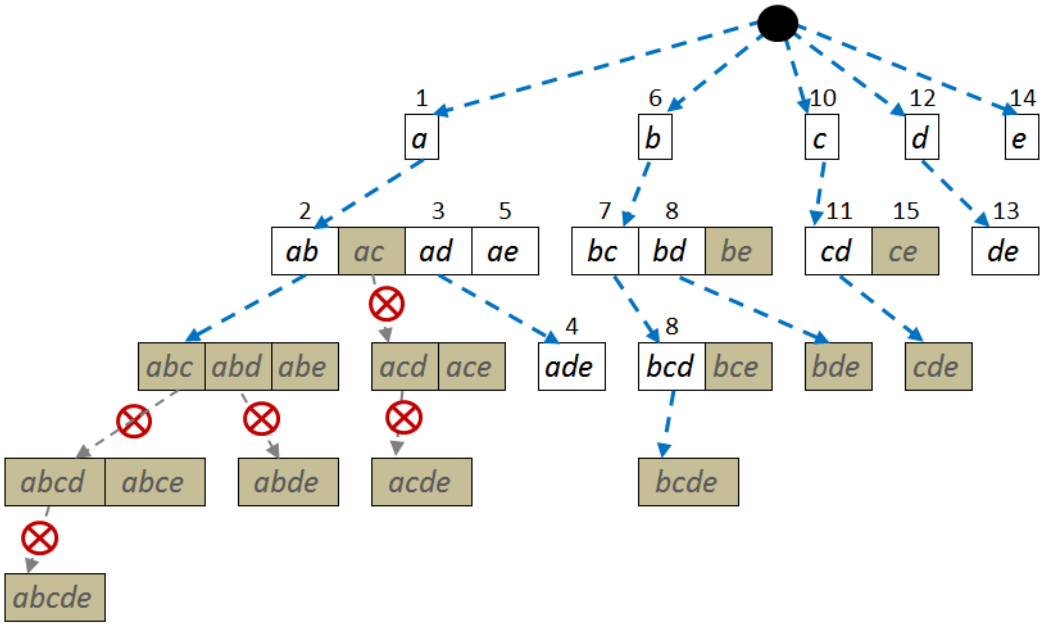

FIGURE 4.3: Termset mining dynamics for a *term-base* $< a, b, c, d, e >$. Numbers over termsets show the mining order. Mined termsets has a white background and only blue arrows indicate algorithm effective recursive calls performed.

Figure 4.4 shows the equivalent *Hasse Diagram*[13] of the example in Figure 4.3. All *non-mined* termsets are highlighted with a dark layer.

Notice that, if a termset t_1 is not mined, every termset t_2 for which $t_1 \subset t_2$ cannot be mined as well. In other words, every subset of mined termset is a mined termset and, dually, no superset of a non-mined termset is a mined termset!

[13] Already seen without decoration in Figure 2.2 in Chapter 2. In the *Hasse Diagram*, termsets are considered as actual *sets*. The connection link between two different termset t_1 and t_2 exists if it is $t_1 \subset t_2$.

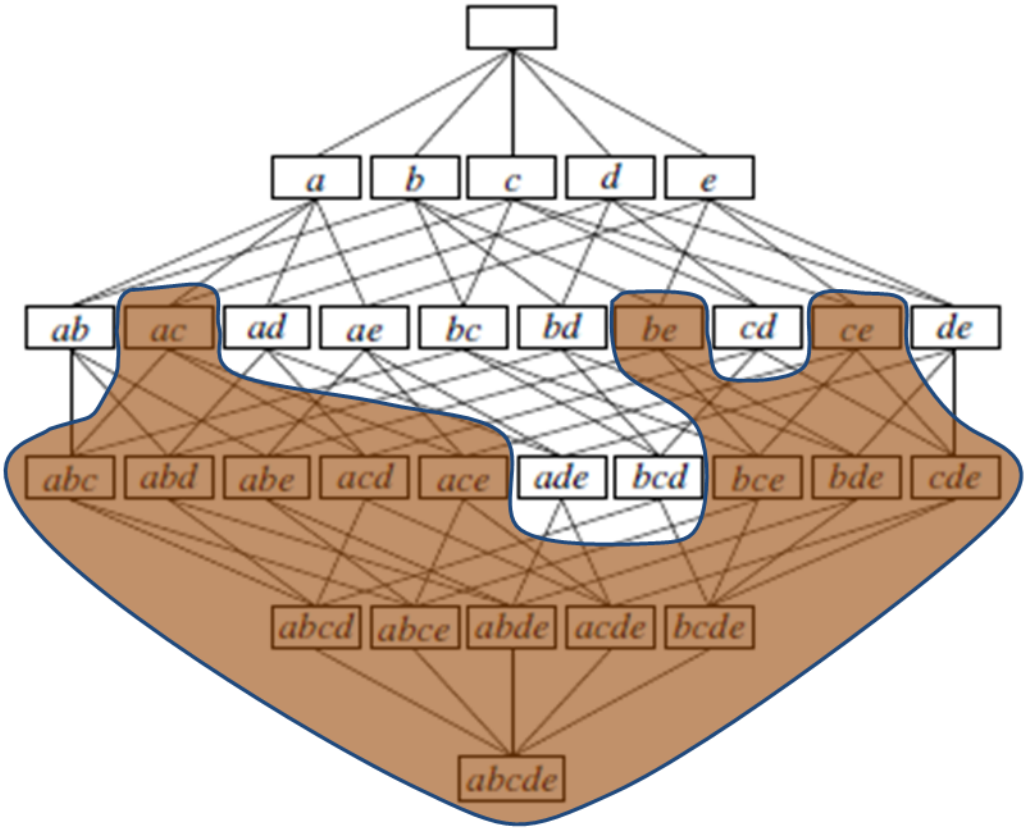

FIGURE 4.4: Hasse diagram showing valid termsets.
Notice that every subset of a mined termset is a mined termset (e.g. termset *ade*), or, dually, no superset of a discarded termset can be mined (e.g. termset *ac*).

4.2 Valid Termset Coverage algorithm - VTC

In Section 3.2.3 were introduced the concepts of *valid* termset and termset *semantic coefficient*. In particular, was told that the *HFC* system consider only *valid* termset and that the termset *semantic coefficient* depends of its composing structure.

In the previous Section 4.1, talking about the *RTM* algorithm, it has been stated that there is the need to have a performing algorithm to evaluate *valid* termsets.

Figures 4.5, Figures 4.6, Figure 4.7 and Figure 4.8 briefly summarize the situation.

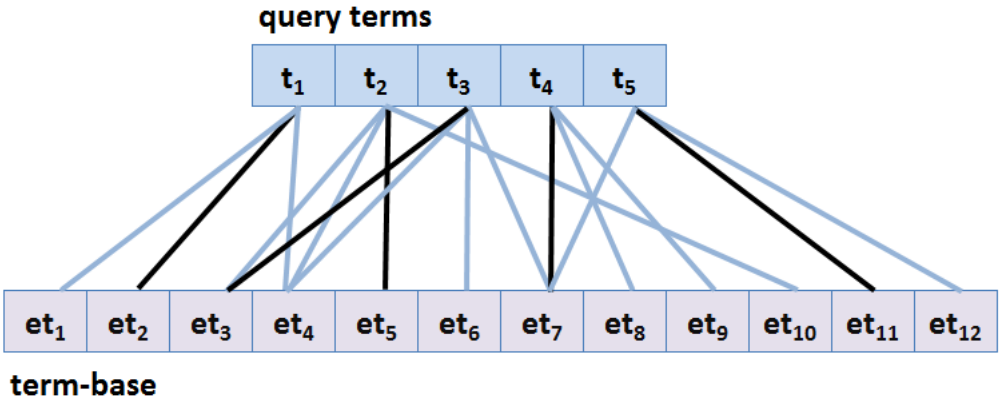

FIGURE 4.5: Graphic representation of query expansion.
Black links show *identity-term* links.

Figure 4.5 shows a query q and its set of terms $\{t_1, \ldots t_5\}$, and the *term-base* of expanded terms derived from q and its set of (expanded) terms $\{et_1, \ldots et_{12}\}$. A link between a generic couple (t_i, et_j) denotes an expansion from query term t_i to expanded term et_j. The *black* color of the link denotes an *identity-term* link, that is, the query term t_i and expanded term et_j are the same term $(t_i = et_j)$.

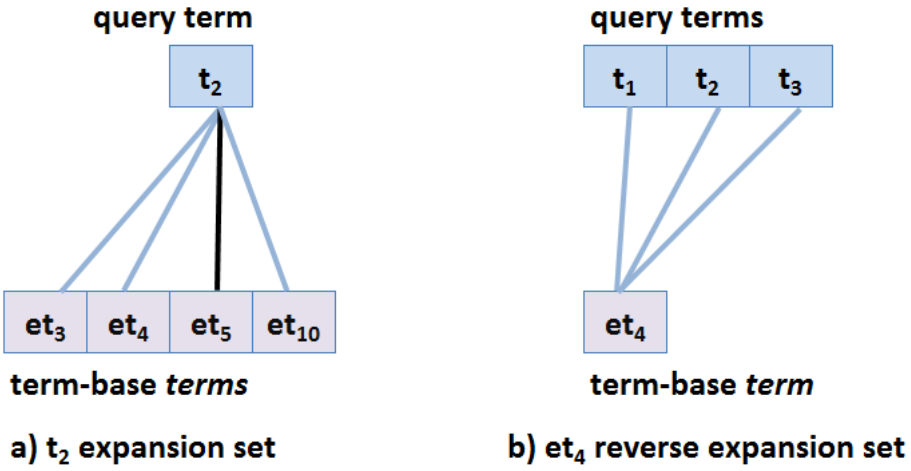

FIGURE 4.6: Expansion set Vs Reverse Expansion Set

Figure 4.6.a) focuses on a query term t and its *expansion set* $ES(t)$ as a subset of

the *term-base*. Figure 4.6.b) focuses on an expanded term *et* in the *term-base* and its *reverse expansion set* $RES(et)$ as a subset of query terms t_k that generate *et*.

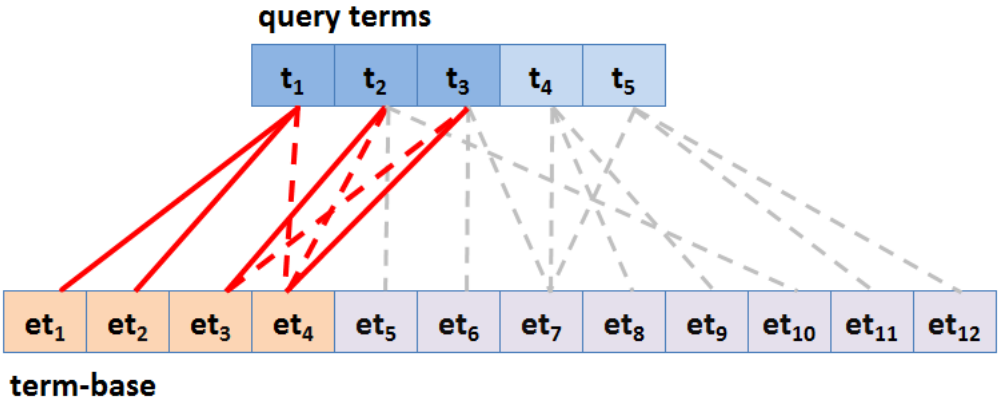

FIGURE 4.7: Non-valid termset graph.
Termset $\{et_1, et_2, et_3, et_4\}$ has no valid coverage over the query.

Figure 4.7 shows a *non-valid* termset example.
The candidate termset T^* is $\{et_1, et_2, et_3, et_4\}$ and the *red* links highlight the subset of query terms that can generate T^*. Continuous-line *red* links represent a specific non-valid *coverage* $T = \{t_1, t_2, t_3\}$ of T^* over the query since $|T| < |T^*|$, while *red* dashed line links are discarded (for this specific coverage). As expanded terms et_1 and et_2 can only be derived from the sole query term t_1, it becomes evident that there is no possible *valid* coverage T in order to get $|T| = |T^*|$.

On the other side, Figure 4.8 shows an example of *valid* termset.
The candidate termset T^* is $\{et_1, et_3, et_4, et_8\}$ and the *blue* links highlight the subset of query terms that can generate T^*. *Blue* continuous line links represent a specific valid *coverage* $T = \{t_1, t_2, t_3, t_4\}$ of T^* over the query since $|T| = |T^*|$, while *blue* dashed line links are discarded (for this specific coverage).

Analyzing the graph in Figure 4.8, it can be noticed that exchanging links (et_3, t_2) and (et_4, t_3), with, respectively links (et_3, t_3) and (et_4, t_2), there is another valid coverage of termset T^* over the query. This affects the calculus of the *semantic*

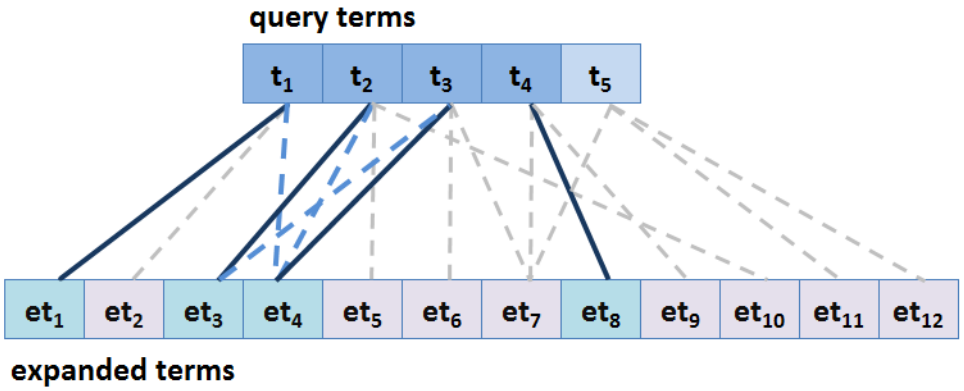

FIGURE 4.8: Valid termset graph.
Termset $\{et_1, et_3, et_4, et_8\}$ has at least one valid coverage over the query.

coefficient $sc_T(T^*)$[14]. Figure 4.9 illustrates the situation showing the two different *coverages*. In detail, coverage *b)* uses a more valuable *identity-term* link[15] (the black link) that coverage *a)* does not use. Thus, for this reason, the difference of semantic coefficient between coverage *a)* and coverage *b)*.

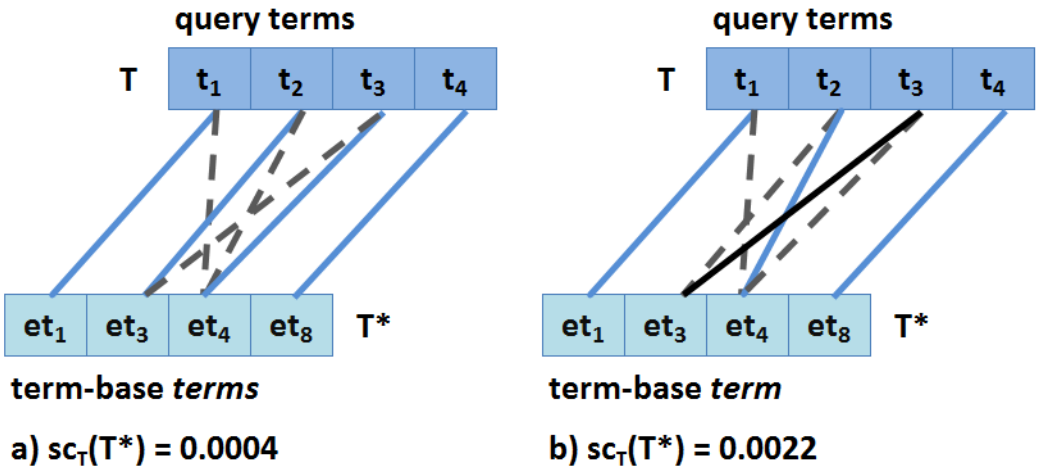

FIGURE 4.9: Valid coverages and semantic coefficient.
Coverage in **b)** uses the *identity-term* link (et_3, t_3) that is not used for coverage **a)**, and this change the semantic coefficient.

[14] See Definition 4 in Chapter 3.
[15] According to the definition of *term semantic coefficient* in Definition 3 of Chapter 3 *identity-term* links worth more than a simple expansion link.

The case analyzed in Figure 4.9, shows that verifying the validity of a candidate termset T^*, is not sufficient. Since the mission of the *HFC* system is, somehow, to research the meaning of the user query if product reviews, it would be natural to consider a termset at its highest level of significance. Thus, once has been verified that a termset is *valid*, it is necessary to further investigate in order to find out that termset *coverage* that brings the highest semantic coefficient. To reach this goal, every possible possible *coverage* must be verified.

With these premises, it is necessary to have an efficient algorithm to find out all valid *coverages* for a termset and calculate the semantic coefficient for each *coverage* found.
In order to achieve this goal, a recursive algorithm, named **V**alid **T**ermset **C**overage (VTC), has been developed. The VTC algorithm is shown in Figure 4.10.
In a few words, the algorithm start to built *coverages* adding recursively terms from *reverse expansions sets* of expanded terms (et) that compose the termset. Starting from an empty set as a *coverage*, the algorithm consider an expanded term $et_i \in termset$, and pick from $RES(et_i)$ a term t up. If t raises *coverage* cardinality (i.e. $t \notin coverage$) it is added to the *coverage* and a recursive call tries to enrich the coverage considering a different term $et_j \in termset$. When the recursive call returns, a new term $t \in RES(et_i)$ is tried to enrich the starting *coverage*, until all terms in $RES(et_i)$ have been used. When a valid *coverage* has been found, its semantic coefficient is calculated, and the termset is being marked as *valid*. The algorithm continues running until all possible *coverages* has been explored. If the termset is *valid*, the highest semantic coefficient calculated is assigned to the termset.

