Politecnico di Milano
V Facoltà di Ingegneria
Corso di laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

RECOMMENDATION FOR NAVIGATION PATHS OVER MULTI-DOMAIN DATA ON THE WEB

Relatore: Prof. Marco BRAMBILLA

Tesi di Laurea di:
Federico GHIGINI Matr. 755483
Nicola LUZZARA Matr. 755659

Anno Accademico 2011-2012

Ringraziamenti

Vorremmo ringraziare innanzitutto il professor Brambilla che ci ha consigliato questo lavoro di tesi e sostenuto durante tutta la stesura di esso.

Ringraziamo inoltre le nostre famiglie e i nostri amici che ci hanno sostenuto e sopportato durante questi anni di studi.

Nicola:

Vorre ringraziare tutta la mia famiglia, i nonni Maria, Ada e Angelo e i genitori Lia e Mauro che mi ha sostenuto e sopportato durante la mia crescita universitaria cercando sempre di venire incontro a ogni mio problema o dubbio con consigli semplici e sempre corretti.

I miei amici di lunga data Azza, Barbara, Monica, Nicolas che ogni volta mi toglievano dalla mente esami, libri da studiare ecc... grazie alla loro simpatia, intraprendenza e voglia di divertirsi sempre e comunque.

Un grazie di cuore all'Amico e collega Allegretti Marco che ha condiviso con me i migliori anni passati al Politecnico e ora condivide la mia passione per la pesca. Grazie Alle.

Ringrazio Tania per tutto ciò che ha fatto per me, per la pazienza avuta ogni giorno e per la forza e fiducia incondizionata che mi ha dato. Spero un giorno di poterti ridare tutto ciÃ²$chemihaidatoperfarmiarrivarefinoaqui$.

Ringrazio Visino, Stefano M. e Stefano R. per la loro amicizia e le infinite giornate di pesca passate che mi hanno fatto rilassare soprattutto nei periodi vicino ad esami.

Ringrazio Dino Bernardi che quel lontano giorno di circa 13 anni fa mi regalò il mio primo computer mostrandomi un mondo che non avevo ancora considerato, l'informatica, che poi Ã¨ entrata nella mia quotidianitÃ .

Infine, ultimo perchè gli ultimi saranno i primi, il mio collega di tesi Federico Ghigini con il quale ho avuto la fortuna di condividire oltre questa tesi, progetti e esami, soprattutto una grande amicizia legata anche forse alla sua sopportazione verso di me per i due anni stupendi di residenza a Milano.

Federico:

Ringrazio i miei genitori e mia nonna Mariuccia per avermi sostenuto durante questi anni al Politecnico di Milano.

Ringrazio Lorenzo e Marco, i due coinquilini con i quali ho diviso l'appartamento a Milano. Grazie per avermi fatto scoprire la cucina mantovana e la buona musica dei Pink Floyd.

Ringrazio Caterina, Matteo, Massimo e Silvia con cui ho condiviso i viaggi sul treno modello 668 verso Cremona.

Ringrazio tutti gli amici che ho conosciuto durante questi anni. A Castello ho incontrato Luca, Federico, Chiara, Stefania, Mariana e Martina.

Suonando ho incontrato Paolo, che mi ha insegnato ad amare la batteria e

che con pazienza mi ha seguito fino al mio primo saggio. Con lui ho conosciuto Zucconi, Tuta, Daniele e Cristina, con cui ho condiviso la passione per la musica.

Frequentando Cremona ho incontrato Angelo, Giulia, Claudio, e tutti gli altri compagni del politecnico.

Inoltre ho avuto la fortuna di conoscere Armando, Vincenzo e sopratutto Valeria, che ha scelto di consacrare la sua vita alla preghiera diventando suora.

Ho conosciuto grazie a Nicola tutti i suoi amici di Mantova: Nicolas, Barbara, Marco, Stefano e Stefano, Monica, Tania che mi dimostrano come la distanza geografica non è per nulla un limite.

Grazie a loro ho conosciuto Marta, Elisabetta, Michela e Luisa che mi hanno mostrato la loro passione per gli animali di qualsiasi tipo e stazza!

A Milano ho avuto modo di incontrare tanti ragazzi: Valentino, Federico, Gaia, Ledia, Lucrezia, e tutti gli altri amici di biomedica. Questi sono solo alcuni e sono sicuro di averne tralasciati altrettanti: a voi tutti grazie per avermi sopportato e per avermi fatto crescere!

Voglio ringraziare Nicola, che non solo mi ha affiancato nella stesura di questo testo ma sopratutto mi ha accolto come amico, mi ha presentato alla sua compagnia e da cui ho imparato molto.

Infine desidero ricordare Matteo, un amico che purtroppo mi ha accompagnato solamente fino all'università.

Ai nostri genitori ...

Contents

Sommario **13**

Abstract **15**

1 Introduction **17**
1.1 Context . . . 17
1.2 Problem addressed . . . 18
1.3 Proposed solution . . . 18
1.4 Summary of the work done . . . 19

2 Background & Related Work **21**
2.1 Related work . . . 21
2.2 SeCo . . . 23
2.2.1 Liquid Queries in Search Computing . . . 24
2.3 Recommender systems . . . 25
2.3.1 Personalization . . . 25
2.3.2 Search-Based . . . 25
2.3.3 Category-Based . . . 27
2.3.4 Collaborative-Filtering . . . 28
2.3.5 Clustering . . . 29
2.3.6 Association Rule . . . 31
2.3.7 Information Filtering . . . 33
2.3.8 Crowd . . . 35
2.3.9 Overview . . . 35
2.4 Apache Mahout . . . 37
2.4.1 Mahout - Taste . . . 38

3 Recommendation algorithms **41**
3.1 Defining the problem . . . 41
3.1.1 Overview . . . 42
3.2 Recommendation algorithms . . . 47
3.2.1 Purely statistic approach . . . 47
3.2.2 Machine-learnig based approach . . . 47

3.2.3 Refined approach for multi-domain search 48

4 Implementation and experiments **51**
4.1 Algorithms implementation 51
4.1.1 Logs of exploratory search activities 51
4.1.2 From logs to suggestion 58
4.2 Experiments 65
4.2.1 Random exploration 65
4.2.2 ad-hoc Exploration 66
4.3 Evaluation 67

5 Conclusions **69**
5.1 Summary of the work 69
5.2 Conclusion 70
5.3 Future work & extensions 70

List of Figures

2.1 expedia.com home page 22
2.2 expedia.com search of a travel + hotel + car 23
2.3 search-based relevance on amazon.com 26
2.4 search-based by popularity on amazon.com 26
2.5 category based on amazon.com 27
2.6 category based - categorization 28
2.7 category based second-level of categorization 28
2.8 example of collaborative filtering 29
2.9 example of clustering 30
2.10 example of clustering 30
2.11 example of clustering 31
2.12 example of association rule 32
2.13 Word frequency calculated on 7 book's titles 34
2.14 TFIDF calculated on 7 book's titles 34
2.15 Mahout-Taste architecture 38

3.1 space of services 42
3.2 path on the space of services 43
3.3 data extraction step-1 44
3.4 data extraction step-2 45
3.5 save instances from service 46
3.6 This graps is an example scenario with seven services 48
3.7 lookup-table 49
3.8 example of path 49

4.1 path over two services 52
4.2 path over one service 53
4.3 database structure 53
4.4 random generation of logs 54
4.5 random generation of logs detailed 56
4.6 getPrefFromLog.java 58
4.7 complete process 59
4.8 Statistic.java process 60

4.9 MahoutTaste.java process 62
4.10 Refined.java process 64
4.11 ad-hoc dataset 66

List of Tables

2.1 *X use case where the algorithm is applicable* 35
2.2 X: use case where the algorithm is applicable. X has been studied in this thesis 36

4.1 Execution time in milliseconds 65

Sommario

Al giorno d'oggi chiunque acceda a internet lo fa per ricercare delle informazioni. Il nostro lavoro propone l'applicazione di una serie di metodi di raccomandazione per aiutare l'utente nella ricerca di informazioni sul web attraverso la ricerca esplorativa multi-dominio. Con *ricerca esplorativa* si intende una ricerca che compone la risposta all'interrogazione attraversando passo dopo passo una serie di servizi che forniscono ognuno parte delle informazioni totali da fornire all'utente. Con *multi-dominio* si intende una ricerca effettuata interrogando più servizi distinti, anche tramite diversi protocolli di comunicazione, ognuno dei quali specializzato in una specifica area tematica (film, cinema, hotel, viaggi, ristoranti, ...). Siamo partiti da uno studio puramente teorico dei metodi di raccomandazione in caso monodominio oggi presenti in letteratura analizzando come e dove questi potessero essere usati in contensto multidominio. Abbiamo poi realizzato diverse soluzioni per implementare tali metodi nel nuovo contesto. Infine abbiamo testato i nostri algoritmi per verificare la validità ed efficacia dell'implementazione ma soprattutto delle idee di base da noi sviluppate.

Abstract

Today hundreds of millions of people use the Internet every day for searching and collecting information related to their cultural or practical needs. Our work aims to study recommendation techniques that can be applied to multi-domain exploratory search to help users during their searches. *Multi-domain* is a search done over more than one data source. When the user submits a query, it is "split" into several sub-queries and each part is sent to the appropriate data source or web service. All the results are collected and then the aggregate response is sent to the user. *Exploratory search* is a search where the user looks for some informations starting from an initial service. Then he choices the next service to query to get more information. The user repeats those steps until he satisfies his needs. We started from theoretical study of the recommendation methods in mono-domain case analyzing how and where they could be used in multi-domain context. Finally, we have tested our algorithms to verify the accuracy of the techniques we have developed during this thesis work.

Chapter 1

Introduction

> Una vita senza ricerca non è degna di essere vissuta.
>
> Socrate in Platone, Apologia di Socrate, 399/388 a.e.c.

1.1 Context

The aim of this thesis work is the study of recommendation techniques that can help the user when performing multi-domain search and exploratory tasks. For multi-domain search engine we mean a system which allows the search for more than one type of information simultaneously. In other words, a multi-domain system offers to the user the access to more services available at the same time and provides a result that is the aggregation of the results of the individual services. This result is not the aggregation of the raw data of each individual service, but it is the result of a selection over each service according to suitable criteria like reordering, clustering, addition or deletion of attributes, etc. With a fully functional multi-domain search engine we will be able to respond to queries formulated over different domains of interest with a good degree of accuracy of the result.

Multi-domain search is a new research field on which Politecnico di Milano is at the forefront of research. In fact, one of the main projects in the IT sector is the multi-domain search engine called SeCo. Exploratory search is a subject that became relevant since the birth of the web and search engines. The study of how a user moves through different information sources is crucial for facilitating and recommending the best search paths for users. The new frontier of the web is tied to multi-domain search. This means to switch from the exploration on single data services to more services at the same time. Multi-domain is a search done over more than one data source. The user submits a query, which is "split" into several sub-queries, and each part is sent to the appropriate data source or web

service. All the results are collected and then the aggregate response is sent to the user. Our work covers both recommendations for mono-domain and multi-domain exploratory search.

1.2 Problem addressed

Have you ever had to go to a new place and wanted to know what it offers? In recent years with the advent of the Internet you can answer many questions like "what hotels are there?" or "Which concerts (or cinemas, museums, ecc ...) are there?" Today, they are all unrelated information and the user is left with the task of aggregating those information to obtain a result. Multi-domain search engines have been designed to solve this problem. They allow users to consult multiple information sources and they return an aggregated result.

Mono-domain search engines like Tripadvisor, Imdb, etc..., provide suggestions to the user in addition to the result, or other information the user might be interested in. The reason is strongly linked to the business. E-commerce search engines try to sell more products recommending other item to user. The search engine choses advices for the user based on his preferences or the item the user has already bought.

The purpose of our discussion is to provide good suggestions on the best exploration directions to follow to the user who is using multi-domain search engines.

1.3 Proposed solution

Our thesis is motivated by the following multi-domain scenario:

> *An user is using a multi-domain search engine. He chooses an initial concept to search for, such as hotel, and requests a list of results. Among these results, the user chooses one of them and decides how to expand his search*[1]*. Once he has selected and queried the new service, the user chooses a result among those proposed and eventually repeats the process.*

We aim to adapt and use some existing recommendation algorithms currently used in mono-domain cases in this multi-domain scenario. Such recommendation should aim at responding to the question: given a selected result by the user, can we suggest what is the next service to be queried? And given the chosen service, is there a result that could be highlighted with respect to the others? The solution we propose is based on analyzing existing algorithms (Crowd searching,

[1]With the term *expand the search* we mean to perform a search of another concept related to the first.