In details the VTC algorithm in Figure 4.10 works this way:

- Line *a* The VTC algorithm is activated by the RTC algorithm at *Line 7* of Figure 4.2, thru a *validCoverage* function, to verify if the *termset* given as parameter is *valid*.

```
a.  function validCoverage (termset: Termset) : boolean,
b.    t.isValid = false
c.    t.sc = 0                          // Semantic Coefficient
d.    recursiveEvalution (t, ∅, 0, 1)
e.    return t.isValid
    end

1.  function recursiveEvaluation (termset: Termset, coverage: setOf⟨Term⟩,
                                  i, sc) : void
2.    et_i, t : Term
3.    if [index < |termset|] then
4.      et_i = termset.getTerm(i)
5.      for each [t ∈ RES(et_i)]
6.        if [t ∉ coverage] then
7.          newCoverage = coverage ⋃ t
8.          newSc = sc * getSc(t, et))
9.          validCoverage (termset, newCoverage, i + 1, newSc)
          endIf
        endFor
10.   else
11.     termset.isValid = true
12.     t.sc = max(t.sc, sc)
      endIf
    end
```

FIGURE 4.10: Valid Termset Coverage algorithm

To make the notation quick, the *Termset* data-type is treated similarly to a *Java* object, with one attribute for semantic coefficient (sc), one boolean attribute for termset validity ($isValid$), and representing the termset as a $listOf\langle Term\rangle$.

- Line b sets the termset as *non-valid*.
- Line c sets the termset semantic coefficient to 0.

- Line *d* activates the recursive *VTC* algorithm.
  Initial parameters are the *termset* to be validate, an empty-set as initial *coverage*, the index of the first term $et \in termset$ to be used to enrich the coverage[16], and 1 as starting intermediate value of semantic coefficient[17]

- Line *e.* returns if the given termset is *valid* or not.

- **Function** *recursiveEvalutation* at Line *1* performs the *VTC* algorithm.
  Parameters are the *termset* to be validated, a starting *coverage* as a set of terms, the index i of the term et_i to use for enriching the *coverage*, the current intermediate semantic value *sc* to be updated. The function has no return value as information is stored directly into the *termset* object.

- Two local variables have been declared in Line *2*: et_i is the current term of the termset; t a generic query term that is $t \in RES(et_i)$ effectively used to enrich the *coverage*.

- Line *3* performs the validity test. When the condition is $false$ all terms $et \in termset$ has been used to enrich the *coverage*, that is, $|coverage| = |termset|$ and so the termset is *valid*, semantic coefficient can be calculated, and the recursion returns back.
  When the condition is $true$ it means there are still $et \in termset$ to use to enrich the *coverage*.

- Line *4* retrieves the current et_i from termset.

- The **for** loop at line *5* is responsible to use every term $t \in RES(et_i)$ try to enrich the *coverage*.

[16] Termset is treated as list, and indexes follow *Java* convention starting from 0.

[17] Term's semantic coefficients sc_i are always $0 < sc_i \leq 1$, and termset semantic coefficient is give by the product of all $sc_i \in termset$, so 1 as starting value is invariant.

- Test at Line *6* verifies if the current term t can effectively enrich the *coverage*. If the test is $false$, no further exploration with current *coverage* and t is necessary and the recursion takes a step back. If the test is $true$ a further recursion call will be performed.

- Line *7* builds a new coverage for the recursion by adding t to the current *coverage*.

- Line *8* updates intermediate semantic coefficient by means a two-variables function that performs formula described in Definition 3 in Chapter 3. The value is assigned to a new variable for the recursion.

- A new recursion call is performed at Line *9*. Parameters are the *termset* to be validated, the new intermediate coverage ($newCoverage$), the index of the following term $et_{i+1} \in termset$, and the new intermediate value of the semantic coefficient ($newSc$).

- In case the termset is *valid*, as a valid *coverage* has been found, at Lines *11* and *12* the termset boolean attribute $isValid$ is set to $true$, the semantic coefficient attribute sc is assigned with the highest value between the current sc and the semantic coefficient of the new *coverage* found, and the recursion takes a step back.

4.3 Minimum Window algorithm

Once a valid termset T is mined, in order to calculate its average density $ad(T)$, according to the ranking model exposed in Section 3.2, it is necessary to get for each review r that holds T ($T \subset r$) the minimum window size that contains T[18].

[18]Considering formula 3.15 in Chapter 3, formula 3.16 can be written in an equivalent way as: $ad_p(T) = \frac{|T|}{|R(p)|} \sum_r \frac{1}{minWin_r(T)}$.

This operation is not particularly complex, but it is likely to be that must be performed a remarkable number of times as it is needed for each review, of every *valid* termset mined, in every product!
Thus, the algorithm must be quite performing.

The strategy, for every termset T, is to build a representation of each review r as a sequence of couples $(term, position)$ where $term$ is a $t_i \in T$, and $position$ is the index position of a specific occurrence of t_i inside r[19]. The sequence is sorted according to the $position$ values. Figure 4.11 shows the sequence representation of a review r for a termset $T = \{t_1, t_2, t_3, t_4, t_5\}$.

review r

t_3	t_2	t_5	t_3	t_1	t_1	t_3	t_3	t_4	t_1	t_2	t_4	t_1	t_3	t_5	t_4	t_1	t_2	t_3
12	14	15	20	21	23	31	34	35	37	38	43	51	53	57	61	67	89	95

FIGURE 4.11: Sequence representation of reviews

The algorithm uses two index-pointers to mark a window in the review. At start, both pointers point the same starting couple $(term, position)$ of the sequence review. Then, the pointers apply an *accordion* dynamics: on the beginning the first pointer (*head*), moves ahead in order to find the first *termset containing window*, that is the a window that contain at least one occurrence per term $t_i \in T$[20]. To verify if the current window contains at least one occurrence for each $t_i \in T$, a *map* $\langle term, counter \rangle$ is used. The *map* is a set of couples that associates every term t_i with an occurrence counter. When the *head* moves a step ahead and finds a new occurrence of term t_i, the related counter is increased. If no counter is *zero*, then the current window contains the termset T. Figure 4.12 shows the dynamics of the *head* until it finds a termset containing window, and the relative instance of the map.

[19] Occurrence's position is calculated considering every term in the review, *not only* the terms $t_i \in T$.

[20] As T is a mined *valid* termset, by construction, there is at least one window that contains all terms of the termset for sure.

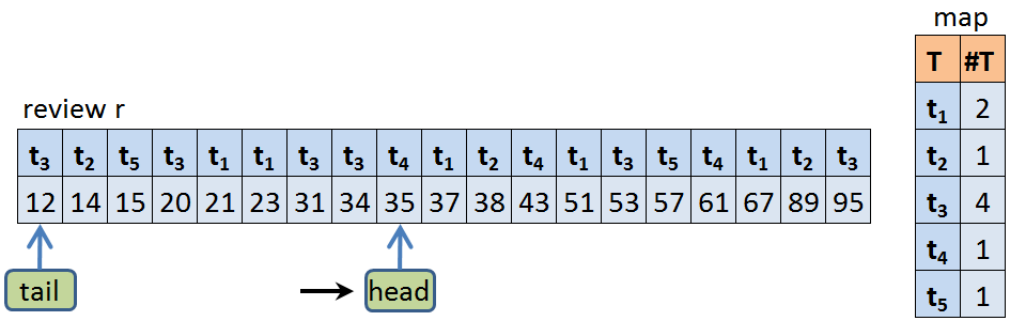

FIGURE 4.12: Minimum window algorithm. Head dynamics

When a termset containing window has been found, the second pointer (*tail*) moves ahead. For every step the *tail* goes ahead, a term t_i occurrence is left out of the window, and its counter is decreased. Every time size of termset containing window change, it is verified if the current window is the minimum. The *tail* goes ahead until one of the counters in the map becomes *zero*: this means the current window does not contain the termset anymore. At this point the *tail* stops, and *head* moves ahead until it finds a new termset containing window. Figure 4.13 shows the dynamics of the *tail* until it makes the window small not to contain the termset, and the relative instance of the counters map.

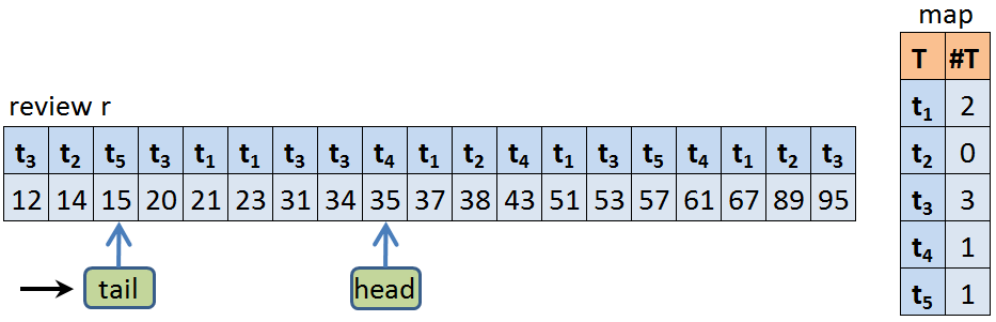

FIGURE 4.13: Minimum window algorithm. Tail dynamics

The algorithm stops when the *head* reaches the end of the review sequence and the *tail* makes the window small not to contain the termset anymore.

The Minimum Window algorithm is shown in Figure 4.14.

```
a.  function minWindowSize(T : Termset, r : seqOf⟨term, position⟩) : integer
    begin
1.    winSize = ∞
2.    head = 0
3.    tail = 0
4.    map : Map⟨term, counter⟩ = initMap(T, 0)
5.    repeat
6.      head++
7.      updateMap (map)
8.      while fullWindow(map) begin
9.        curWinSize = r.get(head).position − r.get(tail).position
10.       winSize =min(winSize, curWinSize)
11.       updateMap (map)
12.       tail++
        end
13.   until head = |review|
14.   return winSize
    end
```

FIGURE 4.14: Minimum Window algorithm

In details:

- Minimum Window algorithm is performed by **Function** minWindowSize at Line *a*. Parameters are a termset T and a review $r \in cover(T)$[21] The function returns the size of the minimum window in r that contains T.
- The function initializes its internal variables in Lines *1*, *2*, *3*, *4*: *winSize* holds the current value of the minimum window size; *head* and *tail* are the two pointers that mark the window. They start from the beginning of the review sequence; *map* is a set of couples $(term, counter)$, and $\forall t_i \in T$ the related occurrence counter is initialized to *zero*.
- The *accordion* dynamics start at Line *5* with a **repeat... until** loop.

[21] See Section 4.1 and Section 2.5. Just as a reminder, $cover(T)$ is the set of reviews that contains T.

- *head* moves one step ahead at Line *6*, and the *map* is updated at Line *7*, increasing the occurrence counter related to term t_i reached by the *head* pointer.

- If the current window contains the termset, it is verified whether the minimum size has been achieved and *tail* moves ahead. The **function** *fullWindow* is a boolean function given, and simply controls that every occurrence counter in the *map*, given as parameter, is *non-zero*. At Line *8*, if *fullWindow* returns *true*, it means a termset containing window has been found and the **while** loop is executed.

- Inside the **while** loop, at Line *9* the current termset containing windows size is calculated as the difference between the position of the term pointed by *head* minus the position of the term pointed by *tail*. At Line *10* is verified if the current window is minimal. Finally, before *tail* moves one step ahead at Line *12*, the *map* is updated at Line *11*, decreasing the occurrence counter related to term t_i that the *tail* pointer is about to leave out the window.

- The algorithm stops when *head* reaches the end of the review at Line *13*.

- At Line *14*, the function returns the size of the minimal window that contains the termset T.

In Figure 4.15 is reported the execution of the algorithm over the example showed before. The table reports all windows containing the example termset. For each window is reported the position of the first term in the review pointed by the *tail* pointer, the position of the last term pointed by the *head* pointer, and the size of the window. Finally, the minimum window is indicated.
A consideration before closing the subject. In the shown example, the minimum windows size for termset $T = \{t_1, t_2, t_3, t_4, t_5\}$ in review r, is 21. Since $|T| = 5$, termset density $d_r(T) \approx 0.24$. It is likely to be that review r is not very significant for termset T, indeed, even taking a quick look at review r representation in Figures

4.11 and following, the reader should have notices how sparse are terms $t_i \in T$ inside of r.

tail	head	window size	
12	35	23	
14	**35**	**21**	← min
15	38	23	
20	57	37	
21	57	36	
23	57	34	
31	57	26	
34	57	23	
35	95	60	
37	95	58	
38	95	57	
43	95	52	
51	95	44	
53	95	42	

FIGURE 4.15: Minimum Windows execution report.
The minimum window size of termset T in the example is 21

Chapter 5

Prototype

After the description of the theoretical model of the *HFC* system presented in Chapter 3, this Chapter is about the architecture that implements the *HFC* search engine, and the data model that allows the *HFC* system to operate.

The *HFC* system is implemented in *Java* language. The specific programming language is not fundamental, *C++* or whatever advanced programming language would fit the need. This reminder is only to justify any future reference to the *Java* language.

5.1 System Architecture

As stated in Section 3.2, when citing the paper *Hints from the Crowd: A Novel NoSQL Database* ([3]), the *HFC* system is a *NoSQL database* system that deals with collections of product reviews, that can be queried by expressing a natural language sentence.

The *HFC* system is composed by several components, each one devoted to perform a specific task as shown in Figure 5.1. In particular, there's a distinction between the *back-end* and the *front-end*: the former is responsible for *off-line* data preprocessing, in order to make data ready for *query engine*; the latter is the actual *on-line query engine* with a user interface built over, in order to let users querying the *HFC* system.

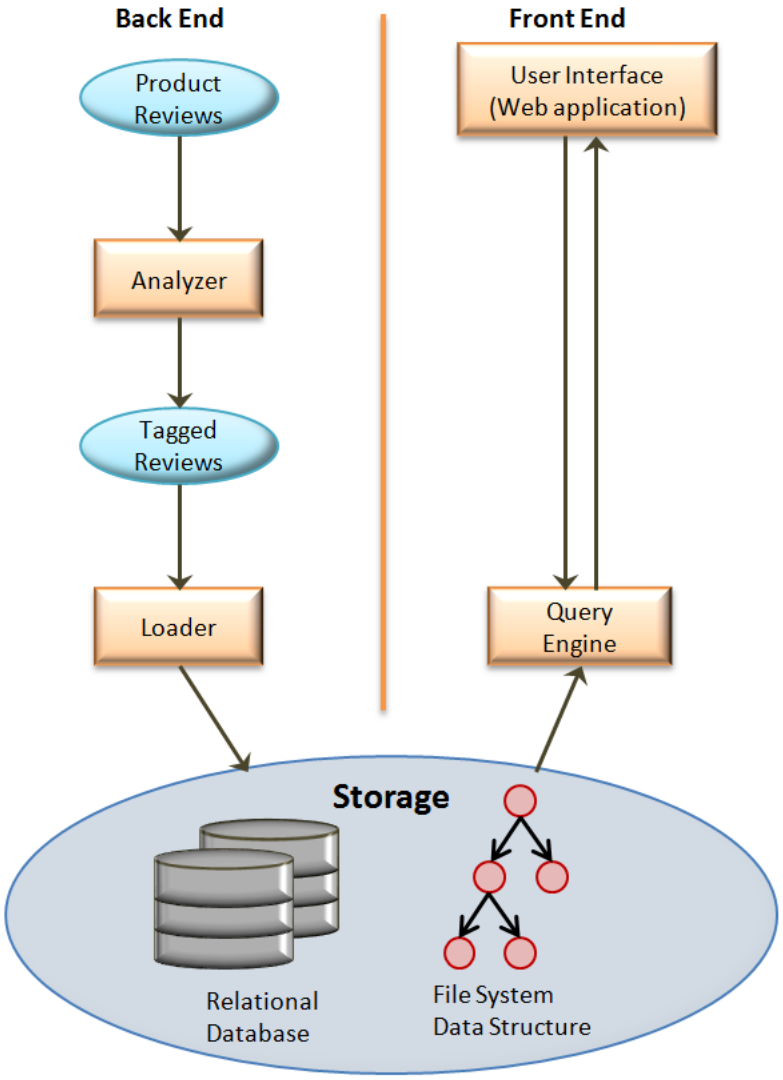

FIGURE 5.1: Architecture of the HFC System

In between the *back-end* and the *front-end*, there is the *storage system* that resides partly on a relational database, and partly on the file system. The reason for this design choice will be deeply discussed in Section 5.2.

The graphic convention in Figure 5.1, is to represent source data and intermediate results by means of ovals shapes, and to represent components that handle data with rectangles shapes.
In the following the *back-end* and the *front-end* will be analyzed.

5.1.1 Back-end

This side of the *HFC* system is responsible to make data source ready for the search engine. For *HFC* system, various data sources have been tested. Data sources will be discussed in Section 6.1, so for now it is only interesting to know that in their original form, product reviews reside as files[1] on the file system. In this form, data sources are quite useless.
The *back-end* is thus responsible for collecting, analyzing and indexing data from product reviews stored as files, creating, and uploading in the *Storage* box, the data-structure that the *HFC* search engine uses to perform user queries.
The key components in the *back-end* are the *Analyzer* and the *Loader*.

5.1.1.1 Analyzer

This component is responsible for analyzing product reviews, stored as files on the file system. For each product there is one file that is a collection of its reviews. The *Analyzer* read each file, identifying words (terms) and their grammar category (noun, verb, adjective or adverb).

[1] One XML file for each product.

The Analyzer is essentially a parser, developed with JavaCC[2], that read each product file and identifies *sentences* and then single *words* (terms) occurrences. *Sentence* identification is important as sentence is the input parameter for the *Stanford NLP Parser*[3] that is used as a library in order to perform the *pos-tagging* operation. Once the sentence has been *pos-tagged*, stopwords are discarded. The remaining tagged words are indexed with a product local index, and their occurrences are stored in a list for each word (from now on, *term*). For every term, each occurrence is a couple $(reviewNumber, position)$.