Personalization, Search-based, Category-based, Collaborative filtering, Clustering, Association Rules) and studing the problem of applying these algorithms in the multi-domain case.

1.4 Summary of the work done

Our work covers the study of the techniques of recommendation in mono-domain case and their extension to the multi-domain case. To accomplish this, we have designed and implemented the necessary data structures and algorithms to cover the needs of recommendation techniques in multi-domain environments. During the analysis of the recommendation techniques used today in mono-domain search engines we have noticed how these can not be applied directly to the multi-domain case. Mono-domain search do not cover important aspects such as the order of access to services, the relationships between services, etc.

We have tested three different algorithms. Two are adaptations of techniques used in mono-domain case, while the third has been realized ad hoc for the multi-domain case. For each of these we have highlighted strengths and weaknesses in order to obtain an evaluation of the recommendation techniques in a multi-domain search engine.

Chapter 2

Background & Related Work

> Part of the inhumanity of the computer is that, once it is competently programmed and working smoothly, it is completely honest.
>
> Isaac Asimov, The Winds of Change and Other Stories, 1983

This chapter will show what are the main recommendation algorithms and how they are used today. For each of them we provide examples of existing applications. We will focus on the most important techniques and show which ones have been adopted in this thesis.

2.1 Related work

Nowadays many application examples of recommendation exist in the mono-domain case. Since the creation of search engines, one of the main challenges has always been trying to refine as best as possible the search of data in order to obtain the best results.

Every time a user enters an e-commerce website, he is bombarded with advice on what to buy, especially if he is a registered user on that site. This is a classic example of recommendation. Similarly, the advertising often show on the side of social networking sites, such as Facebook for example, are related to the search and navigation done by the user. We will show some real-world examples later when we'll discuss in detail all the algorithms of recommendation present today in literature and studied in the thesis. But all this is true in the mono-domain case where one is interested in a search on individual service. There are many articles, from academic sources and not, on research and development of these algorithms in mono-domain case.

multi-domain search changes the way we have to compute the suggestion. It's a new field of research and nowaday there isn't a functional application. We need more study and work to get targeted suggestion to the user. The aggregation of information from multiple services is not easy even if the studies are moving in the right direction. Increasingly, online you can see examples of pseudo-search multi-domain. We take expedia.com as an example of multi-domain search.
When a user accesses on expedia.com and tries to perform a trip he has the impression of being able to make a pure search on a multi-domain search engine where over the trip result the user can choose hotels, rental cars. We see the initial interface:

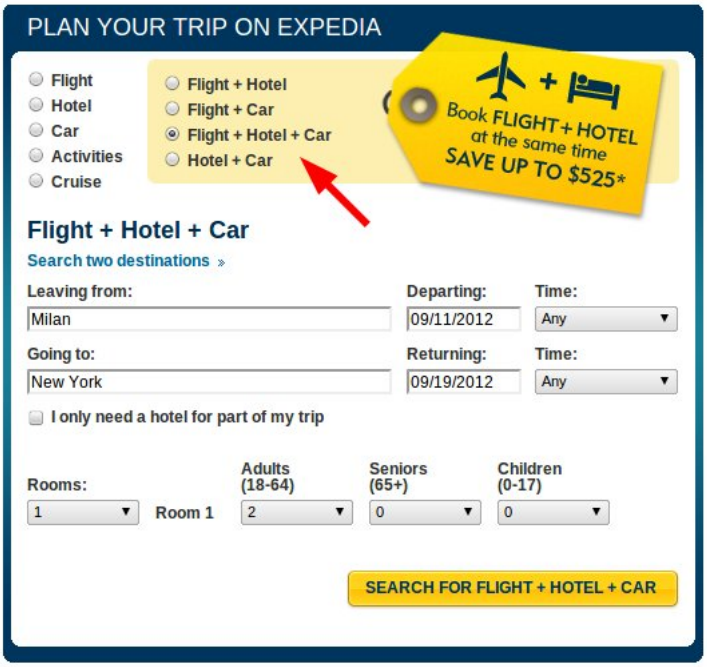


Figure 2.1: *expedia.com home page*

The red arrow in figure 2.1 indicates the area where you can choose among several services. searching a trip to milan newyork, we obtain:

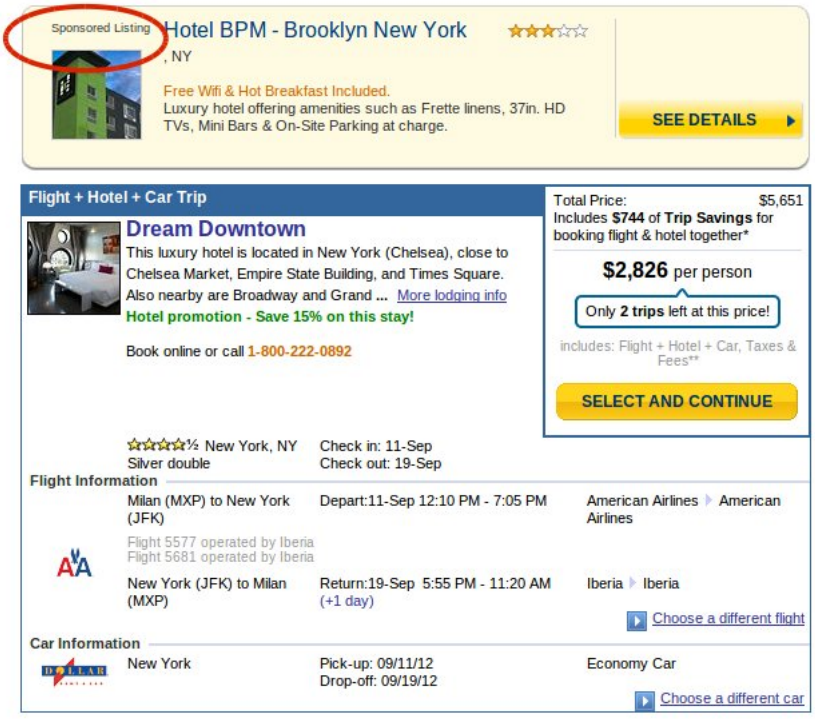


Figure 2.2: *expedia.com search of a travel + hotel + car*

The obtained results are significant but they are not addressed to the user preferences. The first results are those sponsored, then the rank is calculated based on geolocation. Expedia results has not real correlation between the various elements such as fly, car, hotel. For example, an user who chose a five star hotel can be addressed to the same car rent which can be chosen by an user who reserves a bed and breakfast. Moreover it bases every next choice on geolocation. There are no other available relation between each service. Today we want to find a multi-domain search engines with greater flexibility in the selection criteria of the aggregation of results in order to cover all possible types of queries.

Today the web, as shown in figure 2.2, hasn't a very high level multi-domain search engine but it's moving in the right direction. A project that is fully focused on the multi-domain search and are having very positive feedback is SeCo that is shown in the following section.

2.2 SeCo

The project Search Computing (Se-Co) [2] [1] is used to perform complex searches, starting from an initial concept and allowing the user to refine, expand and broaden your search by adding new concepts shown earlier on the results.

This is possible by using the concept of liquid query, complex queries that may change during the search session where the results fill the fields selected by the user as water poured into a container.

The emerging paradigm of software services has so far been neutral to search. Search Computing is an evolution of service computing focused on building the answers of complex queries by interacting with a constellation of cooperating search services, using ranking as the dominant factor for service composition. New language and description paradigms are required for interconnecting services and for expressing queries. Semantic domain knowledge helps enriching terminological knowledge about objects being searched. New protocols help capturing ranking preferences and their refinement; new interfaces present complex results with simple visual descriptions. Ranking is relative to individuals and context and therefore reflects personal and social contributions. Economical and legal implications of Search Computing must be understood and mastered. In summary, Search Computing is a multi-disciplinary effort which requires adding to sound software principles contributions from other sciences such as knowledge representation, human-computer interfaces, psychology, sociology, economical and legal sciences.

The user during the search session from a template in which the initial part of the first search keys on the first concept (topic) that normally refers to a specific search service. Once you have the results, the user can perform a number of operations that can be collected into two categories: remote interactions and local interactions.

Remote queries that require the server to perform some operation to produce new results: the expansion to a new service, get more results from a single service or all services together. Queries reorganize local results without making requests to the server: clasterizzazione results, reorganize, roll-up and drill-down.

2.2.1 Liquid Queries and Liquid Results in Search Computing

Liquid queries [1] are a flexible tool for information seeking, based on the progressive exploration of the search space; they produce "fluid" results which dynamically adapt to the shape of the query, as a liquid adapts to its container. The liquid query paradigm relies on the SeCo service mart and multi-domain query execution concepts: an expert user selects a priori the service marts relevant to the information seeking task at hand and the connections necessary to join them, and publishes such a definition in the SeCo back-end. The Liquid Query client-side interface consumes the application definition created by the expert and dynamically builds a query interface for the end-user. Such interface allows one to supply keywords to query the pre-configured service marts and offers controls for exploring the combinations computed by the SeCo execution engine. The interaction

commands are based on a tabular representation of results and comprise: reordering, clustering, addition or deletion of attributes, addition of extra service marts to the query for specific items in the result set or for the entire result set, request of more results from all services or from selected ones, expansion of details on selected items, and more. The Liquid Query is equipped with multiple data visualization options suited to render multi-domain results and can be instrumented with indicators showing the quality of the result set.

2.3 Recommender systems

This section explains some of the main recommendation algorithms [5] used on the Internet everyday.

2.3.1 Personalization

Personalization concerns adapting to the individual needs, interests, and preferences of each user so for example is defined as filling out a profile that aims to obtain a detailed description as possible of the tastes of the user.

The compilation of the profile can be made directly by the user or can be automated obtaining information already present on other sites or communities. amazon For complex queries the personalization is useful for example to identify which is the main concept on the results presentation. When choosing a trip you can ask what are the main filters: weather, traffic, points of interest, ... Obtained the result, you can make a ranking based on this information.

2.3.2 Search-Based

The Search-Based is one of the basic methods of recommendation. The visitor types a search query and the system retrieves all the items that correspond to that query.

The system recommends some of the records in the result set like better than the other ones, it's a non-personalized ranking (sales rank, popularity,genre, etc...).

For example the user types a search query for a book or kind of a books. The system returns all the items that correspond to that query, e.g. 6 books. The recommendation is based on general, non-personalized ranking filters for books like sales rank, popularity, authors, ...

Now let's see an example of Search-Based.

Consider a user who accesses the site amazon.com for a book, the user does not know a priori which book to choose. So he thinks to look for a best-seller. What happens?

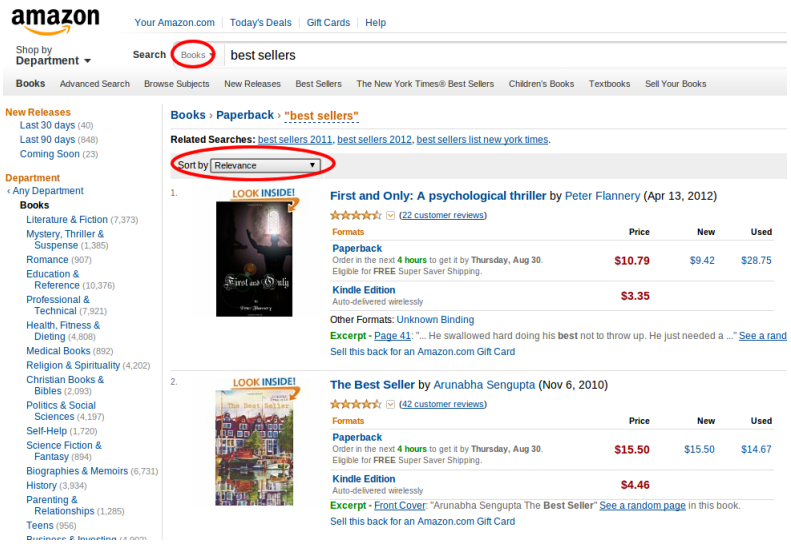


Figure 2.3: *search-based relevance on amazon.com*

The results in this first case are ranked by relevance. The recommendation is then obtained based on the importance of the object. This recommendation, however, can be changed, as can be seen from the image. In fact, there is a combo box for the selection of the discriminant for search. What happens then if I change my method of search?

If the user chooses an ordering of results we compared the popularity of this set of values:

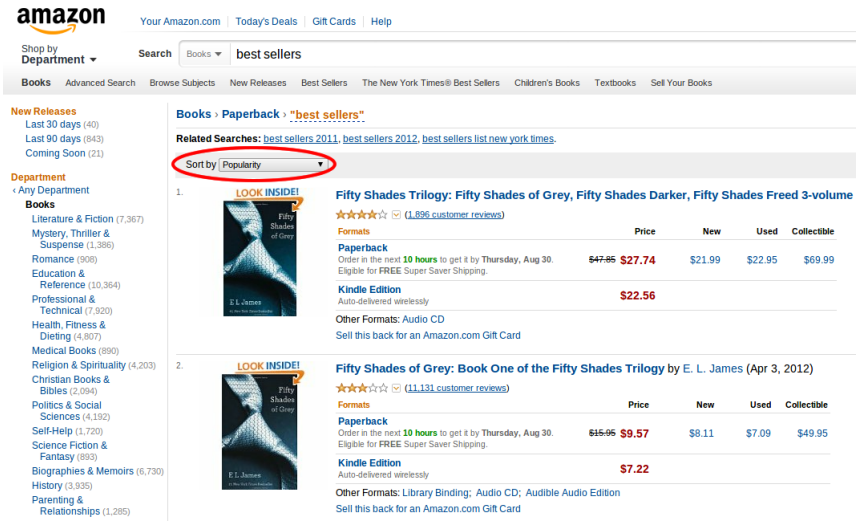


Figure 2.4: *search-based by popularity on amazon.com*

As you can see the results change. Passing by relevance to popularity we get a set of data, in this case books, different.

This type of recommendation is purely tied to the object and has no knowledge of the tastes of the user but still offers the chance to get some advice on what is the best best sellers depending on sales or popularity or other factors such as price.

Pros:

1. Simple to implement

Cons:

1. Not very powerful
2. Which criteria to use to rank recommendations?
3. Is it really recommendations ?
4. The user only gets what he asked for

2.3.3 Category-Based

The category-based is based on the concept that each item belongs to one category or more. The user makes an explicit or implicit choice selecting a category of interest (refine search). The system selects categories of interest on the behalf of the customer, based on the current item viewed, past purchases, etc. Certain items(best-sellers,new items) are eventually recommended.

Now let's see an example of Category-Based. Consider a user who accesses the site amazon.com for a book, the user does not know a priori which book to choose. So he thinks to look for a best-seller. What happens?

As a first step the user types best-seller in the search bar of amazon.com.

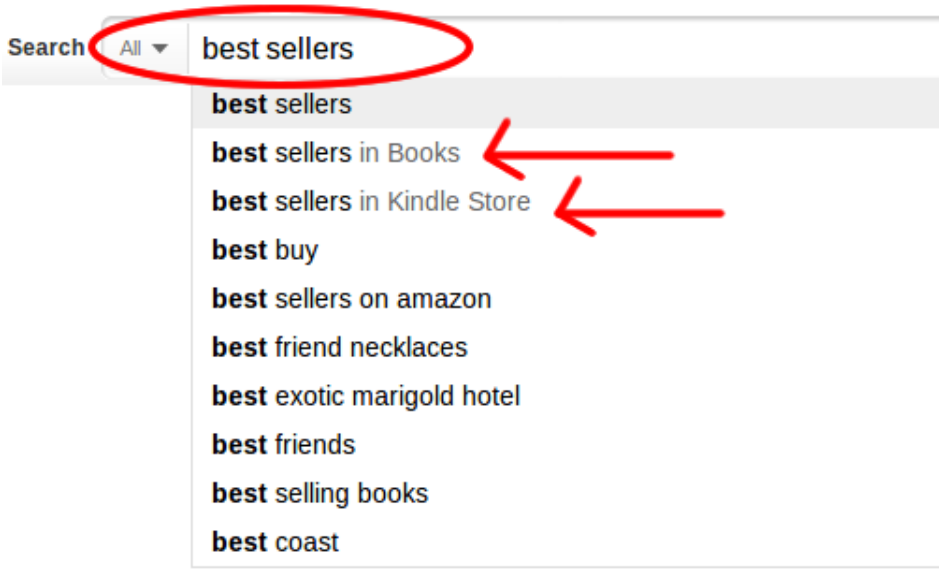


Figure 2.5: *category based on amazon.com*

It can be seen immediately as the system shows, in addition to the query typed by the user, other possible ways to enter / refine the query. In the picture you can see, marked with an arrow, is recommended as a specific category of bestsellers, those in Books (paper for example) or Kindle. This is an example of how the items are categorized.

Assuming that the user chooses bestsellers in books, the system categorizes the choice in the bar showing that the search is on books, rather than all as shown in the follow figure.

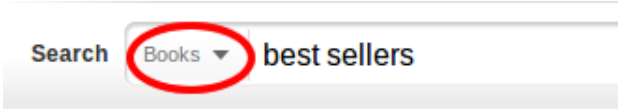


Figure 2.6: *category based - categorization*

The amazon website offers a further categorization of the previous visible. Once you have chosen bestsellers in the category Books, Amazon still offers the possibility to choose between different types of product categories such as:

Figure 2.7: *category based second-level of categorization*

Pros:

1. Still simple to implement

Cons:

1. Not very powerful, which criteria to use to order recommendations? is it really recommendations ?
2. Capacity highly depends upon the kind of categories implemented :
   - Too specific: not efficient
   - Not specific enough: no relevant recommendations

2.3.4 Collaborative-Filtering

Collaborative filtering [4] techniques compare customers, based on their previous purchases, to make recommendations to similar customers. It's also called social filtering. The collaborative filtering techniques find customers who are similar

(nearest neighbors) in term of tastes, preferences, past behaviors, then aggregate weighted preferences of these neighbors and make recommendations based on these aggregated, weighted preferences (most preferred, unbought items).

Imagine a system with multiple users. User C requires a recommendation on the next book to buy.

	Book 1	Book 2	Book 3	Book 4	Book 5	Book 6
Customer A	X			X		
Customer B		X	X		X	
Customer C		X	X			
Customer D		X				X
Customer E	X				X	

Figure 2.8: *example of collaborative filtering*

Customer B is very close to C (he has bought all the books C has bought). Book 5 is highly recommended Customer D is somewhat close. Book 6 is recommended to a lower extent Customers A and E are not similar at all. Weight=0.

Pros:

1. Extremely powerful and efficient

2. Very relevant recommendations

Cons:

1. Difficult to implement, resource and time-consuming

2. What about a new item that has never been purchased?Cannot be recommended

3. What about a new customer who has never bought anything? Cannot be compared to other customers so no items can be recommended

2.3.5 Clustering

Clustering is similar to collaborative filtering because is another way to make recommendations based on past purchases of other customers clustering customers into categories. Each cluster will be assigned typical preferences, based on preferences of customers who belong to the cluster. Customers within each cluster will receive recommendations computed at the cluster level

Now let's see an example of clustering on 5 users.

	Book 1	Book 2	Book 3	Book 4	Book 5	Book 6
Customer A	X			X		
Customer B		X	X		X	
Customer C		X	X			
Customer D		X				X
Customer E	X				X	

Figure 2.9: *example of clustering*

Customers B, C and D are clustered together. Customers A and E are clustered into another separate group. Typical preferences for cluster are:

1. Book 2, very high

2. Book 3, high

3. Books 5 and 6, may be recommended

4. Books 1 and 4, not recommended at all

Now let's add a user assuming has already bought two books, What's happen now?

	Book 1	Book 2	Book 3	Book 4	Book 5	Book 6
Customer A	X			X		
Customer B		X	X		X	
Customer C		X	X			
Customer D		X				X
Customer E	X				X	
Customer F			X		X	

Figure 2.10: *example of clustering*

Any customer that shall be classified as a member of CLUSTER will receive recommendations based on preferences of the group:

1. Book 2 will be highly recommended to Customer F

2. Book 6 will also be recommended to some extent

A significant problem is what happens to the clusters when you the number of the users. Most users increases more likely that a user belongs to multiple clusters (clusters overlap).

	Book 1	Book 2	Book 3	Book 4	Book 5	Book 6
Customer A	X			X		
Customer B		X	X		X	
Customer C		X	X			
Customer D		X				X
Customer E	X				X	
Customer F			X		X	

	Book 1	Book 2	Book 3	Book 4	Book 5	Book 6
Customer A	X			X		
Customer B		X	X		X	
Customer C		X	X			
Customer D		X				X
Customer E	X				X	
Customer F			X		X	

Figure 2.11: *example of clustering*

Predictions are then averaged across the clusters, weighted by participation
Pros:

1. Clustering techniques work on aggregated data: faster
2. It can also be applied as a first step for shrinking the selection of relevant neighbors in a collaborative filtering algorithm

Cons:

1. Recommendations (per cluster) are less relevant than collaborative filtering (per individual)

2.3.6 Association Rule

Association rules are created by analyzing data for frequent if/then patterns and using the criteria support and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true.

In data mining, association rules are useful for analyzing and predicting customer behavior. They play an important part in shopping basket data analysis, product clustering, catalog design and store layout.

Programmers use association rules to build programs capable of machine learning. Machine learning is a type of artificial intelligence (AI) that seeks to build programs with the ability to become more efficient without being explicitly programmed.

One of the original motivations of the latter approach is to help supermar-ket managers to analyze past transaction data and to improve their future business decisions including catalog design, store layout design, coupon design and so on.

Given a large database of customer transactions with each transaction storing items purchased by a customer during a visit, association rule mining aims to discover all significant associations between items in the database.

Association rules work at the item level.

Now let's see an example of the use of the techniques of association-rule in the context of recommendation.

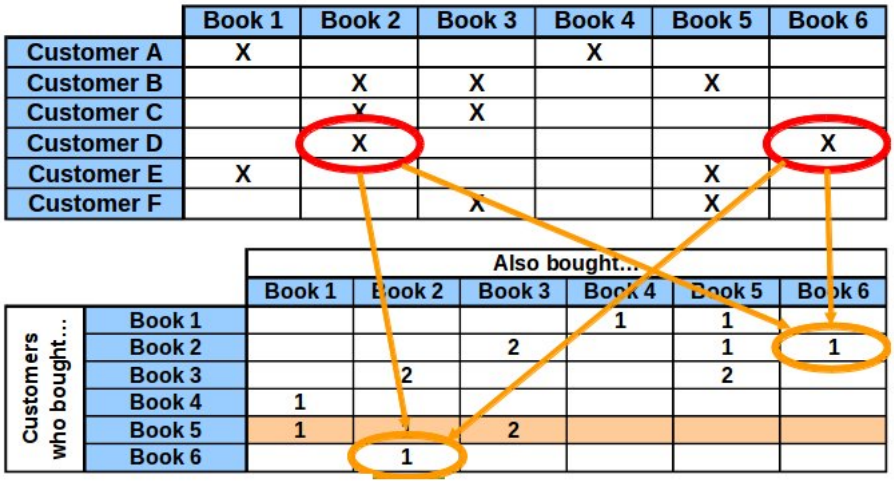


Figure 2.12: *example of association rule*

These association rules are sed to make recommendations.
If a customer has some interest in Book 5, he will be recommended to buy Book 3 as well.
Recommendations are constrained to some minimum levels of confidence, What if recommendations can be made using more than one piece of information? Recommendations are aggregated.
Pros:

1. Fast to implement
2. Fast to execute
3. Not much storage space required
4. Not individual specific
5. Very successful in broad applications for large populations, such as shelf layout in retail stores

Cons:

1. Not suitable if knowledge of preferences change rapidly

2. It is tempting to do not apply restrictive confidence rules so May lead to litteraly stupid recommendations.

2.3.7 Information Filtering

Information filtering is a suggestion technique that compares objects based on their contents or characteristics. It can exploit syntactical information on objects (based on features like word frequency, ...) but also semantic knowledge of objects (using ontologies).

There are two major approaches for information filtering: content-based filtering and collaborative filtering. The content-based filtering system selects items based on the correlation between the content of the items and the user's preferences, while a collaborative filtering system chooses items based on the correlation between people with similar preferences.

The content-based filtering require the definition of the content of an item. It can be explicit attributes or characteristics of the item. For example for a film can be author, main character, production year, ... The content can also be textual content, like title, description, table of content, etc.