The output of the *Analyzer* is a intermediate file as the result of the serialization of the data analyzed. The output file is ready to be taken by the loader to upload the data into the storage.

It is important to underline that the *Analyzer* task is product independent, in the meaning that since the *Analyzer* is using a different indexing system for each product, the analysis of a specific product file has no effect on the analysis of another product file.

Since the *pos-tagging* operation performed by the *Stanford NLP Parser* is a heavy time consuming task, the *Analyzer* is much more slower than the following *Loader* component[4].

As a consequence of the just mentioned considerations, since there is no need to analyze product files sequentially, the *Analyzer* task has been parallelized[5].

[2] *JavaCC* is a parser generator similar to *Antlr* (both descend from *YACC*), that, taken in *input* a high level description of a computer language, returns the parser *java* code of the input language. https://javacc.java.net

[3] From the *Stanford NLP Group*: http://nlp.stanford.edu/index.shtml

[4] Performance will be discussed in Chapter 6.

[5] Briefly anticipating the performance discussion, since the machine used for the test had a *CPU* with eight different cores, it have been used eight different independent computer processes to analyze product files.

5.1.1.2 Loader

The goal of this component is to load *Analyzer* output files into the data structures on which the *search engine* will perform user queries.
Each serialized file coming from the *Analyzer* is analyzed separately from the others, and has it's on internal terms index. So the main issue related to the *Loader* is to *re-map* terms internal index into the global index.
The *Loader* works this way. First inserts a new product into the storage, and gets back a unique global index for the product. Then for each term found it controls if it is already in the *storage* or not. If the term is in the *storage*, the *Loader* substitutes the internal term index with the index given by the *storage*. If the term is not in the *storage*, the *Loader* generates a new global unique index for the term, and upload the term in the *storage*.
Once the index issue has solved for a specific term, the *Loader* provides to upload term occurrences in the *storage*, enriching term occurrence data with product index and term index. A detailed description of Data Model is demanded to Section 5.2.

The last consideration about the *Loader*, is that it is not mandatory it uploads data from files in a unique execution. *Loader* can be stopped, and then executed again in order to update the *storage* with new data.

5.1.2 Front-end.

From the architectural point of view, the key component of the *front-end* is the *Query Engine*: it exploits the preliminary work performed by *back-end* components, and works on data structures in order to answer to user queries. The *User Interface* component has been developed to give end users the capability to exploit the system.

5.1.2.1 Query engine

The *Query Engine* is the component that realizes the ranking model described in Chapter 3.

The *Query engine* receives a user query expressed in natural language from the *User Interface* component and returns back a listing of products as the result the performing of the submitted query.
In order to perform a query, the *Query Engine*, firstly tokenizes the query. The tokenized query is sent to the *Stanford NLP Parser* that provides to *pos-tag* the query. After *pos-tagging*, the *Query Engine* throws away *stopwords* from the query, and the provides to expands query terms by means of the *Wordnet* ontology, in order to get the so-called *termbase* of terms to research. Once the *termbase* has been defined, the *Query Engine* starts to upload all occurrences of terms in the termbase. With term occurrences uploaded the *Query Engine* can determined on the beginning which are the products that satisfy the query. Then for each product, the *Query Engine*, starts to build all termsets available according to the query terms and their occurrences, using the algorithm described in Section 4.1. Once all termsets for product have been built, they are used to calculate product *PRM* score according to the ranking model described in Section 3.2. When all products retrieved by the query have a *PRM* score, they are sorted in descendant *PRM* score order.
The final product listing is send back to the *User Interface* component to be shown to the user.

5.1.2.2 User interface

HFC search engine is meant to be used by users like a common general purpose search engine. When *Google* faced the market, it came out with a white page, a search box and Google's logo. That's it! Still nowadays, except for occasionally

doodles[6], Google's page is simple like that! There is no need for something else to let the user enjoy a search engine.
HFC search engine follows same policy and it can be launched from command line. The user insert his/her query from the prompt. In return he/she will have a short list of products (usually not more than ten or twenty) with a few information accompanying.

Anyway, a simple command line interface like this, is maybe too poor, to tempt people in using the *HFC* search engine. One of the main issues of the project is that it is very hard to find a critic mass of people willing to test the *HFC* search engine. Moreover, there is the issue to verify somehow quality of the results of the *HFC* search engine. Unfortunately this is not the common case of *Information Retrieval*, where usually there is a *ground truth*[7] given, and there is the need to verify quality he/she just need to compare results against the *ground truth*.
For *HFC* case, quality of results strongly depends on the feeling the user receive from the listing he/she gets. And feelings are personal! Different people feels different things. In other words, *there is no a priori optimal result*. The answer of a search engine must satisfy the user need, presenting him/her a short list of results. Statistics demonstrate that rarely users scroll beyond the first ten or twenty results. And in this short list there must be both what the user expect to find, and also what the user does not expect to find but well answer to his/her need (*good unexpected*).

In order to attract more people, and have also the chance to make different tests in order to verify *HFC* search engine's quality of answers, a web-interface has been developed[8].

In Figure 5.2 it is possible to see the starting *salutation* page of the *HFC* search engine web-interface. Here the project is presented to potential users declaring its

[6] http://www.google.com/doodles
[7] http://en.wikipedia.org/wiki/Ground_truth
[8] The web-interface has been designed by *Marcello Di Stefano*

scope, and the goal of the research. It is also described the dataset[9] on which the *HFC* has been implemented.

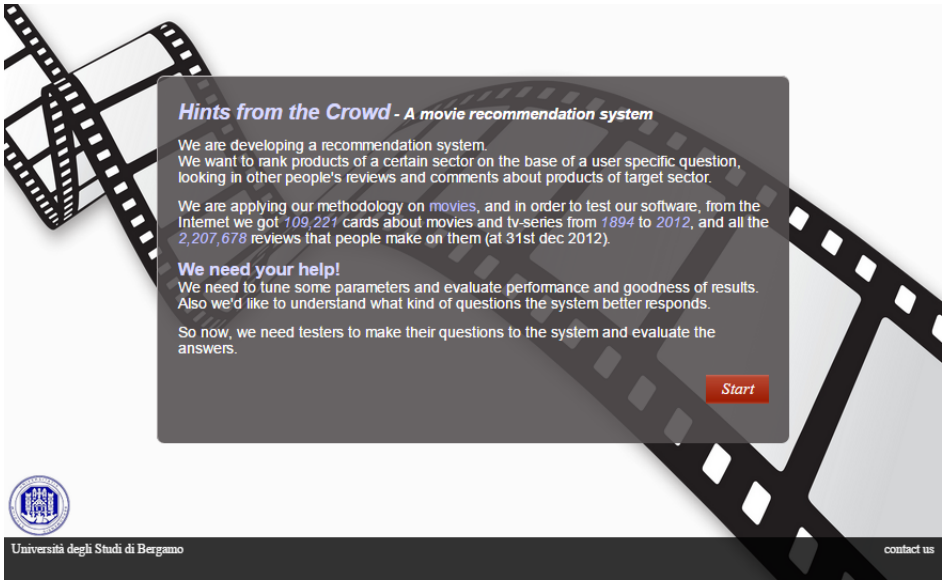

FIGURE 5.2: *HFC* web interface. Salutation

In order to use the web interface, users must be registered. Registration is needed in anticipation of future user profiling, in order to understand if the search engine better respond to specific user profiles. Data required for registration are nothing strictly personal, just a *nickname*, *age*, *sex*, approximate *location* and *job*, but actually only *nickname* is mandatory. It is asked also an *email* address, but it is not mandatory as well, and it is only to communicate with potential interested users if there is the request to perform new tests.

Figure 5.3 shows the *HFC* searching page from which the user can query the system. Similarly to *Google* web page, it is quite simple and essential.

As stated before, web interface is meant to be used to perform different tests in order to verify the quality of the *HFC* answers. Several tests has been designed,

[9] It is the IMDb dataset described in Section 6.1.3.

FIGURE 5.3: *HFC* web interface. Search box

and a couple of them have been implemented as well. However, after the initial tests, this part of the project at the moment in which this thesis has been written, has been abandoned for now and postponed to future developments, as there have been the need to solve severe performance problems when the HFC system as been performed on a very large dataset. Performance problems have been solved, and will be discussed in Section 6.4.

Anyway, some early test has been performed and Figure 5.4 shows an example of *HFC* search engine listing answer. In this case, the web interface presents the user two different listings, one is the *HFC* search engine, and the other is the answer of a different search engine (that can be either an old version of HFC search engine itself, or the answer of a different indexing system such as Lucene[10]), and it is required to the user to rate the two listing according to which one better fits his/her query. If the user wants to leave a comment, he/she can do it. The two listing are positioned randomly, in the meaning that the user can't know in advance if the answer of *HFC* search engine is *Listing A* or *Listing B*, as position change after every query.

[10]See Appendix C.

Products[11] in the listings are accompanied by a brief summary. If the user desires to know more about a specific product, he/she can select one and recall a complete *info box* about it. The example shows this situation (that's why there is a light gray transparent layer screen that made the picture a little bit opaque).

Other tests has been designed. One for example, consisted in giving the user an unsorted set of the first ten (or twenty) products coming from the *HFC* search engine answer to the user query. In this case, the user would have required to order the set of products according to his/her ideas, so that to compare the user sorting with the sorting given by the *PRM* score.
This, and other tests, unfortunately are only drafts, and are demanded to future development.

[11] *Movies*, in the example.

FIGURE 5.4: *HFC* web interface. Result listing
The transparent light gray screen in front of the result listing is activated by the user to highlight movie mini cards (the dark gray box).

5.2 Data Model

Figure 5.5 shows the logical schema of the *HFC* data-structure inside the *storage* box described in the system architecture.

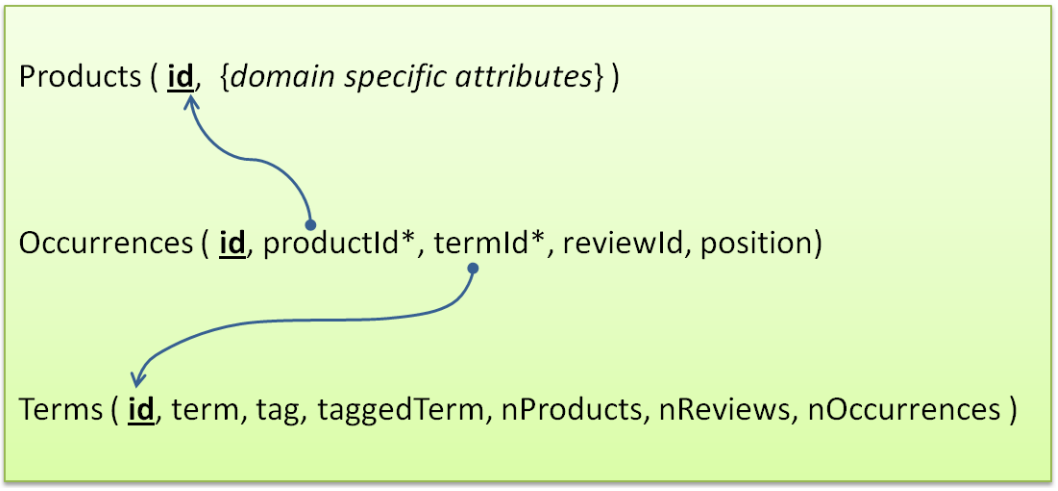

FIGURE 5.5: Logical schema of the relational database.

Somehow it is a rather simple schema, composed by only three tables: a table for *products*; a table for *terms*; a table for *term occurences in products* that links the previous tables.

Figure 5.6 shows the equivalent conceptual schema of the data-structure.

In detail the three tables are:

- Table `Products` describes each single product, and its schema is context-dependent, in the sense that the attributes are defined based on the application domain. For example, if the context is related to *motion pictures*, it is likely to find attributes concerning *movies*, such as title, director, cast, year, genre, and so on; if the context is related to automotive, it is likely to find attributes concerning *cars*, such as constructor, brand, model, engine, fuel system, number of seats and so on.

Indendently to specific product's attribute, each product in the table has a unique key identifier attribute `id`.

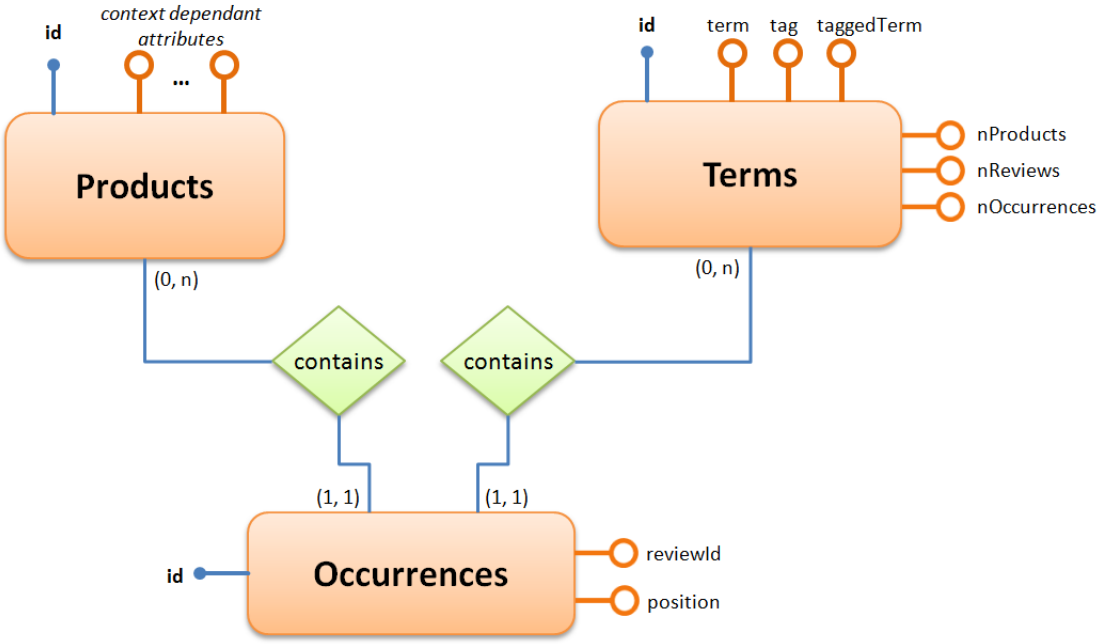

FIGURE 5.6: Conceptual schema of the relational database.

- Table `Terms` is the key table, that describes each single tagged word managed by the system. Attributes `products`, `reviews` and `occurrences` are just to quickly summarize data and count the number of products and the number of reviews in which a tagged term occurs, and the total number of occurrences, respectively. Notice that, while attribute `taggedword` denotes a unique element in the table, since a simple word can be associated to more than one grammar category (i.e. word *book* can be either a *noun* or a *verb*), it can appear several times in the table.

  Each term in the table has a unique key identifier attribute `id`.

- Table `Occurrences` describes all occurrences of tagged terms in product reviews. The table is linked to `Products` table by means of attribute `productId`

and to `Terms` table by means of attribute `termId`. Then, attribute `reviewId` is actually the identifier of the review that holds the term occurrence in the product linked by the previous attributes. The attribute `position` finally indicates the position of the occurrence in the review. Occurrences have an `id` attribute but actually is quite useless. It has been designed only for designing coherence.

5.2.1 Data storing.

The data structure described in the previous Section is merely *conceptual*. It can be implemented on whatever traditional database management systems (*DBMS*).

On earlier *HFC* system versions, when tests were made on medium-small datasets, data structure was completely implemented on *Postgres* database. The only *trick* in data uploading phase, is to remember to *deactivate insert query autocommit*[12]. With this *trick*, uploading is quite fast, especially compared to *analyzing* phase[13].

When the *HFC* search engine, has been executed on a very large dataset, it had been clear this solution was not good enough. Details will be discussed in Chapter 6, but anticipating a little bit the subject, the problem was the very large size of the *Occurrences* table. Even with optimized database indexes, the access time to data was too slow to get answers from the search engine in an user acceptable time. So it had to be implemented a different solution.

Since there are few tables in the conceptual schema, and since links between tables are quite simple, table *Occurrences* has been moved on the file system, splitted in

[12] During the first test, *autocommit* was active and uploading the first small testing dataset took... this thesis would never have not been written.

[13] Approsimatively ten or more times faster. Exact uploading time has not profiled during early tests, and anyway they are not significative because the small size of earlier datasets.

several files. That's why before has been stated that *storage resides partly on a relational database, and partly on the file system.*

Occurrences are stored according to a *reverse index* policy. For each term has been created an *occurrences file* having as file name the *id* of the term. Since terms are likely to be more than hundreds of thousands[14], this can create troubles with the file system as a single folder cannot hold a too large number of files, and anyway it would have been not easy to handle, *occurrences files* are grouped into groups of 1000 files. For each group there is a single folder, and the folder name is given by the integer part of the division of term *id* by 1000. So, just to better explain, folder 0 holds files with an *id* between 0 and 999, folder 1 holds files with an *id* between 1000 and 1999, and a generic folder k holds files with an *id* between 1000*k and 1000*k+999.
Each *occurrences file* has a simplified version of the structure of table *Occurrences*: as *termId* is in the file name, and since occurrence *id* is useless, *occurrences file* holds just a triple (*productId*, *reviewId*, *position*) for each occurrence. How this information is actually coded inside the file will be discussed in Chapter 6, since there have been several different solution in order to improve performance.

Another consequence related to improving performance, is that table *Products* and table *Terms*, during the whole search engine exection time, must be uploaded and reside into central memory. Data upload happen during search engine initial setup phase (see Section 5.2.2). As a consequence, even table *Products* and table *Terms* have no need to reside on *DBMS*, but could be stored on the file system with any kind of file structure suitable for the need (e.g. *XML*). Actually, at the moment table *Products* and table *Terms* **do** still reside on *Postgres* database, but it is only not to write extra programming code, since the file solution for this case do not bring any substantial advantage.