Several techniques exist to compute the distance between two textual documents. The term frequency - inverse document frequency (TF-IDF) is a numerical statistic which represents how important a word is to a document. This weight is computed as

$$TFIDF = \frac{TF \times IDF}{\sum_N (TF \times IDF)^2} \tag{2.1}$$

where

$$TF = log(count + 1) \tag{2.2}$$

is the *Term Frequency*, the number of occurrences of the term in a document, and the

$$IDF = log(\frac{docs + 1}{docs - the - term - occurs - in}) \tag{2.3}$$

is the *inverse document frequency*, a measure of whether the term is common or rare across all documents

	COUNT							
	building data mining applications for crm	Accelerating Customer Relationships: Using CRM and Relationship Technologies	Mastering Data Mining: The Art and Science of Customer Relationship Management	Data Mining Your Website	Introduction to marketing	consumer behavior	marketing research, a handbook	customer knowledge management
a							1	
accelerating		1						
and		1	1					
application	1							
art			1					
behavior						1		
building	1							
consumer						1		
crm	1	1						
customer		1	1					1
data	1		1	1				
for	1							
handbook							1	
introduction					1			
knowledge								1
management			1					1
marketing					1		1	
mastering			1					
mining	1		1	1				
of			1					
relationship		2	1					
research							1	
science			1					
technology		1						
the			1					
to					1			
using		1						
website				1				
your				1				

Figure 2.13: *Word frequency calculated on 7 book's titles*

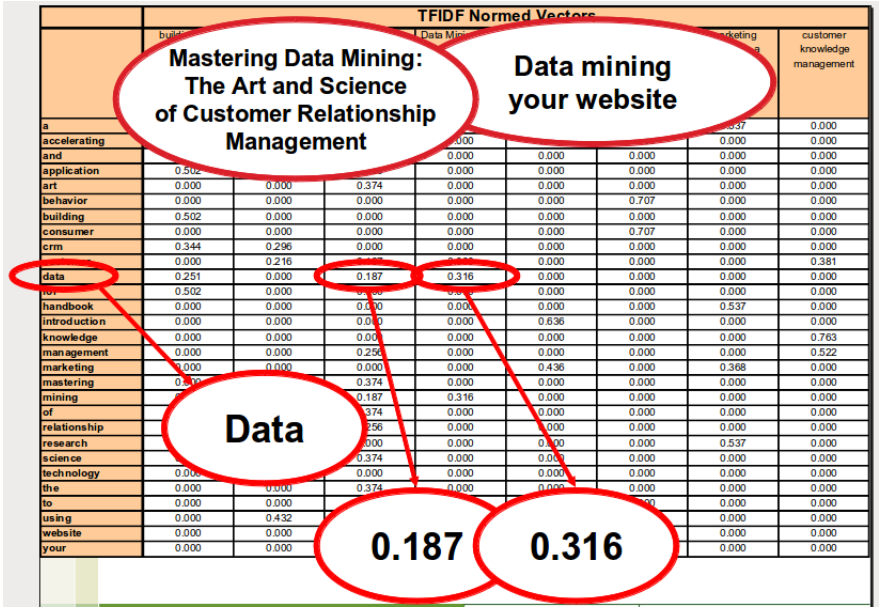


Figure 2.14: *TFIDF calculated on 7 book's titles*

Pros:

1. No need for past purchase history
2. Not extremely difficult to implement

Cons:

Recommendation Algorithms	Questions			
	Services	Instances	Attributes	Displaying
Crowd	X	X	-	-
Personalization	X	X	X	X
Search-based	X	X	-	X
Category-based	-	-	-	-
Collaborative filtering	X	X	-	X
Clustering	X	(X)	-	X
Association rule	X	-	X	X
Information filtering	X	-	-	X

Table 2.1: *X use case where the algorithm is applicable*

1. Static recommendations

2. Not efficient is content is not very informative e.g. information filtering is more suited to recommend technical books than novels or movies

2.3.8 Crowd

The technique of Crowd recommendation is based on shared help between users. A user who needs help asks a question to the community that tries to answer it. Obtained the answers the user chooses what he considers the best one. Examples of such applications are Yahoo-Answer or ToogleChat. The former, the users ask questions and wait for the response of other users connected to the community. To encourage users to answer the system offers bonus that allow users to use Yahoo features. The second one, users are connected to a chat room available into their browsers. They can chat while are searching information in the web so they can ask a question about their search and obtain answer about that. A similar thing is done by the Italian search engine Volunia.

2.3.9 Overview

In section 2.3 we have explained the algorithms one by one with their pros and cons. Now we show in table 2.1 where these algorithms can be used.

A previous thesis made by students of Politecnico di Milano in Como has focused on how to use these suggestion tecniques to chose the best way to display tuples.

We focus mainly on services and instances. Let's see now, with a new table, the case studies discussed in the thesis.

In section 2.3.9 we show how search-based and collaborative filtering can be used to suggest services an instances to the user. Let us now see the scenarios on which algorithms are studied in the thesis.

Recommendation Algorithms	Questions			
	Abstract concept (hotel,...)	Services (Expedia,...)	Instances (tuples to display)	Instances (filters to apply)
Crowd	X	X	X	-
Personalization	X	X	X	-
Search-based	**X**	**X**	X	-
Category-based	-	-	-	-
Collaborative filtering	**X**	**X**	X	X
Clustering	X	X	X	X
Association rule	X	X	-	-
Information filtering	X	-	-	-

Table 2.2: *X: use case where the algorithm is applicable. X has been studied in this thesis*

Search-Based

This technique suggests what were the results most seen based on global statistics. For example, if you search a restaurant in your city, a Search-Based system returns the restaurant with more feedback. It's a technique of ranking.
On the technical concepts the Search-Based technique is not of great importance while on services it is useful because reccomends the most used service.
For instances, seen as tuples to display, the Search-Based has good results because the underlying services have a limited degree of change. For filters, the issue is more subtle. If you search for a restaurant, for example, the system suggests which already filters are applied.

We have focused on Search-Based for physical services: This recommendation methods NOT binds the current user and its profile with the suggestion proposed. Recommended services depend on the overall statistics.

We need a history of searches previously carried out by using the system, so we must keep track of which physical services have been selected by users for each query. This information can be obtained in different ways according to the architecture of the system.

Collaborative-Filtering

This technique suggests similar elements based on previous search of users who have done similar search.
On abstract concepts, this technique is applicable because it encourages user choices using the choices done by similar users (users who have similar search).

For services is not recommended because using a Search Based i would expect better results, but on the other side can bring an advantage if used with more precise suggestion algorithms. Instances, seen as linked tuples, could be used in the Collaborative Filtering like: the user performs a search like a restaurant in a city, and the system tells him what is the most visited restaurant by the largest number from his friends.
For the filters to be applied, Collaborative Filtering is useful because by previous search is possible to know what filters were chosen by the users and then these can be assumed correct for the user.

Finally for attributes is more convenient using other techniques.

We have focused on Collaborative filtering on concept. The system according to the query and the current concept, search who else has made similar queries and it provides suggestions as other concepts. To obtain the reccommendation we need to save:

1. users on the system
2. domains explored

It's also considered very very important in the timeline where the exploration is carried out. We get, in this way, a timeline of the search. The final result is the next service to explore.

2.4 Apache Mahout

Apache Mahout [3] is a library for building scalable machine learning applications. At the time of writing Mahout support four use case:

- Recommendation mining;
- Clustering;
- Classification;
- Frequent itemset mining.

Clustering takes e.g. text documents and groups them into groups of topically related documents.

Classification learns from exisiting categorized documents what documents of a specific category look like and is able to assign unlabelled documents to the (hopefully) correct category.

Frequent itemset mining takes a set of item groups (terms in a query session, shopping cart content) and identifies, which individual items usually appear together.

Recommendation mining takes users' behavior and from that tries to find items users might like. Apache Mahout uses the collaborative filter approach and the sub-framework named Tast implement this tecnique.

2.4.1 Mahout - Taste

Taste is a sub-framework of Apache Mahout since 2008. The system architecture is shown in figure 2.15

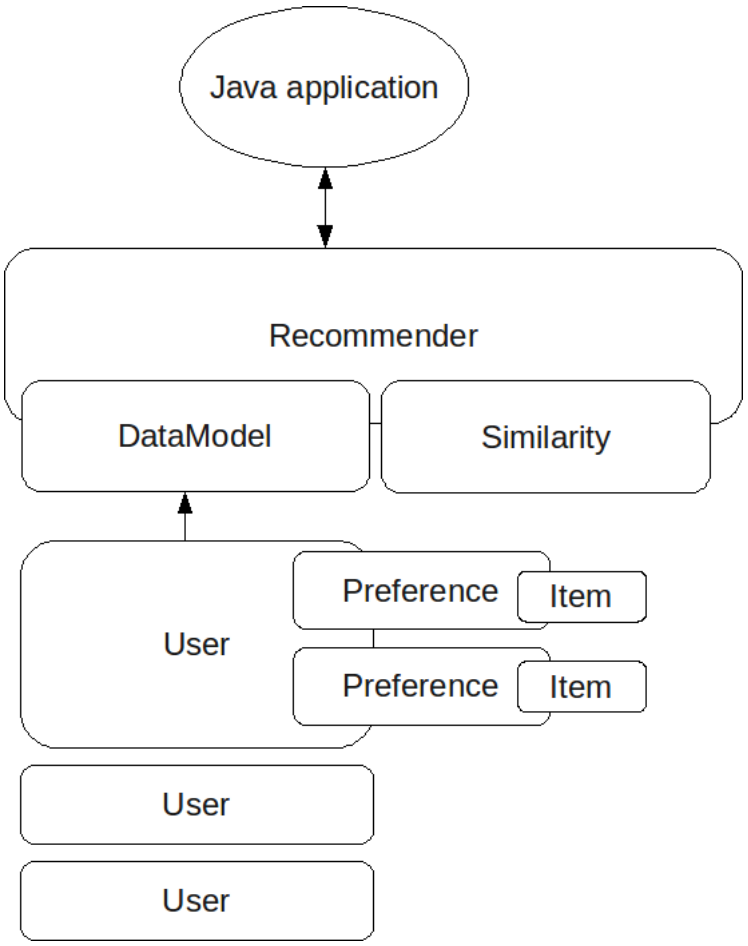


Figure 2.15: *Mahout-Taste architecture*

The main building block in Taste is the Recommender. The Recommender recommends items based on a given item or it determines users with similar tastes. To determine the items (or the users) similar to the chosen one the recommender applies a similarity function on a subset of pairs of items (or users) in the dataset. A similarity function usually returns a value between 0 and 1, with 1 representing two completely similar items and 0 completely dissimilar items. When the similarity function processes pairs in the dataset the resulting similarity values are

collected and are either kept in memory or stored on the filesystem or a database. When the Java application requests a few recommendations for a given item, the Recommender returns the items with the highest similarity.

The Recommender retrieves items and users through the DataModel abstraction. Taste contains DataModel implementations for retrieving your dataset through the filesystem or a database. In addition, the DataModel provides methods that count the total number of users, total number of items, number of users that prefer a certain item, and many more functions. Similarity functions use these numbers to compute a similarity value for pairs of items or users. You can build a recommender with Taste by adding a DataModel and a similarity function to a Recommender. You can also define your own similarity function by extending UserSimilarity or ItemSimilarity to recommend users or items, respectively.

Chapter 3

Recommendation algorithms for exploratory search

> The real danger is not that computers will begin to think like men, but that men will begin to think like computers.
>
> Sydney Justin Harris

This chapter will present all the components and processes necessary to obtain recommendations in multi-domain environment.

3.1 Defining the problem

In recent years any information, doubts, questions of everyday life, they can be placed as a query to a service on the internet. The result obtained, however, often does not reflect what you want to obtain. For example the search returns too many useless information cause lack of refining the query, misunderstandings due to natural language, etc. . . .

Over the years, the various search engines have improved search techniques to provide results as close as possible to the desired one.
But what's happen if a user tries to perform a search of information across multiple services?, ie what happens when the query concerns more services? How can you be sure that the given result is correct?. For example, what is the most comfortable hotel in London? Where is the most famous restaurant near the San Francisco airport? Which museums is recommended to see in Paris? The thesis presents algorithms, ideas and suggestions on how to tackle problems such as previous queries.

It's now necessary to focus on some important definitions to understand how

we implemented the algorithm.

3.1.1 Overview

The user wants to get an aggregate of information (Hotel + Museum + ...); To build this aggregate we use an iterative approach.

The user, starting from a service, step by step, it collects information increasing the data obtained passing from service to the next one. Our goal is to suggest to the user what other source of information use and highlight some records that are the more interesting than other ones. To better understand this process we need to understand what does it mean that the user moves on services. Moreover what it means to get information(instances)?. We begin to answer the former question. Imagine a system with 7 services related according to a certain logic.

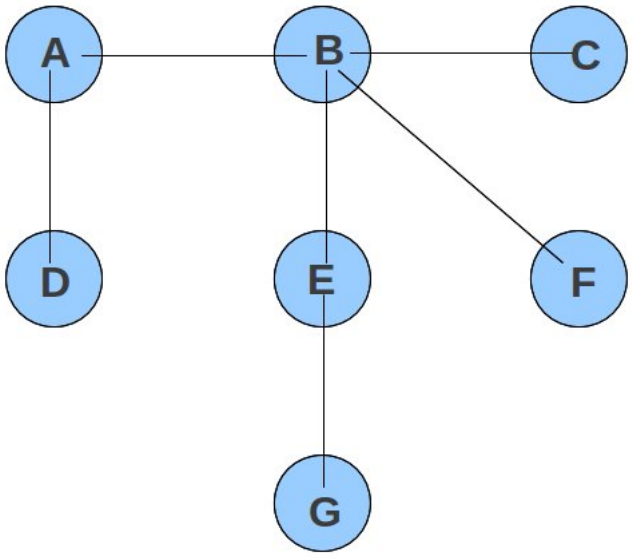


Figure 3.1: *space of services*

For example, the service E represents the cinemas, while the service G movies. This is a services space.