[14]On larger dataset used, there were actually more than one million of different terms!

To conclude, from the above considerations, it should be evident that, to perform a user query, the *HFC* sarch engine **do not** execute any single *sql* code. That's why can be properly defined as a ***NOSQL database***.

5.2.2 Run-time data model

In the following will be shown the run-time data model, i.e. the data model used by the *HFC* search engine during its execution. It is just a quick overview of the main java classes used. For each java class, will be shown a simplified conceptual *class-diagram* with the main attributes. The real case is much more complex, but since complexity is related only to technical details, they have been omitted.
In the following *class-diagrams* there is a large use of the *Hashtable* objects since this kind of data structure guarantee the best performance in terms of execution time, even if they need a large amount of central memory. But central memory usage has not been a problem! Not nowadays. Java *Hashtables* allow to access directly an *element* by its unique *key*, and quickly return also the set of *elements* and also the set of *keys* as list objects (i.e. *Enumeration*). For the scope of *HFC* project *Hashtables* are mandatory!

Figure 5.7, shows the class diagram of the *Search Engine*. When the Search Engine is being instantiated, during its setup, all *products* and *terms* in the data base are uploaded in central memory to improve performance[15].

- Attribute *productTable* is an hashtable that holds all products in the database. The key to access a *product* is its unique *id* number.

[15]In Chapter 6 will be described the biggest data set used. It has more than $100,000$ products and generates more than $1,000,000$ distinct terms, and there have been no problems in keeping all this information in central memory during execution.

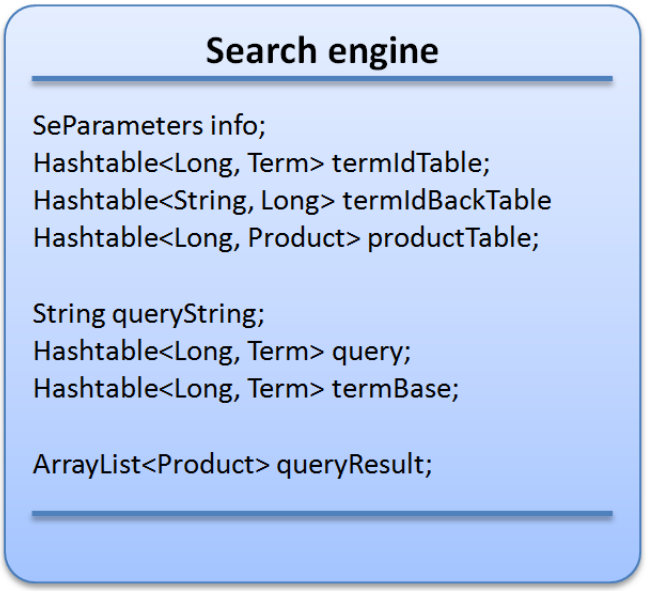

FIGURE 5.7: SearchEngine class diagram

- Attribute *termIdTable* is an hashtable that holds all *terms* in the database. The key to access a *term* is its unique *id* number.
- Attribute *termIdBackTable* is an hashtable that holds all *terms-id* in the database with a reverse index access. It is needed to get *terms-id* from the text after a user query.
- Attribute *info* is a generic container of everything, including *execution-time profiling* attributes.
- Attribute *queryString* holds the query inserted by the user a text in natural language.
- Attribute *query* is the set of query terms before the expansion, represented as an hashtable with term-ids as keys.
- Attribute *termBase* is the set of expanded query terms (*termbase*) to search after the expansion, represented as an hashtable with term-ids as keys. Terms in the termBase are those terms on which termsets are built.

- Attribute *queryResult* is the list of products retrieved by the Search Engine after a user query and sorted according to their *PRM* score.

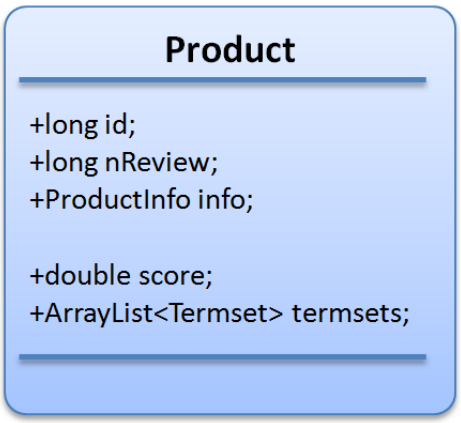

FIGURE 5.8: Product class diagram

Figure 5.8, shows the class diagram of a generic *product*. As stated before, *products* are upload to main memory all *products* and *terms* in the data base are uploaded in central memory to improve performance. Some attributes inside the object are *static*, in the meaning they never change during all search engine execution time. Other attributes depend instead on the specific user query, and they are discarded by the *garbage collector* after the answer of the search engine.

- Attribute *id* is the unique index identifier of a specific product given from the database.
- Attribute *nReview* is just a counter for the number of user reviews for the product.
- Attribute *info* is just a collection of context depending information about the product, e.g. *brand* and *model* for a car, or *name*, *location* and *stars* for a hotel, or *title*, *director*, *cast*, *genre* for a movie.

- Attribute *score* is the *PRM* score the product gets after performing a user query.
- Attribute *termsets* it's the set of termsets mined after performing a user query, and by which can be calculated *PRM* score. After this operation *termsets* are discarded by the garbage collector.

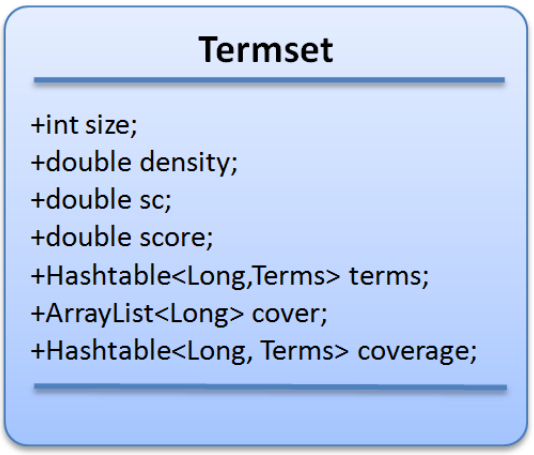

FIGURE 5.9: Termset class diagram

Figure 5.9, shows the class diagram for a generic *termset*. Termsets are kept into *products*, and stated before, they are discarded by the garbage collector after a user query has been answered.

- Attribute *size* denotes the length of the termset.
- Attribute *density* reports the average density of the termsets
- Attribute *sc* reports the *semantic coefficient* associated to the termset.
- Attribute *score* reports the contribute of the the termset to the global *PRM* score.
- Attribute *terms* it's an hashtable with *terms* that compose the termset.

- Attribute *cover* returns the *cover* of the termset, that is the list of reviews in which the termset is found. For termsets it's necessary to have only the *id numbers* of *reviews*.

- Attribute *coverage* describes the *coverage* of the termset over the user query terms. As a reminder, in order to be *valid*, the number of elements in the coverage must be equal to the size of the termset[16]

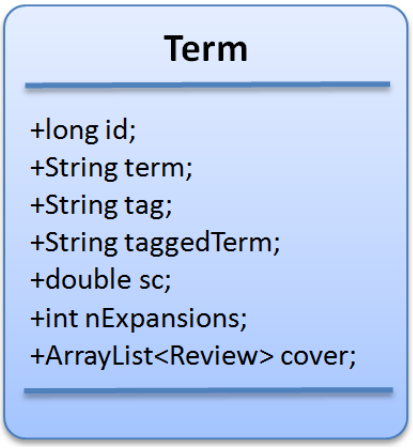

FIGURE 5.10: Term class diagram

Figure 5.10, shows the class diagram for a generic *term*. Term objects are used for describing both the *query* and the *termbase* and the generic *termset* and the termset *coverage*. Thus, some attributes are likely to be used in a context than in another. When attribute is not used is left empty.

- Attribute *id* is the unique index identifier of a specific term given from the database.

- Attribute *term* is the term simple text.

- Attribute *tag* represent the grammar role of the term inside to of a sentence (e.g. *noun*).

[16] See Section 3.2.3

- Attribute *taggedTerm* is the combination of both the attributes *term* and *tag* put together in the form `term/tag`.
- Attribute *sc* reports the *semantic coefficient* associated to the term.
- Attribute *nExpansions* reports the number of expansions available for a query term.
- Attribute *cover* return the list of reviews that hold the term. When the term is part of a termset review objects are filled with term-occurrences. After the search engine performs a query, termsets to rank products are discarded by garbage collector, and so are the terms (and the reviews) associated to termsets.

FIGURE 5.11: Review class diagram

Figure 5.11, shows the class diagram for a generic *review*. Reviews are discarded with termsets and terms by garbage collector after search engine performs a user query.

- Attribute *id* is the order number for a review in a specific product.
- Attribute *occurrences* contains all occurrences in the review of the object (*Term*) that holds the review. This is the main memory consuming attribute. It must be discarded by garbage collector after usage!

Chapter 6

Performing evaluation

This Chapter illustrates the performance of the *HFC* search engine.
On the beginning will be described the *datasets* used to test the *HFC* search engine, and will be explained which dataset has been chosen for the final evaluation test and why. Then, will be presented the indexing process of the dataset used and will be presented the results obtained with different indexing policies. Finally, will be showed the performance of the *HFC* search engine over the indexed dataset.

6.1 Datasets

The challenge to find out a substantial and significant dataset has been one of first main issues of the project.
Since the first conferences where earlier version of *HFC* has been presented, one of the main questions from the audience was *"how big is the dataset?"* in order to understand if the performance could be interesting applied in a substantial real business case.

The other recurrent question was *"how significant is the dataset?"* in order to understand if the results given by the system correctly match with the user query.
This paragraph presents 3 different datasets on which the *HFC* system has been tested. The first dataset to be presented is a corpus of *14,928* blogs containing a total of *671,760* posts. Then, the next dataset is a collection of *9701* reviews about *324* movies downloaded from *Epinions* website. The last dataset is a collection *2,207,678* reviews about *109,221* movies downloaded from the *IMDb* website described in Section 1.1.

6.1.1 Blog Corpus

The *Blog Corpus* has been the first substantial dataset big enough (more than 800 MBytes of text) to make significant performing tests, and it was the dataset used for tests in the paper [2] presented at the *KES* conference held in San Sebastian (Spain) in September 2012, and whose ranking model is described in Section 3.3.2. The dataset, is presented as first, as actually it was downloaded, because another research, before the *HFC* project started, and its file structure was re-used for the following datasets.

The *Blog Corpus* has been downloaded from the webpage of Professor *Moshe Koppel*[1] hosted in the *Bar-Ilan University* (Israel) website. In [17] Professor Koppel et al. analyze how age and gender of bloggers affect blogs. As reported by Professor Koppel[2]:

> *The* Blog Authorship Corpus *consists of the collected posts of* 19,320 *bloggers gathered from* blogger.com *in August 2004. The corpus incorporates a total of* 681,288 *posts and over 140 million words - or approximately 35 posts and 7250 words per person.*

[1] http://u.cs.biu.ac.il/~koppel/
[2] http://u.cs.biu.ac.il/~koppel/BlogCorpus.htm

Each blog is presented as a separate file, the name of which indicates a blogger ID and the blogger's self-provided gender, age, industry and astrological sign. (All are labeled for gender and age but for many, industry and/or sign is marked as unknown).
All bloggers included in the corpus fall into one of three age groups:

- 8240 "10s" blogs (ages 13-17)
- 8086 "20s" blogs(ages 23-27)
- 2994 "30s" blogs (ages 33-47)

For each age group there are an equal number of male and female bloggers. Each blog in the corpus includes at least 200 occurrences of common English words. All formatting has been stripped with two exceptions. Individual posts within a single blogger are separated by the date of the following post and links within a post are denoted by the label urllink.
The corpus may be freely used for non-commercial research purposes.

Table 6.1 reports briefly the main data concerning the *Blog Corpus*.
Actually, of the *19,320* bloggers cited by Professor Koppel, only *14,928* has been taken into account during the evaluation test of the *HFC* system, since only those blogs with a number of posts greater than or equal to 5 has been considered, since, at that time, the ranking model was too much affected by the *support* issue as discussed while introducing *termset density* in Section 3.2.5.

Hence, the number of the indexed reviews has been *671,760* (that means an average of *45* reviews per blog). The number of single distinct terms indexed was *932,435* for a total of *13,576,243* occurrences. The reason for such a large number of distinct terms was due to two main causes: the *first cause* was that the analyzing parser,

Products (Blogs)	Text Size	Reviews
14,928	800 Mb	671,760
	Reviews per Product	
Min	Max	Average
5	4,421	45
Distinct Stopwords	Distinct terms	Term occurrences
$\sim$ 600	932,435	13,576,243

TABLE 6.1: Blog Corpus summary data

except for *stopwords*, does not filter out colloquial exclamations like *aaahhh* or *ahhhh* that are massively present in people written text. Moreover natural language is often full of colloquial jargon like terms *nite* or *nit* instead of the more correct form *night*. And to conclude the parser does not fix typos so the term *nogh* is distinguished from the term *night*.

The *second cause* is that during the indexing phase terms are not stemmed so terms like *night* and *nights* are considered as two different distinct terms. Moreover, the *HFC* considers not the mere simple term, but its *tagged* form as result of the analysis performed by the *Stanford Pos-Tagger*, so actually the term *run* can be considered for example as a nouns run_{noun} or as a verb run_{verb}[3] depending on the context and the role of the term *run* in the sentence in which it is placed.

Each blog is an *XML file* with a simple structure as shown in Figure 6.1 where each file is a simple sequence of couples of tags *date* and *post* that respectively holds information about the date when the blogger wrote the post and the text of the post.

The main issue concerning the *Blog Corpus* was that it was too much hard to control the quality of the *HFC* search engine answer: since the results of the search engine is a ranked list of blogs where the only structured information is *gender*, *age*, *industry* and *astrological sign* of the blogger, in order to understand if a blog better match a

[3] Instead of the strings *verb*, *adjective*, *noun*, *adverb* the *HFC* systems uses numeric codes

⟨Blog⟩
⟨date⟩ [*date of the* 1_{st} *post*] ⟨/date⟩
⟨post⟩ [*text of the* 1_{st} *post*] ⟨/post⟩
⋮ ⋮
⟨date⟩ [*date of the* n_{th} *post*] ⟨/date⟩
⟨post⟩ [*text of the* n_{th} *post*] ⟨/post⟩
⟨/Blog⟩

FIGURE 6.1: Structure of XML files

query like *what is the best music for a Saturday night home party?* it was necessary, in most of the cases, to read a good share of posts as the structured information mentioned before can only give a slight indication of the goodness of the result.
So the *Blog Corpus* satisfied well the question *"how big is the dataset?"* but it was too weak to satisfy the other question *"how significant is the dataset?"*.

Anyway, as stated before, the file structure described in Figure 6.1 is important as the *parser* described in Section 5.1.1.1 has been designed in order to process XML-files of this type. So, when afterwards the dataset has been changed, it is has been managed in order to have the same file structure so to re-use the same parser component with no modification.

6.1.2 Epinions

Epinions[4] is a web site where people can leave their review about a wide variety of products. The site is structured in areas like *Electronics*, *Health & Beauty* or *Computers*. Each area is divided into families of homogeneous products, where products can be selected by various criterion as by specific product, or by brand, or by range of price just to name a few. Each single product can be reviewed by registered users (in Section 1.2 there are some example of *Epinions* web site).

[4] http://www.epinions.com

In order to test the *HFC* search engine the attention has been focus on *Movies*. Movies like any other product can be reviewed by people and have the apparent advantage that usually, at least for *box-office-top* movies, that are likely to be wide known, it can be easy to have a previous idea so to answer the question *"how significant is the dataset?"*, so in a case where the user query is like *what is a romantic movies about married life problems?* is quite easy to understand at first glance that a movie like *When Harry met Sally*[5] is likely to be better ranked than a movie like *A clockwork orange*[6], with a small need to read reviews.

Thus, a *web-crawler* has been specifically developed in order to grab movie reviews from *Epinions*. The output of the web-crawler has been an XML-file for each movie with the same XML-structure described in Figure 6.1 for the *Blog Corpus* case (Section 6.1.1), where each *date* tag has been used just to hold the *id*-number of the review, and each *post* tag has been used to hold the text of the review. Table 6.2 reports briefly the main data related the movies-review downloaded from *Epinions*.

Products	Text Size	Reviews
324	33 Mb	9,701
Reviews per Product		
Min	Max	Average
5	509	30
Distinct Stopwords	Distinct terms	Term occurrences
$\sim$ 600	72,802	709,845

TABLE 6.2: *Epinions* dataset summary

The *Epinions* dataset has been the first dataset on which the *HFC* search engine has been tested, and it was the dataset used for tests in the paper [1] presented at the first *DATA* conference held in Roma in July 2012. The ranking model at that time was too immature and suffered of all issue described in Section 3.3.1.

[5] http://www.imdb.com/title/tt0098635
[6] http://www.imdb.com/title/tt0066921

Anyway, a it was raised also a remark related to the smallness of the dataset. Indeed as Table 6.2 shows, the dataset is composed by only 324 movie of various genre, with a minimum of 5 reviews as the ranking model suffered for the same *support* issue discussed before.

6.1.3 IMDb

After the first two conferences, it was clear there was the need to get a large, substantial, significant dataset. The choice fell on the *IMDb* as the database available on the Internet was definitely large, and full of extra useful information, and also because the *movie* context was already known after *Epinions* dataset.

IMDb has been already described in Section 1.1.
To grab data from the *IMDb* web site has been used web crawler similar to the one used for *Epinions* site, but specifically designed for the *IMDb* web site.
Once the web crawler had been launched, it took about six days to download more than 2 *millions* reviews about more than $100,000$ movies. Downloading time has been recorded for each movie, but the information is little significant as depends too much on the net bandwidth available at downloading moment, and the traffic directed to the *IMDb* servers.
A summary of the *IMDb* dataset is reported in Table 6.3.