A user can access any of these services offered by the system and then, by querying the service, he obtains information to the services directly related. In this way the user creates paths on the space of services. These generated paths will become the future log stored in the database and they are used for the calculation of the recommendations.
Suppose a user U1 accesses the service A, then from there it moves to B and then C.

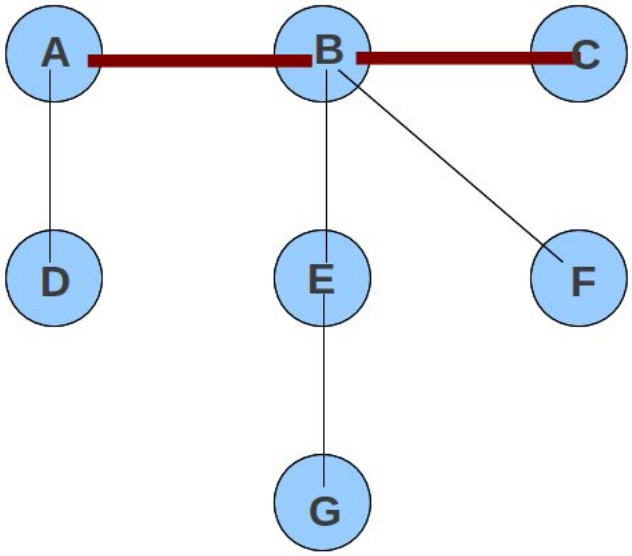


Figure 3.2: *path on the space of services*

From the process show in 3.2 we can obtain the following information:

1. to characterize the user with an identifier;

2. to define the session;

3. to save the search path, ie services visited, obtaining the log.

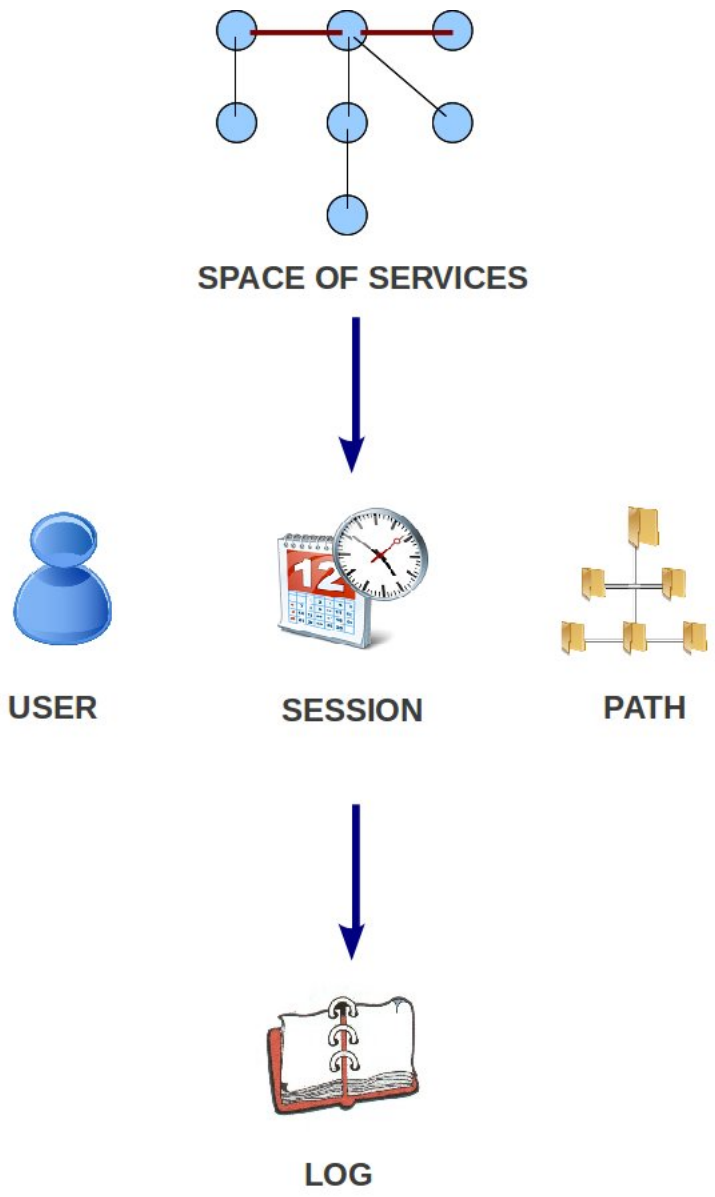


Figure 3.3: *data extraction step-1*

Once the user navigation logs are produced, they are saved in a database and queried using SQL queries to obtain recommendations.

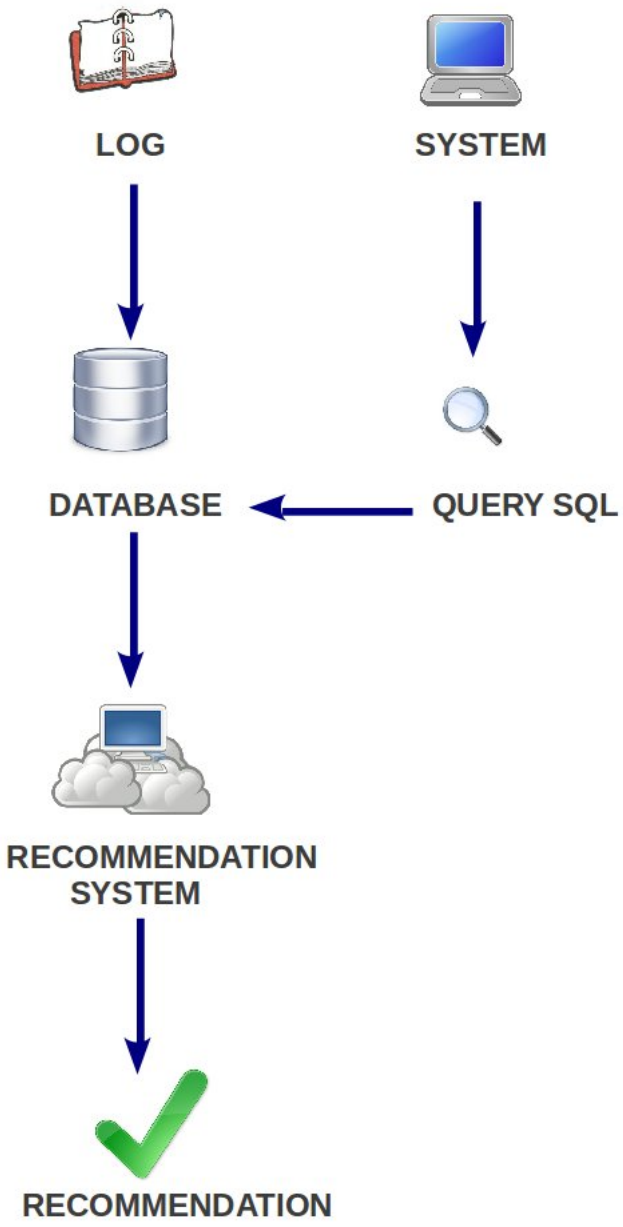


Figure 3.4: *data extraction step-2*

All the previously described process (described in figure 3.4) is implemented in the thesis.

Instances and log

When a user focuses on a service, he implicitly requires to show the instances available on the service that respond to the query.
For example, a user wants to look for hotels in a particular place and obtains some instances. These instances are part of the search conducted by the user and then these are saved in the log.
They are of fundamental importance because they contain the meaning of a move between a service and another. In the case of suggestion for instances, after seeing similar path on the services is necessary to compare instances and thus they must be saved.

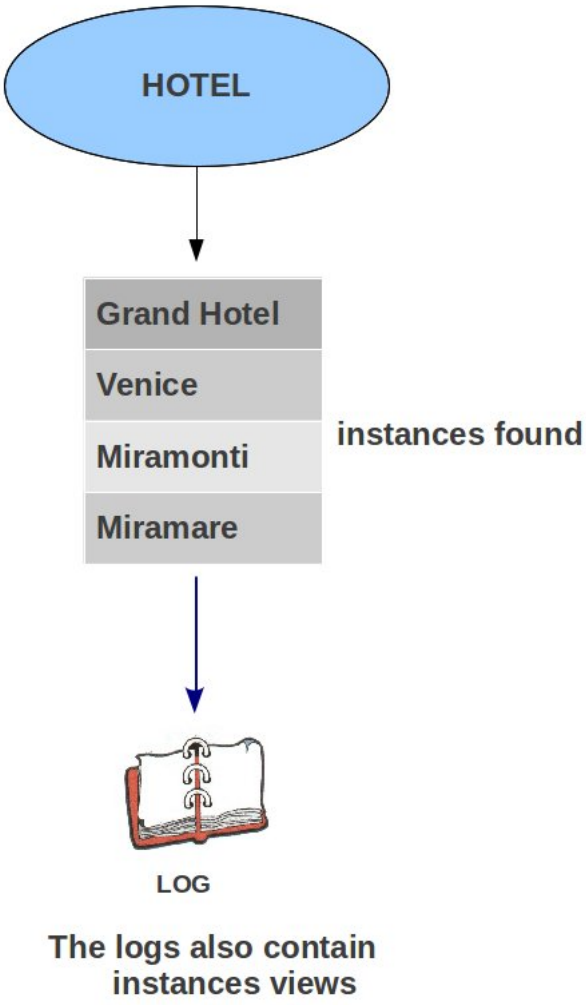


Figure 3.5: *save instances from service*

3.2 Recommendation algorithms

This section describes the various recommendation algorithms designed in the thesis. For each algorithm will expose the basic idea, positive and negative aspects and usage examples. All these algorithms have been written in Java for greater portability to any system can be used.

3.2.1 Purely statistic approach

The first method is purely based on a statistical computing of the services seen by users. During a session, a user can visit a single service or multiple services. At each step of the exploration the recommender system can advise what is the next best service looking at all the paths made by previous users.

In this case the algorithm has a database with all the users logs.

When a user visits a service, the recommendation is calculated by looking what is the next best service by counting the occurrences starting from the current service seen by the user. In other words, if a user has viewed a service A, the algorithm recommends the next services calculating first how many other users have seen the current user service and from then it counts the occurrences of the next services creating the rating. The result will be sorted in descending order to obtain the recommended service (highest number of occurrences)first.

Pros:

1. easy to implement;

Cons:

1. Does not care the user's profile;
2. Purely statistical;

3.2.2 Machine-learnig based approach, implemented with Mahout-Taste

The second method was implemented using Mahout libraries (see chapter 2.4.1). This algorithm, thanks to the use of libraries provided by mahout-taste, is a refinement of the previous. In fact, it takes the similarity between users. In the previous case it had a recommendation given the current path without considering of who is the user and whether the user has done other search previously, now, instead, the recommendation is calculated given the user.

The basic idea is to find similar users and calculate on these the recommendation for the current user.
A problem of the libraries provided by Apache Mahout-Taste was the absence

of the library for the connection to the database. This problem were solved by creating the class DbDataModel.java.

Pros:

1. recommendation based on user profile;
2. wide variety of clustering methods offered by the library

Cons:

1. doesn't care the temporal sequence of the searches;

3.2.3 Refined approach, ad-hoc for multi-domain exploratory search

The third algorithm implemented represents a further evolution of the previous ones. It includes new features such as:

1. the temporal sequence of the visited services;
2. the choice of different methods of recommendation related to the similarity of the search paths.

The basic idea is to compare the paths and find similar sequences. To do so, we first represent all the services and their connection as a graph.