The XML file structure has been changed according to the schema described in Figure 6.2: tags's names and main structure remained the same as for the other datasets, but there is the addition of a REPORT tag at the beginning of the XML file. In the REPORT tag are stored, as in a property file (i.e. a couple PROPERTY:VALUE; for each raw), a set of information about the movie to provide to the user thru the *HFC* web interface described in Section 5.1.2.2. This information are: *movie title*;

Products	Text Size	Reviews
109,221	$>$ 3 Gb	2,207,678
	Reviews per Product	
Min	Max	Average
1	4,876	20
Distinct Stopwords	Distinct terms	Term occurrences
$\sim$ 600	1,151,893	216,345,522
		Stopwords occurrences
		$>$ 300M

TABLE 6.3: IMDb Dataset main data

IMDb code; *IMDb web-page URL*, *title type*[7], *director*, *casting stars*, *releasing year*, *genre*, *plot*, *country*, *language* and also downloading time (in milli seconds). All this information is then uploaded into the *storage* system.

⟨Blog⟩
⟨report⟩ [*movie information (title, etc)*] ⟨/report⟩

⟨date⟩ [1_{st} *review*] ⟨/date⟩
⟨post⟩ [*text of the* 1_{st} *review*] ⟨/post⟩
⋮ ⋮
⟨date⟩ [n_{th} *review*] ⟨/date⟩
⟨post⟩ [*text of the* n_{th} *review*] ⟨/post⟩
⟨/Blog⟩

FIGURE 6.2: Structure of *IMDb* XML files

When the *IMDb* dataset has been downloaded, the *support* issue that affects the *Epinions* and *Blog Corpus* datasets was solved by the new ranking model described in Section 3.2, so there have been no lower limit concerning the number of reviews. Thus, there have been both downloaded, *movie*, *tv-movies*, *tv-series* and *mini-series* from 1894 (*"Miss Jerry"*[8], which actually has 1 review) to all titles produced until *31st of December 2012*.

[7]There have been downloaded *movie*, *tv-movies*, *tv-series* and *mini-series*
[8]http://www.imdb.com/title/tt0000009

In Table 6.4 is reported a comparison between the three datasets used.

	Blog Corpus	*Epinions*	*IMDb*
Products	14,928	324	109,221
Text Size	800 Mb	33 Mb	> 3 Gb
Reviews	671,760	9,701	2,207,678
Min #reviews	5	5	1
Max #reviews	4,421	509	4,876
Average #reviews	45	30	20
Distinct Stopwords	600	600	600
Distinct terms	932,435	72,802	1,151,893
Term occurrences	13,576,243	709,845	216,345,522

TABLE 6.4: Comparison between datasets

From the table is soon evident that *Epinions* dataset is the smallest by far. Instead, at a first glance there is not such great difference in size between *IMDb* and *Blog Corpus* datasets. *IMDb* dataset has about 4 times the text size with respect to the *Blog Corpus* dataset. And even the number of the single terms identified are only about 20% more.

So, when the *IMDb* dataset has been downloaded and then indexed by the *HFC back-end* system, it was likely to suppose that *HFC* search engine performance would have been changed something in between +20% and +400% because the biggest size of the dataset. When the *HFC* search engine has been performed against the *Blog Corpus* dataset, there have been no issue at all concerning performance, indeed no performance data were recorded as apparently little meaningful. Thus, before the first time the *HFC* search engine had to be performed against the *IMDb* dataset, performance aroused no concern.

Then, the first test was made. And after more than 30 minutes, the *HFC* search engine had still to answer, and so was forced to quit. The system had been checked and double checked, and then the test was repeated. And still, happened the same again.

Houston, we've got a problem!

6.2 The quest for performance

As stated at the end of the previous Section, when the *HFC* search engine had been tested for the first time on the whole *IMDb* dataset achieved such bad performance in query execution time that made the project almost to be abandoned. The *IMDb* dataset was considerably larger than the *Blog Corpus* dataset, but not so larger to justify such decay of performance. Not the difference in size itself!
What made the performance decay so bad was the very large number of occurrences in *IMDb* dataset. Table 6.4 shows that IMDb dataset had more than 216 *millions* occurrences to be indexed, that is about more than 16 times the number of occurrences found in *Blog Corpus* dataset.
At the time of this test, the *HFC data model*, described in Section 5.2, was completely implemented on a *Postgres database*. The very large number of occurrence in the *IMDb* dataset put in crisis the internal indexes of the Postgres database, and made the performance so poor.

The project was about to be abandoned, unless finding an alternative to the database. The alternative was given by *Google*. It is widely known that *Google* does not use database to index its data: all information are stored in files. This fact was taken as inspiration to design the *reverse index file* solution to index terms occurrences, described in Section 5.2.1.
As soon as the new occurrences indexing method as been implemented, performance suddenly improved. But not as much as desired. The *reverse index file* method had to be further improved. Thus, there have been several improvements of the occurrences indexing method.
Table 6.5 reports the historical trend of performance as changing the occurrences indexing method.

All experiments, except the last, have been run on a PC with two *Intel Xeon Quad-core 2.0GHz/L3-4MB* processors, 12GB RAM, four 1-Tbyte disks and *Linux* operating system. In the last experiment, disks were substituted with one 256$GBSSD$ storage system.

All experiments, except the first, have been made by measuring the average execution time of the *HFC* search engine on a set of 25 queries (see Appendix E).

HFC Version	Occurrences Indexing Method	Query Average Execution Time	
1.0	Database	> 30	minute
1.1	File Reverse Index (FRI) - Text	12	seconds
1.2	FRI - Binary	5.1	seconds
1.3	FRI - Binary zip	2501	millisec
1.4	Ver. 1.3 + Multi thread (5)	1995	millisec
1.5	Ver. 1.4 on SSD	1211	millisec

TABLE 6.5: Indexing History

Table 6.5 reports for each experiments the *version* of the *HFC* search engine used, the *occurrences indexing method* used, and the average time in executing the set of testing queries.

The versions of the *HFC* search engine are the followings:

- Version **1.0**

  It is the initial version *HFC* search engine, where terms occurrences were stored completely on the *Postgres* database. Tests have not been completed as execution time was not acceptable.

- Version **1.1**

  This is the first version in which term occurrences have been stored on the file system as discussed in Section 5.2.1.

Each file of term occurrences is a text file and have *term id* as filename. Each file contains one term occurrence, as a text, per raw. Each term occurrence is a triple (*productId*, *reviewId*, *position*), where comma *character* separates each value. Each value uses the minimum number of characters required to represent the value.

Since the movies in the *IMDb* dataset[9] are more than 100,000 and the highest number of reviews per movie is about 4,500, and considering reasonable a maximum review length of 10,000 terms, each occurrence needs an average of 14-15 characters, considering also separators and *end of line* characters. Since occurrences in the *IMDb* dataset are more than 200 millions, the global size of all term occurrences files was over 3Gb (that becomes more than 5Gb considering the large number of i-node of the Unix system in order to manage the large number of generated files.)

Using this occurrence indexing method, the average execution time dropped drastically around 12 seconds. It was still too long to be acceptable by the average user, that expects an answer in less than *1 second*, but it pointed out that this way deserved to be deeply investigated.

- Version **1.2**

  After *version 1.1*, the *HFC* search engine started to be profiled in order to understand where performance could be further improved[10]. The result was that occurrences loading was still to much time consuming both because the loading itself and the text conversion into number operation.

  So, it has been decided to try to use a *binary* representation of the occurrences, that should require less space to represent an occurrence and, also, avoid to perform the conversion operation.

  Thus, each textual occurrences file, has been converted into a binary file with

[9] See Figure 6.4

[10] Unfortunately, profiling information until version 1.3 are lost.

fixed pattern. Each occurrence triple (*productId*, *reviewId*, *position*) is represented by a 8 byte pattern: 4 bytes for *productId*; 2 bytes for *reviewId*; 2 bytes for *position*. No separator have been used.

With this occurrence coding method, the global size of all term occurrences files decreased around to 1.7 Gb (more or less 3.5 Gb on the file system).

More important was the performance improvement around 5 seconds.

- Version **1.3**

  Version 1.2 has been a good development step, but still it was not optimal. Thus, the following version contemplated a compression of binary data. Instead of having a fixes binary format, it has been designed a *zip* format. The basic idea is that very common terms, are likely to be present in products, but most of all in single review, a considerably large number of times. Thus, since in occurrences files, occurrences of the same product, and the same review are sequential, there is a remarkable redundancy of the same productId and reviewId information. The idea, thus, is to collect this information in order to write (and then read) it only one time.

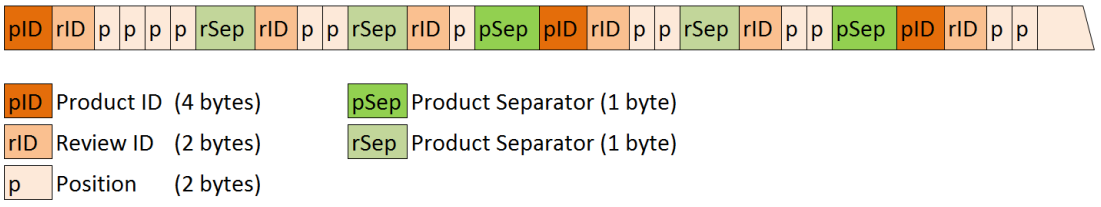

FIGURE 6.3: Occurrences file *zip* format

The *binary zipped occurrences file* format is showed in Figure 6.3. In a few words, exploiting sequentiality of occurrences, an occurrences file, is divided into blocks. Each block starts with a 4 bytes *productId* and groups occurrences of the same product. Blocks are separated by a product separation byte.

Blocks are internally divided into chunks. Each chunk starts with a 2 bytes *reviewId* and groups a list of 2 bytes occurrences position in the same review. Chunks are separated by a review separation byte.

This improvement makes *HFC* performance quite interesting: the average execution time on the testing queries decreased around to 2.5 seconds. Still to much, but not far from goal.

- Version **1.4**

  From profiling, emerges that in *Version 1.3*, about 80% of execution time was still spent on occurrences loading. Thus, the next development test tried to exploit CPU cores, by replicating term occurrences loading phase into 5 different threads[11].

  To allow the search engine to perform this task, the file system of occurrences files has been horizontally split into 5 five similar but smaller file systems, where each of one includes $\frac{1}{5}$ of every occurrences files. The split divided products into 5 different groups, and has been made in order to balance the weight of each mini file system.

  Results show a further improvement, as the average execution time on the testing queries decreased to less than 2 seconds. Not the goal yet. But quite close.

  Unfortunately, multi threads split the loading task into smaller tasks, but as all threads were sharing the same *memory bus*, the improvement has not been as good as it was expected.

- Version **1.5**

  At the end of the *PhD*, there have been the chance for a short period of time, to substitute the old mechanical rotating hard-drive of the server machine, with a

[11] Actually, as the machine server has 8 different cores, it could have been used 8 different threads, but as the server machine was shared with other users, it would have been better not to saturate it. It was important to demonstrate that this multi-thread approach could improve performance, not the performance itself.

more modern and performing SSD storage. Articles and specialized magazines report that *SSD*s are 10 up to 100 times faster than traditional *HDD*s, so a test of *Version 1.4* upon the the same server machine empowered with *SSD* storage has been tried.

Anticipating for a little while the content of Table 6.8, it become evident that the new machine configuration brings a substantial improvement. Average execution time decreased around to 1.2 seconds, that is *very close to goal* of achieving the answer in *less than 1 second*!

Figure 6.4 graphically reports on an histogram, the average execution time trend as varying the *HFC* version.

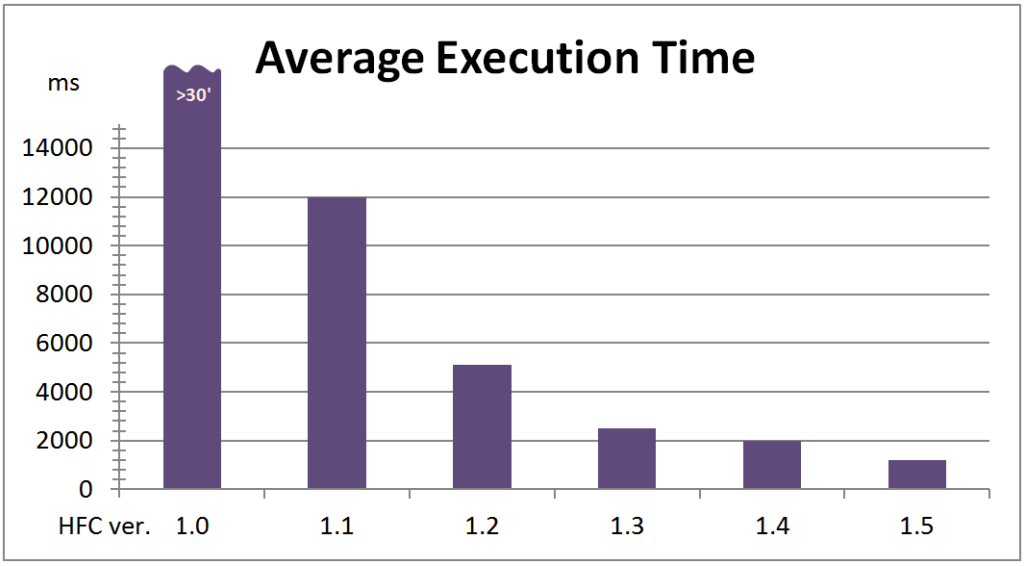

FIGURE 6.4: *HFC* performance trend

After this long *excursus*, a consideration must be done. Until now the *HFC* has been always performed against the whole *IMDb* dataset. But in this context, and much more in a different application contexts, such as could be an application toward the hotel booking issue, users probably don't want or need a research of products on the whole dataset, but they have a little idea how to select *a priori* some of the products

they want. As an example, a traveler don't need to have the *complete world-wide* list of the hotels far from *belfries* so he/she can sleep at morning, but only those hotel in Prague where he/she is going to spend a weekend in April 2015. And maybe he/she wants only hotels with a free wi-fi connection!
In other words, in most of the cases it should be considered a smaller piece of the dataset. The selection of products usually depends on the classical structured features that traditional databases are very good exploiting. The database problem is only how to sort results in an order the user can appreciate.
A combination of the *HFC* system with traditional databases could perfectly fit both the needs: 1. improving HFC performance; 2. giving traditional databases a sorting system closer to users *desiderata*.

6.3 Indexing

Once has been clear the *quest for performance* of the *HFC* search engine, it is interesting having a look at time taken for the whole indexing process.

As described in Section 5.1.1, the indexing process is performed by the *back-end*. The *back-end* is composed by two main component: 1. the *Analyzer* that is responsible to retrieve term occurrences from the source data (1.1 *parsing*), and then to *pos-tag* the terms retrieved (1.2 *pos-tagging*); 2. the *Loader* that, is responsible to load toward the traditional database information about products and terms (2.1 *Db loader*) and then also to create reverse index occurrences files (2.2 *File writing*).

Table 6.6 reports the aggregate time taken by all this tasks. Time is expressed in *hours*.

Since has been state more than once, that pos-tagging is very hard time consuming task, a comparison was made between indexing the *IMDb* dataset having the *pos-tagger* active, and indexing the same dataset with the pos-tagger turned off.
Turning off the pos-tagger means do not call the *Stanford Parser* and just assigning to each term found the same grammatical label.

	Pos-Tagger	active	inactive	Diff %
	Distinct tagged terms	1,151,827	776,852	48.27%
	Occurrences	216,345,522	216,345,522	0.00%
p1	Analyzer = p1.1+ p1.2	2,226.80 h	3.82 h	+58193.19%
p1.1	Parsing Time	2.11 h	2.42 h	-12.81%
p1.2	Pos-tagging Time	2,224.69 h	1.40 h	+158806.43%
p2	Loader = p2.1+p2.2	56.05 h	51.76 h	+8.29%
p2.1	Db loader	48.32 h	45.49 h	+6.22%
p2.2	File writer	7.73 h	6.27 h	+23.29%
T	Total Time (T=p1+p2)	2,282.85 h	55.58 h	+4007.32%

TABLE 6.6: Pos-tagging vs No pos-tagging. Indexing time

Comparing the two cases is evident that in both case the total number of occurrences should not change. What change is the number of distinct term. When the the pos-tagger is turned on, the number of distinct termset raises almost of 50%. This would have been easily predictable.
What is impressive the quantity of time taken by pos-tagger!
When the pos-tagger is turned off it takes only 36.6% of the *Analyzer* time, but when the pos-tagger is active it takes instead more than 99.9%
When the pos-tagger is turned off the *Analyzer* share of *back-end* time is only 6.9%, but when the pos-tagger is active it take instead than 97.3%.
That's why the Analyzer has to be duplicated into as much as possible processes! Without *Analyzer* process duplication the whole *back-end* indexing time would have been taken more than 92 days! *More than 3 months*! Duplicating the *Analyzer* in 8 distinct processes this time has been reduced to *only* 15 days! Two weeks!

Table 6.7 shows how turning off the pos-tagger affects the *HFC* search engine in terms of performance.

Pos Tagging	active	inactive	Diff %
Total time	1,995 ms	3,480 ms	+74.47%
Movies	2,067	2,994	+44.85%
Average Terms in termbase	21	41	+95.23%
Occurrences Retrieved	107,200	226,994	+111.75%
Termsets analyzed	5,414	13,795	+154.80%

TABLE 6.7: Pos-tagging vs No pos-tagging. Query execution

The version of HFC search engine used is the 1.4. The set of queries submitted is always the same listed in Appendix E. It is evident that turning off the pos-tagger raise the average execution time, since the number of terms in *termbase* is significantly higher, and so the number of occurrences loaded.

6.4 Execution

Finally it is time to analyze the execution time performance of the *HFC* search engine. Table 6.8 shows a comparison between performance of Version *1.3*, Version *1.4* and Version *1.5*.