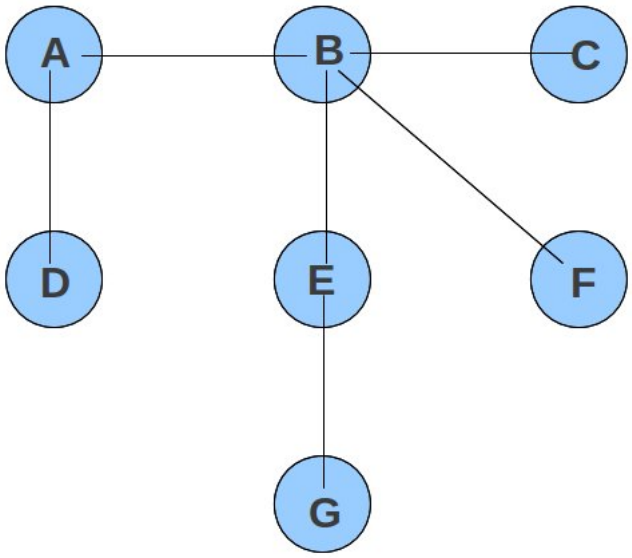


Figure 3.6: *This graps is an example scenario with seven services*

The graph in figure 3.6 shows a scenario with seven services. Each node represents a service labeled by a character while an arch represent the available connection between two services. The user can move from a service to another only if they are connected by an arc.

Service	Letter
Hotel	A
Restaurant	B
Museum	C
Theater	D

Figure 3.7: *lookup-table*

As we label each service with a letter of the alphabet, a string identify a path among the services. For example we can take a lookup table in figure 3.7. If an user explore hotel, restaurant and museum in this order, the corrisponding string will be "abc".

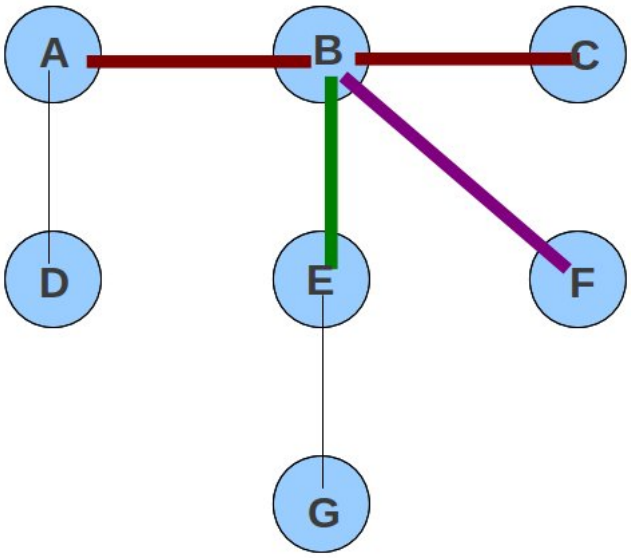


Figure 3.8: *example of path*

The graph in figure 3.8 shows a three steps exploration made by three different users. These paths are highlited in the above graph and can be represented by the sequence of the node labels. So, in this case the users paths are converted in the following strings: "abc", "abe", "abf". The strings obtained can be compared

in many ways to define the degree of similarity between them. The recommendation will be obtained by using strings comparison algorithms implemented in this thesis. In other words the algorithm passes from a study of similarity between paths to a study of similarity between strings. The string processing algorithms implemented in this case study are:

armonic returns a score greater for paths with final services similar to the current one.

exponential use an exponential calculation giving decreasing weight from newest to oldest. It admits paths of different length.

Levenshtein calculate the Levenshtein distance. It admits paths of different length. The calculation of the similarity is the inverse of the distance of levenshtein added to one.

Chapter 4

Implementation and experiments

> Program testing can be used to show the presence of bugs, but never to show their absence!
>
> Edsger Dijkstra, Structured Programming, 1972

This chapter will show the implementation of the various processes related to the studied recommendation techniques. In addition we will show the experiments we performed for validating the quality and effectiveness of the different techniques.

4.1 Algorithms implementation

All the algorithms implemented in the thesis has been written in Java for maximum portability on different systems.

As first step we studied how should be the structure of a log to understand how to store it in the database and then use it for our purposes. This structure is based on the described one in the previous chapter in the overview section, but we now want to investigate some things about them.

4.1.1 Logs of exploratory search activities

Logs of exploratory search activities input data for the designed algorithms in this thesis.

The logs come from paths made by users on the state space. During these search paths, users move over the services looking for the request instances. This process is a fully important kind of information that must be saved. In figure 3.4 we can see how you can save the information content of a log in a database.

For "Step of user on the state space", we mean the movement of a user over an edge between two services. There is a particular case when the user looks for only one service or the user is on the last service viewed on his search path. Take for example a user U1 who step between two services:

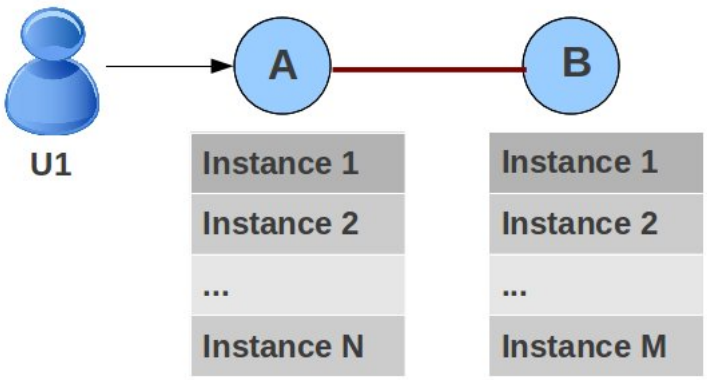


Figure 4.1: *path over two services*

We can obtain the following information:

1. user identification;
2. initial service
3. next service
4. instance seen in initial service
5. instance seen in next service
6. timestamp that identifies the search step in the current user session

Assuming instead a single step, ie a visit to one service or the last step of exploration, we have:

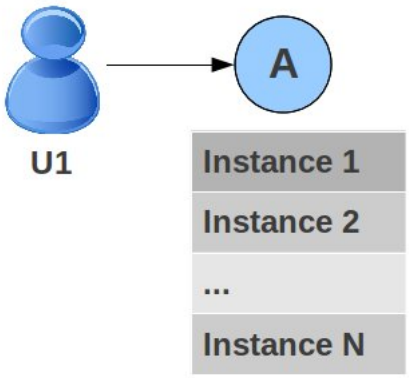


Figure 4.2: *path over one service*

So we can obtain the following information:

1. user identification;
2. service seen by the user
3. next service seen by te user is null because there is no more step in the exploration like for istance seen in it.
4. instance seen in the service
5. timestamp that identifies the search step in the current user session

So the structure of the database for saving the log can be seen as the following:

user_id	session_id	service	item_id	next_service	next_item_id	timestamp
U1	1	a	2	b	2	04/19/12 10:11 AM
U1	5	a	1	*NULL*	NULL	04/19/12 12:12 PM

Figure 4.3: *database structure*

In figure 4.3 you can see the representations, in terms of data to be saved on database, of the explorations described above, respectively in figure 4.1 for the first record and in figure 4.2 for the second record

Given the structure of logs, an important part is now to show how these logs can be grouped into two categories: *random* and *real*.

Random-log

We created an algorithm that automatically generates paths on graphs services.This was done to get a good set of data to work on.

The question now is: how reliable are these paths?
The answer is not simple. This method is the most effective way to perform a stress test on the implemented algorithms in order to have a wide variety of possible paths .

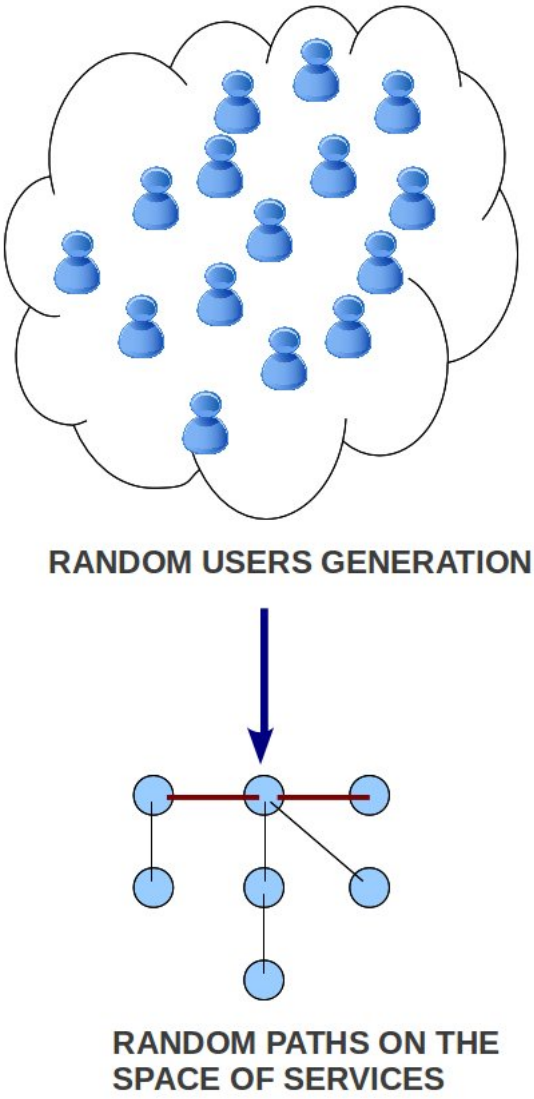


Figure 4.4: *random generation of logs*

In addition, the paths generation algorithm respects the logical ordering of the

arcs between services, so we tried to create a graph as close as possible to a real space services in order to obtain paths as close as possible to real person.

At the implementation level, the structure of the state space (with constraints) is stored in the database by entering the source and destination of each arc.

The Java class PrefGenerator.java draws connections between services and create paths so creates the logs. In fact, the database will contain the complete log obtained as:

1. generating random user
2. for each user the generated paths
3. session identified by the timestamp

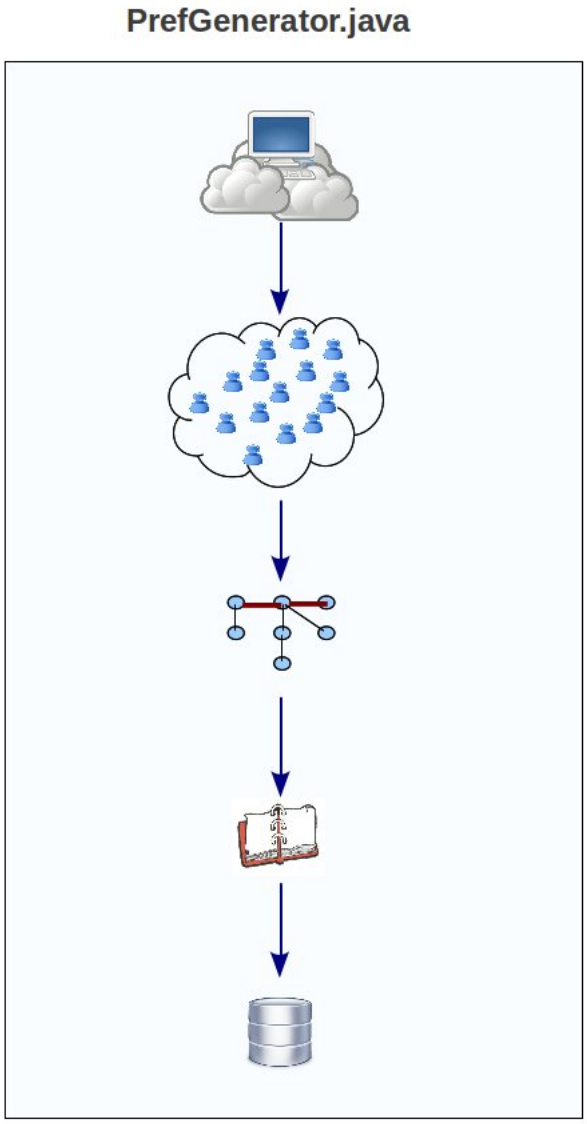


Figure 4.5: *random generation of logs detailed*

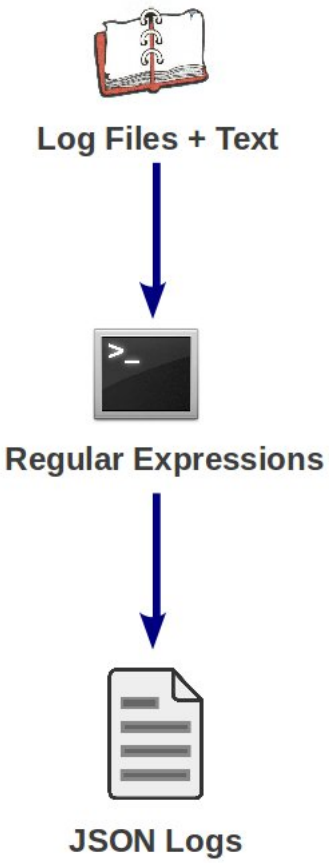
In this way we can create all possible log randomly but with a logic that makes them as close as possible to the real ones. Implementation details of the algorithm and database management will be explained in chapter 4.