For each version is reported the average execution time of the queries listed in Appendix E.

The average execution time is given by the sum of the average execution time of the various tasks that the search engine must perform in order to implement the ranking model and to respond to a query. These tasks are:

1. *Query expansion.*

   It's the initial task after a user submit a query. The query is tokenized, postagged by the *Stanford Parser*, and then expanded by *Wordnet* in order to get the *termbase* of terms to search.

2. *Thread generation.*

   It's the task the generate the different threads in order to quickly upload term's occurrences. Actually, Version 1.3 used, that should be the monothread version, is in real a version 1.4 with only 1 thread generated.

	HFC Version	1.3	1.4	1.5
	Average Time (T) T=1+2+3+4+5	2,501.12 ms	1,992.61 ms	1,211.17 ms
1	Query Expansion	286.44 ms 11.45%	284.36 ms 14.27%	285.21 ms 23.55%
2	Thread generation	0.40 ms 0.02%	1.88 ms 0.09%	1.79 ms 0.15%
3	Thread execution $\leq$ o3.1+o3.2	2,199.64 ms 87.95%	1,691.59 ms 84.89%	909.45 ms 75.09%
3.1	Occurrences Loading	1,962.52 ms 78.47%	1,639.84 ms 82.30%	839.84 ms 69.34%
3.2	Ranking	237.12 ms 9.48%	75.12 ms 3.77%	76.21 ms 6.29%
4	Thread merging	1.64 ms 0.07%	1.80 ms 0.09%	1.78 ms 0.15%
5	Sorting	13.00 ms 0.52%	12.98 ms 0.65%	12.94 ms 1.07%

TABLE 6.8: Performance comparison between *HFC* version 1.3, 1.4, 1.5

3. *Thread execution.*

   It's the task the perform the occurrences loading (subtask 3.1), and then termset mining and scoring (subtask 3.2). Notice that the average execution time of this task, it is not the sum of the two subtasks that compose it,

but holds the inequality expressed in Table 6.8, because the average time of the main task is given by the the slowest thread, that could not *a priori* being the slowest in both its subtasks.

4. *Thread merging.*

   When all threads complete their task, sub results must be merged in order to get a unique sorting.

5. *Sorting.*

   When all tasks have been performed, each product has a *PRM* score and is in an unsorted list. This task is responsible to sort the list according a descending *PRM* score.

The Table 6.8, shows average time for each task, and under the % share contribution of each task to the final execution time.
From the analysis of data, it is evident how much in version 1.3 and 1.4 the *occurrences loading* affects the final result. This influence decrease considerably in Version 1.5 but it is still around 70%. It is likely to be that the only way to definitely cut off occurrences loading time is to distribute the *HFC* search engine into several machine server.

	HFC version comparison	1.3 vs 1.4	1.4 vs 1.5
	Average Time (T) $T = o1 + o2 + o3 + o4 + o5$	-20.33%	-39.22%
o1	Query Expansion	-0.73%	+0.30%
o2	Thread generation	+370.00%	-4.79%
o3	Thread execution $\leq$ o3.1+o3.2	-23.10%	-46.24%
o3.1	Occurrences Loading	-16.44%	-48.79%
o3.2	Ranking	-68.32%	+1.45%
o4	Thread merging	+9.76%	-1.11%
o5	Sorting	-0.15%	-0.31%

TABLE 6.9: Performance % variation between *HFC* versions

To conclude, The Table 6.9 shows the percentage variation, *task-per-task*, when passing from Version 1.3 to Version 1.4, and from Version 1.4 to Version 1.5.
The passage between 1.3 and 1.4 highlights that multi-threading affects better the *Ranking* task, instead of *Occurrences loading* task. This because each thread share the same single memory bus, and so the improvement is modest. *Thread generation* raising must not impress, as it's global weight is very modest, and then it is only due to reply the same simple operation (i.e. *creating a thread*) 5 times. Same talk for *thread merging* task. The other parameters change according to small statistical fluctuations.
The passage between 1.3 and 1.4 highlights instead an important cut off off *Occurrences loading* time, while instead *Ranking* task changes according to a small statistical fluctuation. The other task does the same. This is because the introduction of the *SSD storage* does not affect the computing power of the server machine, but only its speed in reading data from the mass storage!

Chapter 7

Conclusions

7.1 Conclusions

Working on this project has been challenging and stimulating.
It is nice to think that everything has started by the consideration made over a single a termset that simply summarize the comments of a movie. And it was exciting to see the ranking model evolving in time, and then to see the first results of the work done. Most of all, it was thrilling to see the quest for performance evolving toward a success.

However the project has not concluded yet. There is still some work to to further improve performance, and most of all to verify the quality of the answers of the query engine.

In the following are summarized the main results achieved and the open issues and future works.

7.1.1 Results achieved

After a hard job, overcoming all the issues presented in this thesis, the *HFC* project achieved a certain number of different goals:

- **Ranking model**
  The ranking model designed to rank products exploiting itemset mining techniques represents a new approach in Information Retrieval, that must be investigated more deeply.
- **New approach in developing recommendation system**
  Developing a recommendation system based of user expressed desiderata, and not upon an automatic user profiling, as it happens at the moment in most of the commercial web sites, could open new market possibilities.
- **File reverse index approach**
  It has demonstrated the lack of performance of traditional database, whose management system prevents them to quickly retrieve large amounts of data. The *reverse index* approach based of a set of file instead demonstrates to perfectly meet this lack of performance.
- **Approach feasibility**
  Performance are still not optimal, but decreasing execution time from the initial *more than 30 minutes* to *around one second*, and having further improvements, demonstrates that this approach is feasible and deserves to be investigated more in the future.

7.1.2 Open issues and future work

Although the goals achieved, there are a set of open issues and future development that should be implemented in order to finalize the scope of the project.

- **Improving performance**

  Although the quest for performance demonstrate the feasibility of this approach, performance should be further improved. It is likely to be that the investigation direction is toward using different machine since using a multi thread process to load occurrences cannot permit to substantially improve performance since all threads must share the same memory bus.

- **Extension to *Entity Recognition* and *Linked Data***

  Expanding user queries only on system based on ontologies (*Wordnet*) seems being not perfectly fit for the scope of the project. From a certain point of view, the scope of the project is to intercept user desiderata and try to find it on people's review. Thus, if the user is talking asking for a *good movie about the* `Enigma` *machine*, it is likely to be that a movie about *Alan Turing* could fit the scope. *Entity recognition* is the operation that allows the search engine to recognize `Enigma Machine`, while *Linked Data* is the approach the allows to associate it to *Alan Turing*.

  Extending the *HFC* in order to expands user query in this way as well, seems to be an attempt that should be tried. Appendix D addresses this topic.

- **Design quality evaluation methods**

  Whatever is next improvement or development, it cannot prescind to find proper ways to verify quality of the answer of the HFC search engine. *Ground truth* based methods do not seems to be the appropriate approach. The better approach should be the market! But this would be having the ability to involve lots and lots of people to use the system. Seems to be more a *marketing strategy* than a software issue.

- **Combining HFC with a traditional database**

  At first glance could be reductive, but the ranking model should be applied to extends a *order by* clause of *sql* language.

Traditional database management system are very good in quickly retrieving structured data. On the other side the HFC manifested some time performance issues, because it has been applied to the whole IMDb dataset! But users usually have already a slight idea of what they are looking for. When they ask for a movie about *World War II* probably they tend to discard comedies. Or when they ask for an hotel in Barcelona far away from belfries and with Russian speaking staff, the are implicitly selecting hotels in Barcelona and not in Madrid, for instance, or in wherever other city in the world.

If it would be possible to combine the *HFC* search engine with a traditional database, the latter, by selecting objects, would have the task to reduce the size of the dataset. Then with a reduced dataset to analyze for the *HFC* search engine could be much faster, and sorts the objects selected by the database according to the user desiderata.

Some how, it would be desirable having *sql* code like this:

SELECT Movie

WHERE

Year > 1958 AND

Year < `1972` AND

(Genre="`thriller`" OR Genre="`comedy`")

ORDER BY "`I would like to know more about Greece and Persian wars in ancient times`"

Appendix A

Part-of-speech (POS) Tagging

In *corpus linguistics*, *part-of-speech tagging* (*POS tagging* or *POST*), also called *grammatical tagging* or *word-category disambiguation*, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context, i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph[1].

Computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive *tags*. *POS-tagging* algorithms fall into two distinctive groups: *rule-based* and *stochastic*.

In the mid 1980s, researchers in Europe began to use *Hidden Markov models*[2] (*HMMs*) to disambiguate parts of speech. *HMMs* involve counting cases, and making a table of the probabilities of certain sequences. For example, after an article such as `the`, perhaps the next word is a noun 40% of the time, an adjective 40%, and a number 20%. Knowing this, a program can decide that `can` in `the can` is far

[1] The main source of this Appendix is *Wikipedia*:
http://en.wikipedia.org/wiki/Part-of-speech_tagging

[2] http://en.wikipedia.org/wiki/Hidden_Markov_model

more likely to be a noun than a verb or a modal. The same method can of course be used to benefit from knowledge about following words.

More advanced *higher order HMMs* learn the probabilities not only of *pairs*, but *triples* or even larger sequences. So, for example, after a noun followed by a verb, the next item may be very likely a preposition, article, or noun, but much less likely another verb.

When several ambiguous words occur together, the possibilities multiply. However, it is easy to enumerate every combination and to assign a relative probability to each one, by multiplying together the probabilities of each choice in turn. The combination with highest probability is then chosen. The European group developed *CLAWS*[3], a tagging program that did exactly this, and achieved accuracy in the 93-95% range.

In 1987, *Steven DeRose* in [18] and *Ken Church* in [19], independently *developed dynamic programming* algorithms to solve the same problem in vastly less time. Their methods were similar to the *Viterbi* algorithm known for some time in other fields. *DeRose* used a table of pairs, while *Church* used a table of triples and a method of estimating the values for triples that were rare. Both methods achieved accuracy over 95

Some current major algorithms for part-of-speech tagging include the *Viterbi* algorithm, *Brill Tagger*, *Constraint Grammar*, and the *Baum-Welch* algorithm (also known as the forward-backward algorithm). *Hidden Markov model* and *visible Markov model* taggers can both be implemented using the Viterbi algorithm.

Many machine learning methods have also been applied to the problem of POS tagging. Methods such as *SVM*, *Maximum entropy classifier*, *Perceptron*, and *Nearest-neighbor* have all been tried, and most can achieve accuracy above 95%.

[3] http://ucrel.lancs.ac.uk/claws/

Appendix B

Wordnet

Wordnet has been widely exploit by the *HFC* search engine.
A deep description of *Wordnet* is taken from *Wordnet* site[1] itself:

> **About WordNet**
>
> *WordNet*® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. *WordNet* is also freely and publicly available for download. *WordNet*'s structure makes it a useful tool for computational linguistics and natural language processing.
>
> *WordNet* superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, *WordNet* interlinks not just word forms - strings of letters - but specific senses of words. As a result, words that are found

[1] urlhttp://wordnet.princeton.edu/

in close proximity to one another in the network are semantically disambiguated. Second, *WordNet* labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

Structure

The main relation among words in WordNet is synonymy, as between the words shut and close or car and automobile. Synonyms -words that denote the same concept and are interchangeable in many contexts- are grouped into unordered sets (synsets). Each of WordNet's $117,000$ synsets is linked to other synsets by means of a small number of conceptual relations. Additionally, a synset contains a brief definition (*gloss*) and, in most cases, one or more short sentences illustrating the use of the synset members. Word forms with several distinct meanings are represented in as many distinct synsets. Thus, each form-meaning pair in WordNet is unique.

Relations

The most frequently encoded relation among synsets is the super-subordinate relation (also called *hyperonymy*, *hyponymy* or *ISA* relation). It links more general synsets like (furniture, piece of furniture) to increasingly specific ones like (bed) and (bunkbed). Thus, *WordNet* states that the category furniture includes bed, which in turn includes bunkbed; conversely, concepts like bed and bunkbed make up the category furniture. All noun hierarchies ultimately go up the root node entity. *Hyponymy* relation is transitive: if an armchair is a kind of chair, and if a chair is a kind of furniture, then an armchair is a kind of furniture. *WordNet* distinguishes among Types (common nouns) and Instances (specific

persons, countries and geographic entities). Thus, armchair is a type of chair, *Barack Obama* is an instance of a president. Instances are always leaf (terminal) nodes in their hierarchies.

Meronymy, the part-whole relation holds between synsets like (chair) and (back, backrest), (seat) and (leg). Parts are inherited from their superordinates: if a chair has legs, then an armchair has legs as well. Parts are not inherited *upward* as they may be characteristic only of specific kinds of things rather than the class as a whole: chairs and kinds of chairs have legs, but not all kinds of furniture have legs.

Verb synsets are arranged into hierarchies as well; verbs towards the bottom of the trees (*troponyms*) express increasingly specific manners characterizing an event, as in (communicate)-(talk)-(whisper). The specific manner expressed depends on the semantic field; volume (as in the example above) is just one dimension along which verbs can be elaborated. Others are speed (move-jog-run) or intensity of emotion (like-love-idolize). Verbs describing events that necessarily and unidirectionally entail one another are linked: (buy)-(pay), (succeed)-(try), (show)-(see), etc.

Adjectives are organized in terms of *antonymy*. Pairs of *direct* antonyms like wet-dry and young-old reflect the strong semantic contract of their members. Each of these polar adjectives in turn is linked to a number of *semantically similar* ones: dry is linked to parched, arid, dessicated and bone-dry and wet to soggy, waterlogged, etc. Semantically similar adjectives are *indirect antonyms* of the contral member of the opposite pole. Relational adjectives (*pertainyms*) point to the nouns they are derived from (criminal-crime). There are only few adverbs in *WordNet* (hardly, mostly, really, etc.) as the majority of English adverbs are straightforwardly derived from adjectives via morphological affixation (surprisingly,

strangely, etc.)

Cross-POS relations

The majority of the *WordNet*'s relations connect words from the same *part of speech* (*POS*). Thus, WordNet really consists of four sub-nets, one each for nouns, verbs, adjectives and adverbs, with few cross-POS pointers. Cross-POS relations include the *morphosemantic* links that hold among semantically similar words sharing a stem with the same meaning: observe (verb), observant (adjective) observation, observatory (nouns). In many of the noun-verb pairs the semantic role of the noun with respect to the verb has been specified: (sleeper, sleeping car) is the LOCATION for (sleep) and (painter)is the AGENT of (paint), while (painting, picture) is its RESULT.

Wordnet has also a browser client application in order to allow users to use it standalone. In Figure B.1 an example screenshot.

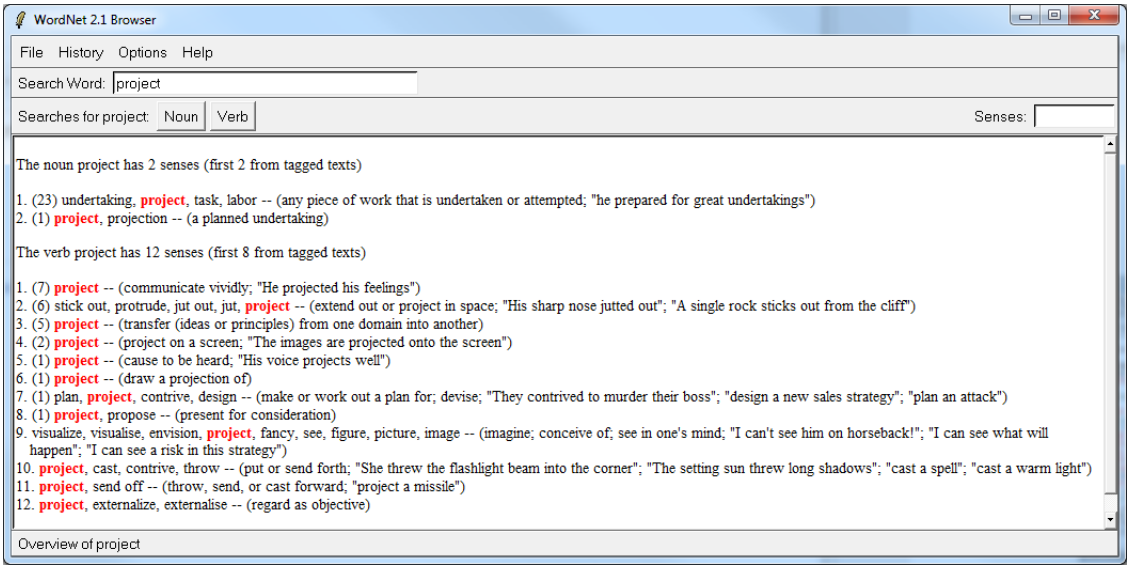

FIGURE B.1: *Wordnet* browser application

Appendix C

Apache Lucene

Lucene is a useful library to perform text indexing and searching tasks.
It was been used to index the *Epinions* dataset and perform queries on it, in order to make a quality comparison with *HFC* answers.
But a question made the research stop: *who guarantee that Lucene indexing is effective?*

From Wikipedia[1]

> *Apache Lucene* is a free/open source information retrieval software library, originally created in Java by Doug Cutting. It is supported by the Apache Software Foundation and is released under the Apache Software License.
>
> While suitable for any application which requires full text indexing and searching capability, *Lucene* has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching.

[1] http://en.wikipedia.org/wiki/Lucene

At the core of *Lucene*'s logical architecture is the idea of a document containing fields of text. This flexibility allows *Lucene*'s API to be independent of the file format. Text from PDFs, HTML, Microsoft Word, and OpenDocument documents, as well as many others (except images), can all be indexed as long as their textual information can be extracted

From Lucene Tutorial[2]

Basic Concepts

Lucene is a full-text search library in Java which makes it easy to add search functionality to an application or website.