Real-log

The real logs come from paths made by users on real services such as those offered by Seco. These logs are saved in a text document with parts of JSON code. In order to use these logs we have:

1. filter files using regular expressions to divide text portions and JSON-code

2. eliminate the uninteresting parts like events on "mouse moves on the screen".



real log

3. extract the names and instances serivizi seen.

4. build paths for each user with their sessions.

5. put it all in the related database

This is all done by a Java parser (getPrefFromLog.java) that takes files filtered by unnecessary parts and "translates" routes stored in the JSON in database queries.

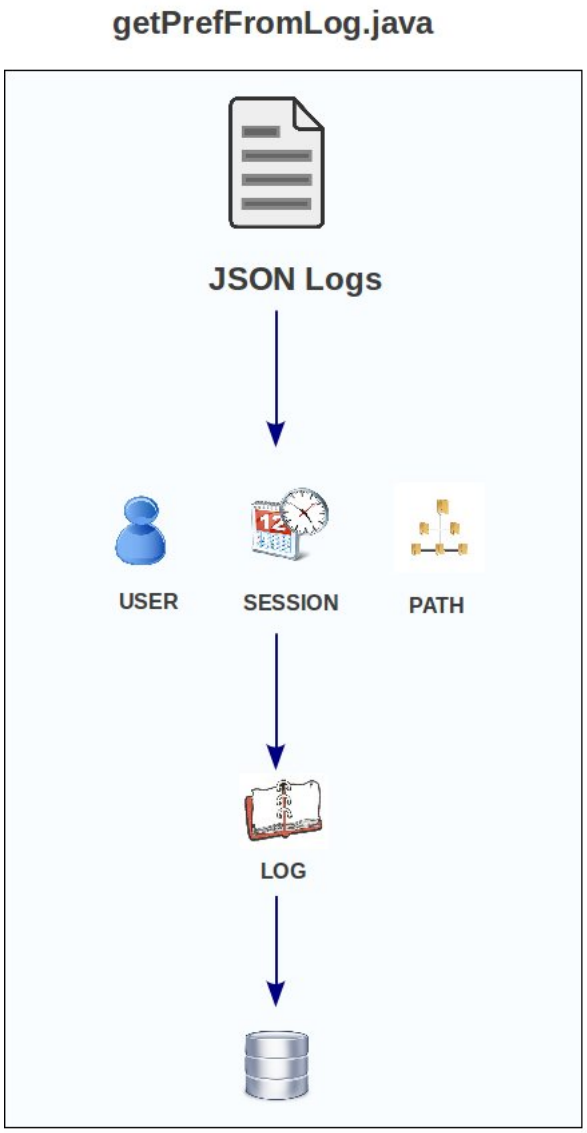


Figure 4.6: *getPrefFromLog.java*

4.1.2 From logs to suggestion

We structured logs to support multi-domain search features and we saved all the necessary information in the database. After that we have obtained the set of inputs needed to find recommendations. Given the input in the correct form, it is possibile, through the recommender system, to obtain the reccomendation.

Figure 4.7 shows the various steps.

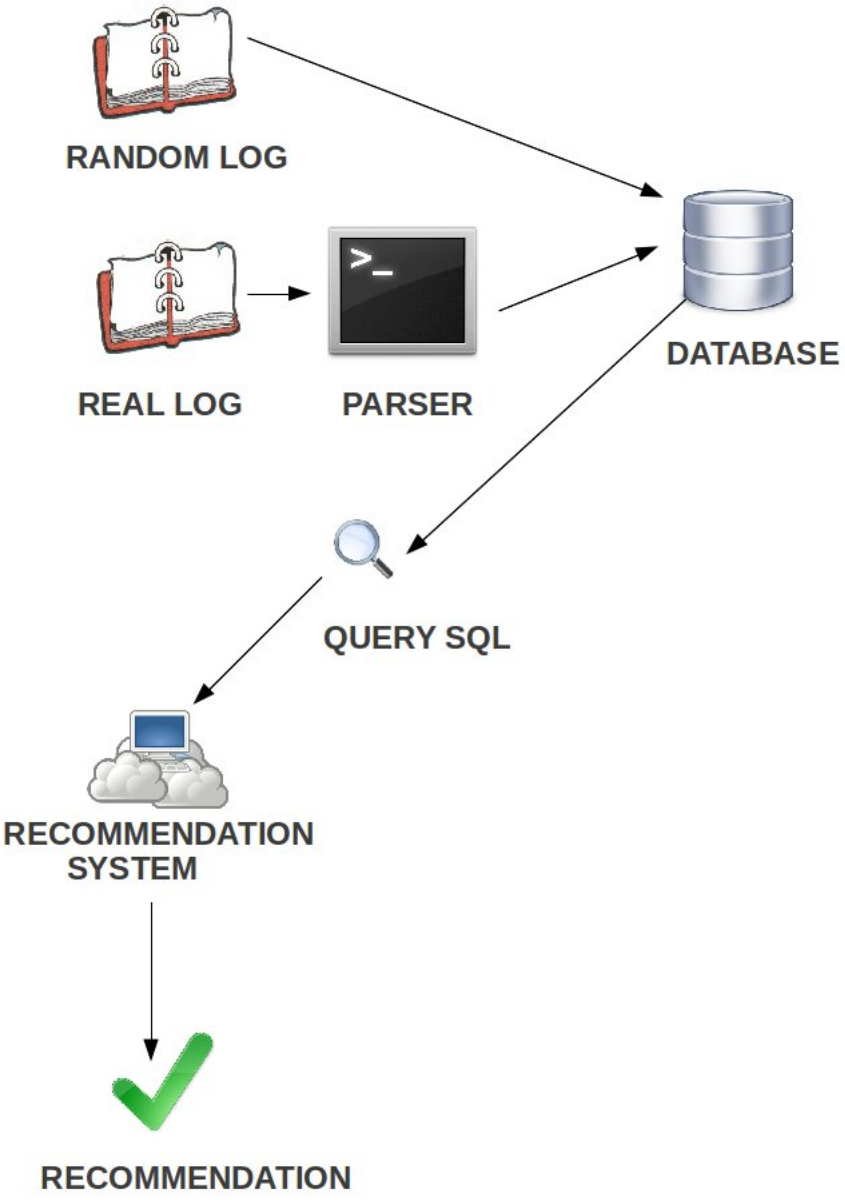


Figure 4.7: *complete process*

The implemented algorithms present, at high-level view, the same type of process from the input data (log) to the output(recommendation).
Entering in the details of three algorithms is possible to notice some difference that is good to expose. All implemented algorithms take as input a set of data that are the logs in the computable form. Then return the output as the recom-

mendation of what is the best next service at every step. What really changes in the implemented algorithms is the process inside them.

Statistic

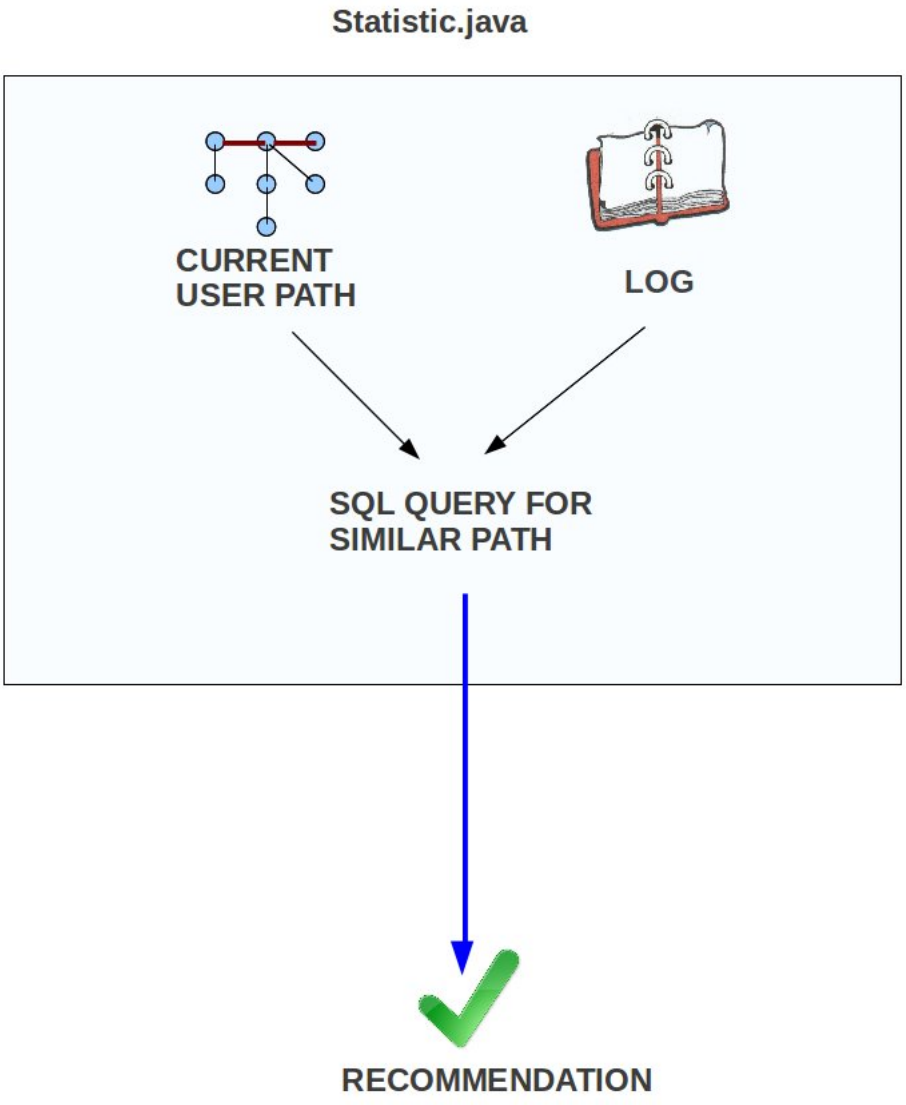


Figure 4.8: *Statistic.java process*

As shown in figure 4.8 the class Statistic.java takes as input the log and the current path and calculates the recommendation counting the occurrences of the next services obtained as looking what is the next best service by counting the occurrences of next-services starting from the current service seen by the user.

Machine-learnig with Mahout-Taste

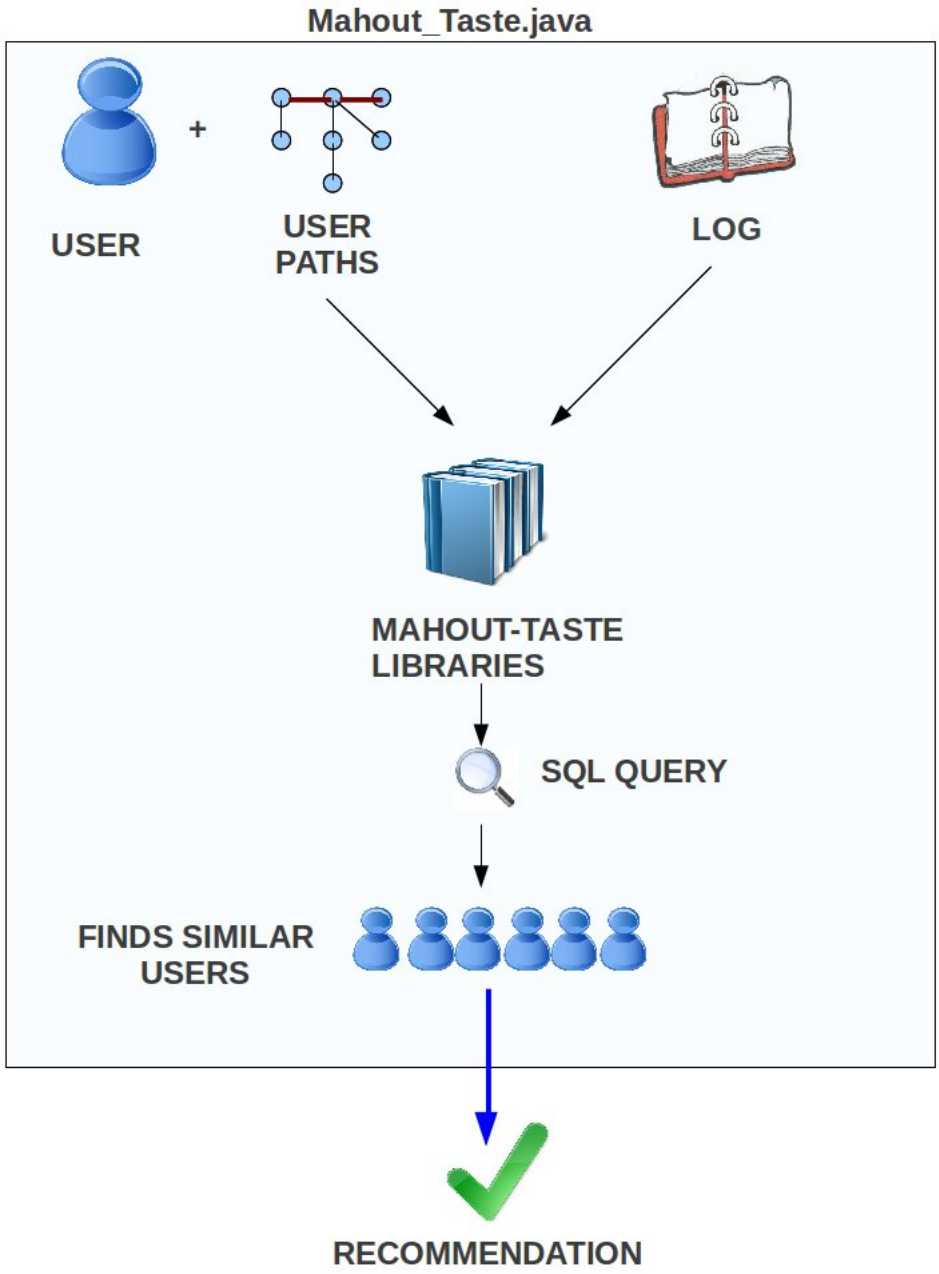


Figure 4.9: *MahoutTaste.java process*

As shown in figure 4.9 the class MahoutTaste.java takes as input the logs and the user with his paths and, thanks to the mahout-taste libraries, finds which are the most similar user referring the current user an then calculate using again the mahout-taste functions the recommendation.

Refined

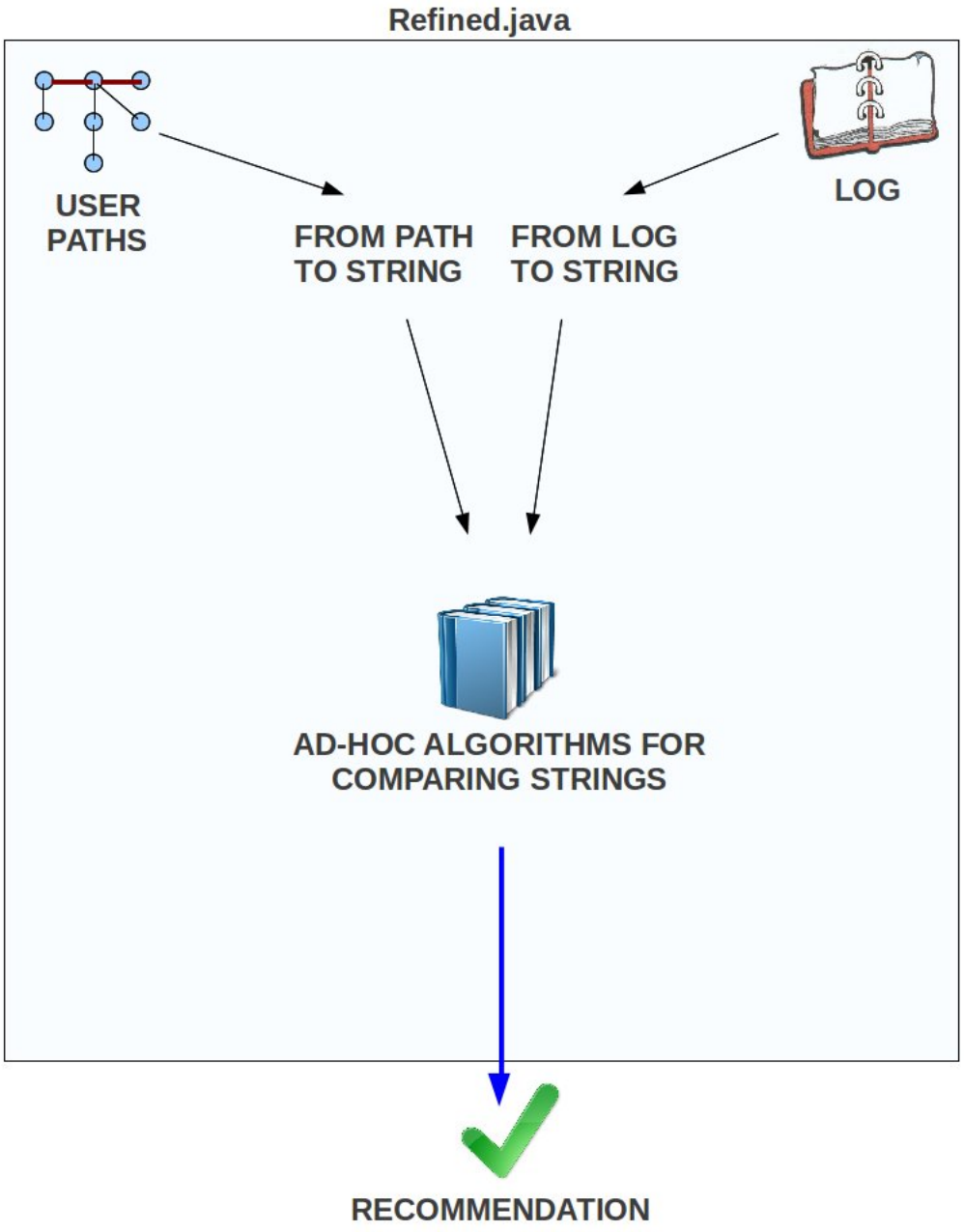


Figure 4.10: *Refined.java process*

As shown in figure 4.10 the class Refined.java take as input current user path and log and convert both into string and thanks to the ad-hoc implemented algorithms for stirng comparison can generate the recommendation.

Table 4.1: *Execution time in milliseconds*

Algorithm	Execution Time 1M records	Execution Time for 2.5M records
Static	500	1068
Mahout	2256	5347
Refined	56000	141236

4.2 Experiments

The goal of these experiments is to compare the performance of each algorithm on the same dataset.

The former test done is the calculus of the execution time. The execution time has been calculated over the 2.5 millions record randomly generated by our script. In a real application this time can be reduced by using a distributed version these algorithms or by other optimization (record index, ...)

The latter test has been the comparison of the outputs of each algorithm. This test has been made for show the different suggestions made by each strategy.

4.2.1 Random exploration

We have generated a random graph with ten node and twenty oriented connection. Each node represent a web service or another datasource where the user can extract information and select an item. The connections represent the possible expansion of each service.

Each exploration path generated by this graph is a simulation of a real exploration of a generic user. The exploration has been generated as follow: the algorithm chose a random service, then it chose a neighbour and "jump" on the next service. The number of jumps is a number between 1 and 5. If there are any path aviable, the program end the exploration of the current user and generate another path on the graph.

After the generation of these paths we have run all the implemented algorithms on dataset of different size. The goal of this test is to show how much time one user could wait to obtain a suggestion.

The table 4.1 shows the execution time of each implmented algorithms.

It's not surprising that the former case is the fastest one. The main operation is a simple query on a database. This operation has been optimize using two index on colum userid and serviceid. The data we have obtained this way are shown with a simple postprocess.

The execution time of the second case is very good, because in a real application we can show data and suggestion to the user at the same time. Mahout is a stable library with ten years of continue development and this suggestion-engine is optimized, as expected.

The latter case has the worst execution time. The sugestions made by this

algorithm can't be presented with the data. The long execution time is due to the low code optimization. The suggestions made by this algorithm have to be shown after the presentation of results obtained from the webservice chosed. Some multi-domain search engine do so:in the example exposed in chapter 2.1 we have shown a suggestion made by Expedia. That suggestion has been made in 34 seconds.

4.2.2 ad-hoc Exploration

We chose this dataset to show the difference between each algorithm. We want some suggestion for the next step for the exploration of user 1

Figure 4.11 show the dataset.

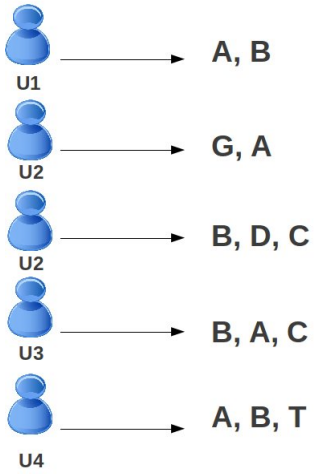


Figure 4.11: *ad-hoc dataset*

User 2 has made two exploration in two different session. This is a real case, because every user can make more than one search using the same account.

The suggestions made with the first alghorithm are (service/rank):

T/1, A/1, D/1

These suggestions do not care of the path chose by the user 1. These suggestions show only that t, a and d were visited one time after one user visited the service b

The suggestion mades with the second algorithm is (service/rank):

C/1

This suggestion does not take in advice that service b is never followed by service c. Service c could be not conneced to b, but is suggested because user 2 and 3 has visited this Service together with service a and b.

The suggestion made with the latter algorithm are (service/rank):

1. T/1.0, A/0.5, D/0.5 for Levenshtein algortihm

2. T/1.36, A/1, D/1 for Exponential algorithm

These suggestion take care that service t, a and d are visited after service b (most probably they are connected) and the algorithm rank these service based on the similarity between the search path of user 1 and the search path made by other users.

4.3 Evaluation

The *purely statistic approach* has fast execution time, and the suggestion made are quite good. These suggestion suggestions do not consider the user's preferences and therefore it is not the best choice for obtaining good quality recommendations.

The *Machine-learnig based approach*, implemented with Mahout-Taste, is fast and take care of the users preferences, but it doesn't take care of the existing connection between services. Mahout can suggest some services for the next step of the exploration but we have to chose between these suggestions and use these that are connected to the last service selected by the user.

The *refined approach*, ad-hoc for multi-domain exploratory search, is the most accurate one but is very slow.

Chapter 5

Conclusions

> One of the main causes of the fall of the Roman Empire was that, lacking zero, they had no way to indicate successful termination of their C programs.
>
> Robert Firth

5.1 Summary of the work

Nowadays mono-domain search engines use recommendation techniques to provide more information tailored to the user. This task becomes much more complex in the case of exploratory search over multi-domain sources. Our goal has been to apply the existing recommendation algorithms in multi-domain environment. We have proposed three different strategies for the multi-domain case:

1. *purely statistic approach* that is based on a statistical computing of the services seen by users
2. *Machine-learnig based approach* that is a refinement of the previous but takes the similarity between users
3. *refined approach* that compares the paths converted to string and find similar sequences

.

To validate the proposed approaches, we have performed two esperiments:

1. tested the implemented algorithms with a randomly generated dataset to evaluate performance.
2. extracted a set of data, from real logs, for testing the algorithms in real case.

Finally, we have evaluated the results obtained and then we have analyzed the quality of the contributions of the different techniques in case of multi-domain exploratory search.

5.2 Conclusion

In conclusion we can say that the recommendation techniques used today in mono-domain search engines can be integrated and extended to cover multi-domain search.

In this thesis we have integrated the techniques of search-based and collaborative-filtering successfully and we have demonstrated the possibility of this kind of integration.

Each algorithm implemented can be used in one specifc application. The former algorithm (static.java) is very fast but the suggestion may not be correctly addressing the user profile. This is because it is based on general statistics of all the users. The second algorithm (mahout-taste.java) is fast enough to show the suggestions together with the data retrived. Unfortunately in some cases the suggestions are not available for the user because the obtained services are not connected to the services explored by the user. The latter case (refined.java) is the most accurated and targeted on the user profile, but it may require longer waiting time so suggestions may be shown with some delay to the user.

5.3 Future work & extensions

We have analyzed the techniques of search-based and collaborative-filtering. These are only a part of the recommendation techniques existing in literature. A future development will be the analysis of other techniques not discussed in the thesis. Another future work is the integration of the implemented algorithms into a multi-domain search engine such as SeCo to verify the potential of the studied techniques within a real system.

The next extension we can made is to realize a system that provides suggestions at instance level by highlighting the best result instances and not only the services. The most difficult issue to tackle in this direction is that typically there is no way to uniquely identify a result returned by a set of web services in multi-domain search systems.

Bibliography

[1] Alessandro Bozzon, Marco Brambilla, Stefano Ceri, and Piero Fraternali. Search computing. chapter Information exploration in search computing, pages 10–25. Springer-Verlag, Berlin, Heidelberg, 2011.

[2] Stefano Ceri and Marco Brambilla, editors. *Search Computing: Challenges and Directions [outcome of the first SeCO Workshop on Search Computing Challenges and Directions, Como, Italy, June 17-19, 2009]*, volume 5950 of *Lecture Notes in Computer Science*. Springer, 2010.

[3] The Apache Software Foundation. *Scalable machine learning and data mining*, 2012 (accessed September 16, 2012).

[4] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[5] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, March 1997.