It does so by adding content to a full-text index. It then allows you to perform queries on this index, returning results ranked by either the relevance to the query or sorted by an arbitrary field such as a document's last modified date.

The content you add to Lucene can be from various sources, like a SQL/NoSQL database, a file system, or even from websites.

Searching and Indexing Lucene is able to achieve fast search responses because, instead of searching the text directly, it searches an index instead. This would be the equivalent of retrieving pages in a book related to a keyword by searching the index at the back of a book, as opposed to searching the words in each page of the book.

This type of index is called an inverted index, because it inverts a page-centric data structure (page-¿words) to a keyword-centric data structure (word-¿pages).

[2] http://www.lucenetutorial.com/basic-concepts.html

Documents

In Lucene, a Document is the unit of search and index.

An index consists of one or more Documents.

Indexing involves adding Documents to an IndexWriter, and searching involves retrieving Documents from an index via an IndexSearcher.

A Lucene Document doesn't necessarily have to be a document in the common English usage of the word. For example, if you're creating a Lucene index of a database table of users, then each user would be represented in the index as a Lucene Document.

Fields

A Document consists of one or more Fields. A Field is simply a name-value pair. For example, a Field commonly found in applications is title. In the case of a title Field, the field name is title and the value is the title of that content item.

Indexing in Lucene thus involves creating Documents comprising of one or more Fields, and adding these Documents to an IndexWriter.

Searching

Searching requires an index to have already been built. It involves creating a Query (usually via a QueryParser) and handing this Query to an IndexSearcher, which returns a list of Hits.

Queries

Lucene has its own mini-language for performing searches.

The Lucene query language allows the user to specify which field(s) to search on, which fields to give more weight to (boosting), the ability to perform boolean queries (AND, OR, NOT) and other functionality.

Appendix D

Named Entity Recognition & Linked Data

Named-entity recognition (*NER*) (also known as *entity identification*, *entity chunking* and *entity extraction*) is a subtask of information extraction that seeks to locate and classify elements in text into pre-defined categories such as the names of persons, organizations, cities, geographic features, locations, expressions of times, quantities, monetary values, percentages, etc.

NER systems have been created that use linguistic grammar-based techniques as well as statistical models. Hand-crafted grammar-based systems typically obtain better precision, but at the cost of lower recall and months of work by experienced computational linguists. Statistical *NER* systems typically require a large amount of manually annotated training data. Semi supervised approaches have been suggested to avoid part of the annotation effort.

Research indicates that even state-of-the-art *NER* systems are brittle, meaning that *NER* systems developed for one domain do not typically perform well on other domains.

In natural language processing, *entity linking*, *named entity disambiguation* or *named entity normalization* (*NEN*) is the task of determining the identity of entities mentioned in text. It is distinct from *named entity recognition* (*NER*) in that it identifies not the occurrence of names (and a limited classification of those), but their *reference*.

Entity linking needs a *knowledge base* of entities to which names can be linked. A popular choice for *entity linking* on open domain text is Wikipedia, and when that is used, the process may be called *wikification*.

Any *entity linking* algorithm must battle the inherent ambiguity that even names have. Various approaches to tackle this problem have been tried. In the seminal approach of *Milne and Witten* ([20]), supervised learning is employed using the anchor texts of Wikipedia itself as the training data. The training data also can be collected by an automatic approach based on unambiguous synonyms.

Entity linking has been suggested as a way to automate the construction of a *Semantic Web*. It has been used to improve the performance of information retrieval systems.

The *Semantic Web* is a collaborative movement led by international standards body the *World Wide Web Consortium* (*W3C*), directed by *Tim Berners-Lee*[1] The standard promotes common data formats on the World Wide Web. By encouraging the inclusion of semantic content in web pages, the *Semantic Web* aims at converting the current web, dominated by unstructured and semi-structured documents into a `web of data`. The *Semantic Web* stack builds on the W3C's *Resource Description Framework* (*RDF*).

[1] http://en.wikipedia.org/wiki/Tim_Berners-Lee

Berners-Lee et al. in [21] and [22] call the resulting network of *Linked Data* the *Giant Global Graph* in contrast to the HTML-based World Wide Web.

In Figure D.1 and D.2 is shown Linked Data evolution graph from 2011 to 2014,

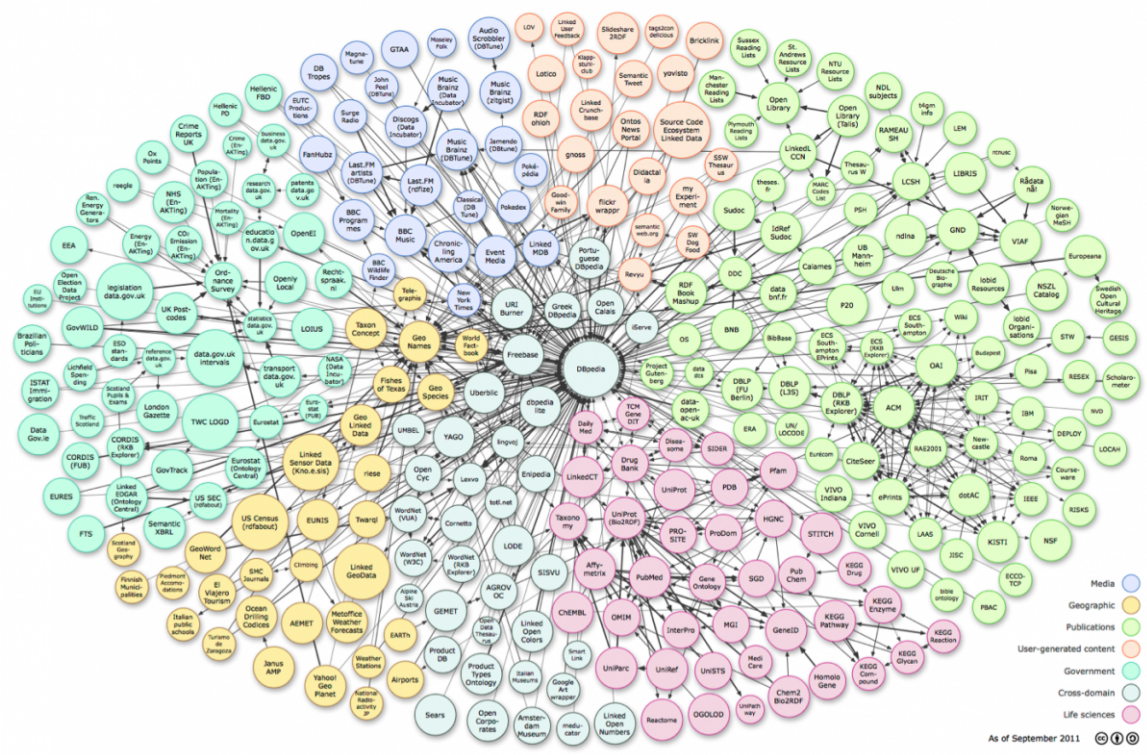

FIGURE D.1: Web of *Linked data* in September 2011

Tim Berners-Lee posits that if the past was document sharing, the future is data sharing. His answer to the question of `how` provides three points of instruction.

1. a URL should point to the data.
2. anyone accessing the URL should get data back.
3. relationships in the data should point to additional URLs with data

A complete description description of *Linked Data* is given from the *Semantic Web* page on *W3C* website[2]:

[2] http://www.w3.org/standards/semanticweb/data

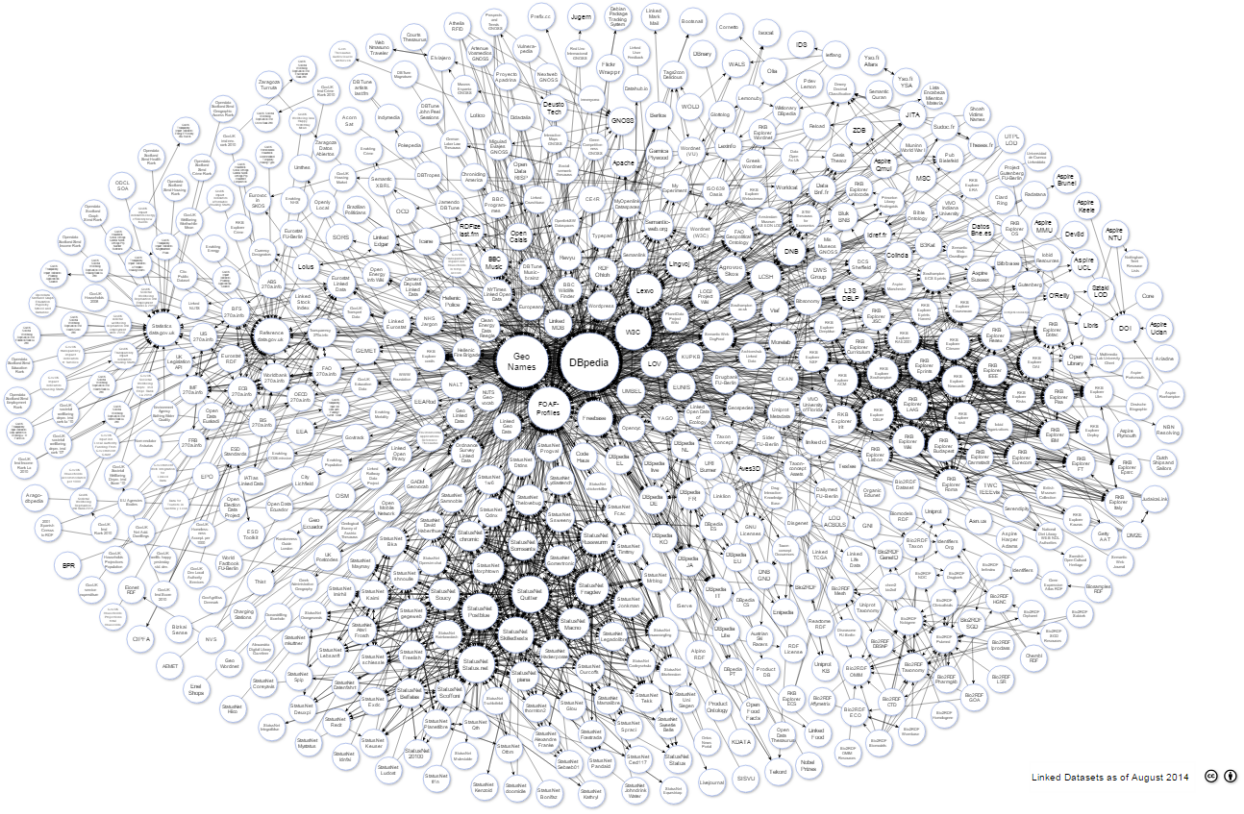

FIGURE D.2: Web of *Linked data* in August 2014

What is *Linked Data*?

The *Semantic Web* is a *Web of Data* - of dates and titles and part numbers and chemical properties and any other data one might conceive of. The collection of Semantic Web technologies (*RDF*, *OWL*, *SKOS*, *SPARQL*, etc.) provides an environment where application can query that data, draw inferences using vocabularies, etc. However, to make the Web of Data a reality, it is important to have the huge amount of data on the Web available in a standard format, reachable and manageable by Semantic Web tools. Furthermore, not only does the Semantic Web need access to data, but relationships among data should be made available, too, to create a Web of Data (as opposed to a sheer collection

of datasets). This collection of interrelated datasets on the Web can also be referred to as *Linked Data*. To achieve and create *Linked Data*, technologies should be available for a common format (*RDF*), to make either conversion or on-the-fly access to existing databases (relational, *XML*, *HTML*, etc). It is also important to be able to setup query endpoints to access that data more conveniently. *W3C* provides a palette of technologies (*RDF, GRDDL, POWDER, RDFa, the upcoming R2RML, RIF, SPARQL*) to get access to the data.

What is *Linked Data* Used For?

Linked Data lies at the heart of what Semantic Web is all about: large scale integration of, and reasoning on, data on the Web. Almost all applications listed in, say collection of Semantic Web Case Studies and Use Cases are essentially based on the accessibility of, and integration of *Linked Data* at various level of complexities.

Examples

A typical case of a large *Linked Dataset* is *DBPedia*, which, essentially, makes the content of *Wikipedia* available in RDF. The importance of *DBPedia* is not only that it includes Wikipedia data, but also that it incorporates links to other datasets on the Web, e.g., to *Geonames*. By providing those extra links (in terms of RDF triples) applications may exploit the extra (and possibly more precise) knowledge from other datasets when developing an application; by virtue of integrating facts from several datasets, the application may provide a much better user experience.

Appendix E

Testing queries

Here are listed the set of queries used during the tests with the *IMDb* large dataset. It is a set of twenty-five different sentences in natural language (*english*). Some are just movie titles, others recalls actors. Some sentences are taken directly from movie reviews, others are citation of famous movie quotes.
In bold font, searching terms are highlighted. The rest of the text is considered *stopwords*, and so discarded. Notice that some strictly contextual terms are considered as stopwords. In this *IMDb-movie* context words *actor*, *film*, *movie* and their declinations are *stopwords*.

1. **Great funny hilarious jokes**[1]
2. **Eddie Murphy**[2]'s ***Raw***[3]
3. I want to know more about the **history** of **ancient Greece** and the **Persian wars**

[1] During test with termset extraction, this has the first termset noticed to well describe a movie: *Eddie Murphy's Raw*.
[2] http://www.imdb.com/name/nm0000552
[3] (1987) http://www.imdb.com/title/tt0092948

4. I want to know about the **history** of **Greece** and **Persian wars** with **Sparta** and **Athens**
5. I want a *movie* with **great funny hilarious jokes**
6. A **nice drama** for the **afternoon**
7. **Saturday night pajama party**
8. I have a **dinner** with my **girlfriend** and I want to **see** a **romantic** *movie*
9. My **kids** are too **quiet** and i **need** to **thrill** them
10. My **granny** is **getting sad** and I want to **make** her **laugh**
11. The **joke** was **sad** but the **landscapes** were **gorgeous**
12. **Better** not to say, but this **jokes** was not so **good** and **funny**
13. I want a *movie* with **fast cars** and **adrenaline**
14. *When* ***Harry met Sally***[4]
15. *The* ***Lord*** *of the* ***rings***[5]
16. *All* ***quiet*** *on* ***western front***[6]
17. *The* ***Sting***[7] with **Paul Newman**[8] and **Robert Redford**[9]
18. ***Butch Cassidy*** *and the* ***Sundance Kid***[10] with **Paul Newman** and **Robert Redford**[11]

[4](1989) http://www.imdb.com/title/tt0098635
[5](2001) http://www.imdb.com/title/tt0120737
[6]From the novel of *Nobel* prize awarded *Erich Maria Remarque* (1930) http://www.imdb.com/title/tt0020629
[7](1974) http://www.imdb.com/title/tt0070735
[8]http://www.imdb.com/name/nm0000056
[9]http://www.imdb.com/name/nm0000602
[10](1969) http://www.imdb.com/title/tt0064115
[11]*Robert Redford* named his famous *Sundance Film Festival* after his character in this movie

19. My **favorite French heartthrob Sami Frey** is **well-cast** as a **cruise ship tour guide**[12]

20. **Observations** of the **award show**: some of the **winners** had to **walk** a **great distance**[13]

21. I would like to know something about **Prussia**

22. ***Attack ships*** *on* ***fire*** *off the* ***shoulder*** *of* ***Orion***[14]

23. *All those* ***moments*** *will be* ***lost*** *in* ***time*** *like* ***tears*** *in* ***rain***[15]

24. A **neo-realistic italian** *movie* with *actress* **Sophia Loren**[16]

25. What are **sword** and **sandal** *films*?

26. *Any question?*[17] *;-)*

[12] From a movie review
[13] From a movie review
[14] Part of *Roy Batty* final speech of in *Blade Runner*. (1982) http://www.imdb.com/title/tt0083658
[15] Part of *Roy Batty* final speech of in *Blade Runner*. (1982) http://www.imdb.com/title/tt0083658
[16] http://www.imdb.com/name/nm0000047
[17] *See you all at final dissertation. Hope you liked the thesis and presentation will be good.*

Bibliography

[1] Paolo Fosci and Giuseppe Psaila. Toward a product search engine based on user reviews. *DATA*, pages 223–228, July 2012.

[2] Paolo Fosci and Giuseppe Psaila. Finding the best source of information by means of a socially-enabled search engine. In *KES*, pages 1253–1262, 2012.

[3] Paolo Fosci, Giuseppe Psaila, and Marcello Di Stefano. Hints from the crowd: A novel nosql database. *Model and Data Engineering*, pages 118–131, 2013.

[4] Paolo Fosci, Giuseppe Psaila, and Marcello Di Stefano. The hints from the crowd project. In *Database and Expert Systems Applications*, pages 443–453. Springer Berlin Heidelberg, 2013.

[5] Alfredo Cuzzocrea, Marcello Di Stefano, Paolo Fosci, and Giuseppe Psaila. Effectively and efficiently supporting crowd-enabled databases via nosql paradigms. In *Proceedings of the 3rd International Workshop on Semantic Search Over the Web*, SS@ '13, pages 7:1–7:5, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2483-0. doi: 10.1145/2509908.2509914. URL http://doi.acm.org/10.1145/2509908.2509914.

[6] Alfredo Cuzzocrea, Marcello Di Stefano, Paolo Fosci, and Giuseppe Psaila. Enhanced query processing for nosql crowdsourcing systems. In *Proceedings of the 6th IEEE International Conference of Soft Computing and Pattern Recognition (SoCPaR 2014)*. SoCPaR, 2014.

[7] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[8] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. *Stanford InfoLab*, 1999.

[9] Lei Tang and Huan Liu. Community detection and mining in social media. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 2(1):1–137, 2010.

[10] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, September 1994.

[11] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286, 1997.

[12] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[13] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[14] Jennifer A Jacobi, Eric A Benson, and Gregory D Linden. Recommendation system, March 15 2011. US Patent 7,908,183.

[15] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston,

et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[16] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6): 914–925, December 1993.

[17] Jonathan Schler, Moshe Koppel, Shlomo Argamon, and James W Pennebaker. Effects of age and gender on blogging. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, pages 199–205, 2006.

[18] Steven J. DeRose. Grammatical category disambiguation by statistical optimization. *Comput. Linguist.*, 14(1):31–39, January 1988. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=49084.49087.

[19] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, ANLC '88, pages 136–143, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics. doi: 10.3115/974235.974260. URL http://dx.doi.org/10.3115/974235.974260.

[20] David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 509–518, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3. doi: 10.1145/1458082.1458150. URL http://doi.acm.org/10.1145/1458082.1458150.

[21] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[22] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96–101, 2006.

[23] Christian Borgelt. Efficient implementations of apriori and eclat. In *FIMI-2003 Int. Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida (USA)*, November 2003.

[24] G. Bordogna and G. Psaila. Customizable flexible querying for classical relational databases. *J. Galindo (ed.) Handbook of Research on Fuzzy Information Processing in Databases*, 2008.

[25] G. Bordogna, A. Campi, G. Psaila, and S. Ronchi. A language for manipulating clustered web documents results. *Proceedings of CIKM08, CA, USA, November, 2008*, 2002.

[26] G. Bordogna and G. Psaila. Soft aggregation in flexible databases querying based on the vector p-norm. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 17:25–40, 2009.

[27] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Proceedings of the 22st VLDB Conference*, Bombay, India, September 1996.

[28] F. Crestani, M. Dunlop, M. Jones, S. Jones, and S. Mizzaro (eds.). International journal of personal and ubiquitous computing, special issue on interactive mobile information access, results. *Proceedings of Springer Verlag, 2006*, 2006.

[29] Giuseppe Psaila and Stefano Crespi-Reghizzi. Adding semantics to xml. 1999.

[30] O. Buyukkokten, O. Kaljuvee, H. Garcia-Molina, A. Paepcke, and T. Wingrad. Efficient web browsing on handheld devices using page and form summarization. *ACM Trans. Inf. Syst*, 20(1):82–115, 1999.

[31] Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, and Riccardo Torlone. *Database systems.* Mc Graw-Hill, 1999.

[32] M. Noirhomme-Fraiture, F. Randolet, and L. Chittaro an G. Custinne. Data visualizations on small and very small screens. *in Proceedings of Applied Stocastics Models and Data Analysis*, 2005.

[33] G. Buchnan, M. Jones, and G. Marsden. Exploring small screen digital library access with the greenstone digital library. *in Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries, LNCS 2458, Springer Verlag*, pages 583–596, 2003.

[34] S. K. Card, J. D. Mackinlay, and B. Shneiderman. Readings in information visualization: Using vision to think. *Morgan Kaufmann Publishers Inc. San Francisco, CA*, 1999.

[35] H. Chen and S. Dumais. Bringing order to the web: Automatically categorizing search results. *Proceedings of the SIGCHI conference on Human factors in computing systems*, IR-76:145–152, 2000.

[36] W. Chung, H. Chen, and J.F. Jr. Nunamaker. Business intelligence explorer: a knowledge map framework for discovering business intelligence on the web. *System Sciences*, Proceedings of the 36th Annual Hawaii International Conference on System Sciences:10, 2003.

[37] T. Coates, D. Connolly, D. Dack, L. Daigle, R. Denenberg, P. Grosso M. Durst, S. Hawke, R. Iannella, G. Klyne, L. Masinter, M. Mealling, M. Needleman, and N. Walsh. Uris, urls, and urns: Clarifications and recommendations 1.0. *Technical report, World Wide Web Consortium, URI Planning Interest Group W3C/IETF*, Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2001.

[38] M. A. Hearst and J. O. Pederson. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. *Proceedings of the Conference on Research and Development in Information Retrieval*, 1996.

[39] N. Kampanya, R. Shen, S. Kim, C. North, and E. A. Fox. Citiviz: A visual user interface to the citidel system. *LNCS, Springer Verlag*, 3232:122–133, 2004.

[40] A. V. Leouski and W. B. Croft. An evaluation of techniques for clustering search results. *Technical Report of the Department of Computer Science f University of Massachusetts at Amherst*, IR-76:122–133, 1996.

[41] E. Staley and M. Twidale. Graphical interfaces to support information search. *Technical report, University of Illinois*, 2000.

[42] L. Zadeh. Fuzzy sets. *Information and control*, 8:338–353, 1965.

[43] S. Osinski. An algorithm for clustering of web search results. Master's thesis, Department of Computing Science, Poznan' University of Technology, http://project.carrot2.org/publications/osinski-2003-lingo.pdf, 2003.

[44] R. Agrawal and E.L. Wimmers. A framework for expressing and combining preferences. In *Proceedings of ACM SIGMOD 2000 International Conference on Management of Data, Dallas*, pages 297–306, 2000.

[45] R. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *Proceedings of CIDR 2003 First Biennal Conference on Innovative Data Systems Research, Asilomar, CA, USA*, 2003.

[46] G. Bordogna and G. Pasi. A flexible approach to evaluating soft conditions with unequal preferences in fuzzy databases. *Journal of Intelligent Systems*, 22:665–689, 2007.

[47] S. Borzsonyi and D. Kossmann K. Stocker. The skyline operator. In *Proceedings of IEEE ICDE 2001 17th International Conference on Data Engineering*, pages 421–430, 2001.

[48] P. Bosc and O. Pivert. Sqlf: a relational database language for fuzzy querying. *IEEE Transactions On Fuzzy Systems*, 3(1), 1995.

[49] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems*, 27(2):153–187, June 2002.

[50] B. Buckles and F. Petry. A fuzzy model for relational databases. *International Journal of Fuzzy Sets and Systems*, 7:213–226, 1982.

[51] J. Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466, 2003.

[52] D. Dubois and H. Prade. Using fuzzy sets in flexible querying: why and how ? *Andreasen T, Christiansen H, Larsen HL (eds) Flexible query answering systems*, pages 45–60, 1997.

[53] D. Dubois, H. Prade, and C. Testemale. Weighted fuzzy pattern matching. *International Journal of Fuzzy Sets and Systems*, 28:313–331, 1988.

[54] J. Galindo, J.M. Medina, and G.M.C. Aranda. Querying fuzzy relational databases through fuzzy domain calculus. *International Journal of Intelligent Systems*, 14:375–411, 1999.

[55] J. Galindo, A. Urrutia, and M. Piattini. *Fuzzy Databases, Modeling, Design and Implementation*. Idea Group, USA, 2006.

[56] M. Grabisch, S. A. Orlovski, and R. R. Yager. Fuzzy aggregation of numerical preferences. *R. Slowinski (ed.) Fuzzy Sets in Decision Analysis, Operations Research and Statistics*, pages 31–68, 1998.

[57] T. Ichikawa and M. Hirakawa. ARES: a relational database with the capability of performing flexible interpretation of queries. *IEEE Transactions On Software Engineering*, 12(5):624–634, 1986.

[58] J. Kacprzyk and S. Zadrosny. FQUERY for Access: Fuzzy querying for windows-based DBMS. *P. Bosc and J. Kacprzyk (ed.) Fuzziness in database management systems*, 1995.

[59] J. Kacprzyk and S. Zadrosny. Implementation of OWA operators in fuzzy querying for Microsoft Access. *R.R. Yager and J. Kacprzyk (ed.) The ordered weighted averaging operators: theory and applications*, pages 293–306, 1997.

[60] J. Kacprzyk and A. Ziolkowski. Database queries with fuzzy linguistic quantifiers. *IEEE Transactions On Systems, Man and Cybernetics*, 16:474–479, 1986.

[61] W. Kiessling. Foundation of preferences in databases systems. In *Proceedings of VLDB 2002 28th International Conference on Very Large Databases, Hong kong, China*, 2002.

[62] W. Kiessling and G. Kostler. Preference SQL - design, implementation, experiences. In *Proceedings of VLDB 2002 28th International Conference on Very Large Databases, Hong kong, China*, 2002.

[63] M. Lacroix and P. Lavency. Preferences: Putting more knowledge into queries. In *Proceedings of VLDB 1987 13th International Conference on Very Large Databases, Brighton, GB*, pages 217–225, 2002.

[64] H.L. Larsen. Importance weighted OWA aggregation of multicriteria queries. In *Proceedings of NAFIPS 1999 18th International Conference of the North American Fuzzy Information Processing Society, New York, USA*, pages 740–744, 1999.

[65] A. Motro. VAGUE: A user interface to relational databases that permits vague queries. *ACM Transactions on Information Systems*, 6(3):187–214, July 1988.

[66] E.E. Petry. *Fuzzy Databases*. Kluwer Academic Publisher, USA, 1996.

[67] G. Salton. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, USA, 1983.

[68] G. Salton, E. Fox, and H Wu. Extended boolean information retrieval. *Communications of the ACM*, 26(12):1022–1036, 1983.

[69] V. Tahani. A conceptual framework fro query processing: a step toward very intelligent database systems. *Information Processing and Management*, 13: 289–303, 1977.

[70] O. Zamir and O. Etzioni. Grouper: a dynamic clustering interface to web search results. *Proceedings of the 8th International World Wide Web Conference*, 1999.

[71] A. Takaci and S. Skrbic. Comparing priority, weighted and queries with threshold in PFSQL. In *Proceedings of SISY 2007 Fifth International Symposium on Intelligent Systems and Informatics, Subotica, Serbia*, pages 77 – 80, 2007.

[72] L. Tineo. Extending rdbms for allowing fuzzy quantified queries. In *Proceedings of DEXA 2000 11th International Conference on Database and Expert Systems Applications, Greenwich, London, UK*, pages 407–416, 2000.

[73] J. Kacprzyk and S. zadrozny. On combining intelligent querying and data mining using fuzzy logic concepts. In *in G. Bordogna, G. Pasi (eds.), Recent Issues on Fuzzy Databases*, pages 67–81. Physica Verlag, 2000.

[74] C. Chen, Q. Wei, and E.E. Kerre. Fuzzy data mining: Discovery of fuzzy generalized association rules. In *in G. Bordogna, G. Pasi (eds.), Recent Issues on Fuzzy Databases*, pages 45–66. Physica Verlag, 2000.

[75] R.Srikant and R.Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, San Jose, California, May 1996.

[76] M. A. W. Houtsma and A. Swami. Set-oriented mining for association rules in relational databases. In *11th International Conference on Data Engineering*, Taipei, Taiwan, March 6-10 1995.

[77] T. Hong and Y. Lee. An overview of mining fuzzy association rules. *H. Bustince, F. Herrera and J. Montero (eds.), Fuzzy Sets and Their Extensions: Representation, Aggregation and Models*, Studies in Fuzziness and Soft Computing, Volume 220:397–410, 2008.

[78] T. Hong, C. Kuo, and S. Chi. Mining association rules from quantitative dat. *Intelligent Data Analysis*, 2(5), 1999.

[79] G. Chen and Q. Wei. Fuzzy association rules and the extended mining algorithms. *Information Sciences*, 147(1-4), 2002.

[80] D. Dubois, H. Prade, and T. Sudkamp. A discussion ofindices for the evaluation off uzzy associations in relational databases. In *Proceedings of 11th IFSA World Congress, Istambul*, pages 111–118, 2003.

[81] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[82] C.M. Kuok, A.W.-C. Fu, and M.H.Wong. Mining fuzzy association rules in databases. *SIGMOD Record*, 27 (1):41–46, 1998.

[83] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of SIGMOD-93, the ACM SIGMOD Int. Conference on Management of Data*, pages 207–216, Washington, D.C., 1993. British Columbia.

[84] T. Sudkamp. Examples, counterexamples, and measuring fuzzy associations. *Fuzzy Sets and Systems*, 149:57–71, 2005.

[85] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD Record*, 26 (2):255–264, 1997.

[86] Massimo Marchiori. The quest for correct information on the web: Hyper search engines. *Computer Networks and ISDN Systems*, 29(8):1225–1235, 1997.

[87] H. Lee and H.L. Kwang. An extension of association rules using fuzzy sets. In *Proceedings of IFSA-97 World Congress, Prague, Czech Republic*, 1997.

[88] W.H. Au and K.C.C. Chan. Mining fuzzy association rules. In *Proceedings of 6th Int. Conf. Information Knowledge Management, Las Vegas, NV, USA*, pages 209–215, 1997.

[89] R.R. Yager D. Rasmussen. Summarysql - a fuzzy tool for data mining. *Intelligent Data Analysis*, 1 (1-4):49–58, 1997.

[90] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In Kim et al. [91], pages 168–177. ISBN 1-58113-888-1.

[91] Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, 2004. ACM. ISBN 1-58113-888-1.

[92] D Tony Liu and X William Xu. A review of web-based product data management systems. *Computers in industry*, 44(3):251–262, 2001.

[93] Pimwadee Chaovalit and Lina Zhou. Movie review mining: a comparison between supervised and unsupervised classification approaches. In *HICSS* DBL [94]. ISBN 0-7695-2268-8.

[94] *38th Hawaii International Conference on System Sciences (HICSS-38 2005), CD-ROM / Abstracts Proceedings, 3-6 January 2005, Big Island, HI, USA*, 2005. IEEE Computer Society. ISBN 0-7695-2268-8.

[95] Kushal Dave, Steve Lawrence, and David M. Pennock. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *WWW*, pages 519–528, 2003.

[96] Wonyoung Kim, Joonsuk Ryu, Kyu Il Kim, and Ung-Mo Kim. A method for opinion mining of product reviews using association rules. In Sohn et al. [97], pages 270–274. ISBN 978-1-60558-710-3.

[97] Sungwon Sohn, Ling Chen, Soonwook Hwang, Kyungeun Cho, Shigeo Kawata, Kyhyun Um, Franz I. S. Ko, Kae-Dal Kwack, Jong Hyung Lee, Gang Kou, Kiyoshi Nakamura, Alvis Cheuk M. Fong, and Patrick C. M. Ma, editors. *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human 2009, Seoul, Korea, 24-26 November 2009*, volume 403 of *ACM International Conference Proceeding Series*, 2009. ACM. ISBN 978-1-60558-710-3.

[98] Bing Liu, Minqing Hu, and Junsheng Cheng. Opinion observer: analyzing and comparing opinions on the web. In Ellis and Hagino [101], pages 342–351. ISBN 1-59593-046-9.

[99] Li Zhuang, Feng Jing, and Xiaoyan Zhu. Movie review mining and summarization. In Yu et al. [100], pages 43–50. ISBN 1-59593-433-2.

[100] Philip S. Yu, Vassilis J. Tsotras, Edward A. Fox, and Bing Liu, editors. *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6-11, 2006*, 2006. ACM. ISBN 1-59593-433-2.

[101] Allan Ellis and Tatsuya Hagino, editors. *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, 2005. ACM. ISBN 1-59593-046-9.

[102] Dongjoo Lee, Ok-Ran Jeong, and Sang goo Lee. Opinion mining of customer feedback data on the web. In Kim and Choi [103], pages 230–235. ISBN 978-1-59593-993-7.

[103] Won Kim and Hyung-Jin Choi, editors. *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication, ICUIMC 2008, Suwon, Korea, January 31 - February 01, 2008*, 2008. ACM. ISBN 978-1-59593-993-7.

[104] Jeff J. Sandvig, Bamshad Mobasher, and Robin D. Burke. Robustness of collaborative recommendation based on association rule mining. In Konstan et al. [105], pages 105–112. ISBN 978-1-59593-730-8.

[105] Joseph A. Konstan, John Riedl, and Barry Smyth, editors. *Proceedings of the 2007 ACM Conference on Recommender Systems, RecSys 2007, Minneapolis, MN, USA, October 19-20, 2007*, 2007. ACM. ISBN 978-1-59593-730-8.

[106] Minqing Hu and Bing Liu. Mining opinion features in customer reviews. In McGuinness and Ferguson [107], pages 755–760. ISBN 0-262-51183-5.

[107] Deborah L. McGuinness and George Ferguson, editors. *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 2004. AAAI Press / The MIT Press. ISBN 0-262-51183-5.

[108] Duy Khang Ly, Kazunari Sugiyama, Ziheng Lin, and Min-Yen Kan. Product review summarization based on facet identification and sentence clustering. *CoRR*, abs/1110.1428, 2011.

[109] Hecht Robin and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *CSC-2011 International Conference on Cloud and Service Computing, Hong Kong, China*, pages 336–341, December 2011.

[110] R. Cattell. Scalable sql and nosql data stores. *SIGMOD Record*, 39 (4):12–27, 2011.

[111] C. Strauch. Nosql databases. *http://www.christof-strauch.de/nosqldbs.pdf*, 2011.

[112] Tom Gruber. Collective knowledge systems: Where the social web meets the semantic web. *J. Web Sem.*, 6(1):4–13, 2008.

[113] Jill Freyne, Rosta Farzan, Peter Brusilovsky, Barry Smyth, and Maurice Coyle. Collecting community wisdom: integrating social search & social navigation. In Chin et al. [114], pages 52–61. ISBN 1-59593-481-2.

[114] David N. Chin, Michelle X. Zhou, Tessa A. Lau, and Angel R. Puerta, editors. *Proceedings of the 2007 International Conference on Intelligent User Interfaces, January 28-31, 2007, Honolulu, Hawaii, USA*, 2007. ACM. ISBN 1-59593-481-2.

[115] Jonathan Schler, Moshe Koppel, Shlomo Argamon, and James W. Pennebaker. Effects of age and gender on blogging. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs* DBL [116], pages 199–205.

[116] *Computational Approaches to Analyzing Weblogs, Papers from the 2006 AAAI Spring Symposium, Technical Report SS-06-03, Stanford, California, USA, March 27-29, 2006*, 2006. AAAI.

[117] Souvik Debnath, Niloy Ganguly, and Pabitra Mitra. Feature weighting in content based recommendation system using social network analysis. In *Proceedings of the 17th international conference on World Wide Web*, pages 1041–1042. ACM, 2008.