haute école
neuchâtel berne jura
arc ingénierie
www.he-arc.ch

Haute École Arc

Report

Social Recommendation System

Autumn project

Project ID :
216 - 2013

Student :
Diego Antognini INF3dlm-a

Supervisor :
Dr. Hatem Ghorbel

Duration :
17 September 2013 - 24 January 2014

Version :
1.0

Last modification :
24 January 2014

Abstract

The website `http://www.gokera.ch` proposes a list of events in the French-speaking part of Switzerland. Its purpose is to recommend events for users taking into account the ratings of other events. In this project, we are interested only in event ratings and use a hybrid recommendation system based on two approaches which mixes the user similarity and the decomposition of a product in features (collaborative filtering and content-based). In the case where we do not have user's data it is impossible to realize an efficient recommendation. It is the reason why we experiment the add of a social dimension based on *Facebook*. The addition of this social dimension allowed us to get information of the user related with our features found during the event analysis process and this gave us the possibility to recommend events even if the rating history is empty. In a general way, the system works but has to be adjusted to run on a production environment, especially the text analysis. As for the pertinence of the recommended events, we cannot device for sure because we do not have some set of data for testing but we will have some during the challenge *ESWC-14 Challenge: Linked Open Data-enabled Recommender Systems*[1].

Le site web `http://www.gokera.ch` propose une liste d'événements en Suisse romande. Le but de ce site est de recommender des événements aux utilisateurs en prenant en compte leurs évaluations effectuées sur d'autres événements. Dans le cadre de ce projet, nous nous intéressons à la recommendation d'événements et utilisons un sysème de recommendation hybride basé sur deux approches mélangeant la similaritude entre utilisateurs et la décomposition d'un produit en *feature* (*collaborative filtering et content-based*). Dans le cas où nous n'avons aucune données, il nous est impossible d'effectuer une recommendation efficace. C'est pour cette raison que nous expérimentons l'ajout d'une dimension sociale basée sur *Facebook*. L'ajout de cette dimension sociale nous a permis de récupérer des informations sur l'utilisateur en relation avec nos *features* trouvées lors de l'analyse d'événement, ce qui a pu amener à des recommandations d'événements, même si l'historique d'évaluation est vide. D'une manière globale, le système fonctionne mais doit subir quelques ajustements pour être placé dans un environnement de production, notamment l'analyse textuelle. Quant à la pertinance des événements recommendés, nous ne pouvons pas encore nous prononcer en raison de jeux de données que nous aurons lors du concours *ESWC-14 Challenge: Linked Open Data-enabled Recommender Systems*[2].

[1] More information on `http://2014.eswc-conferences.org/important-dates/call-RecSys`.

[2] Plus d'informations sur `http://2014.eswc-conferences.org/important-dates/call-RecSys`.

Contents

Chapter 1

Introduction

1.1 Context

The recommendation system is used in the domain of the online sales platforms. The purpose is to suggest to the visitor a product which might interest him for a purchase. For this, they collect all the possible data they can when a customer is visiting their platform : the products viewed, the ratings and opinions consulted, the purchases and their ratings etc. The final purpose is to find in away to take into consideration all this data and find products that might interest the customer and for that, we use recommendation systems. There are several kind but all of them have some advantages and disadvantages. To improve the quality of a recommendation, we can mix several of them in order to fill their weaknesses.

The website `http://www.gokera.ch` contains events in the French-speaking part of Switzerland in a various field of a category. The present purpose is to recommend events to the user depending on his rating ("I like, I dislike") on other events in which he has participated. Finally, it would be possible to recommend events for a group of users, taking into account the ratings of each user. The producers can create events and have to link it to a category among several proposed.

1.2 Goal

The objectives of this project is to realize a hybrid recommendation system mixing the collaborative filtering and the content-based approaches and try to fix their weaknesses with the social dimension. The recommendation will be only focus on a single user. Moreover, to try to fill their weakness we will take into consideration the *Facebook* profile.

First, we have to create a simply prototype of a website that will contain all the events in *Gokera*. Then we will realize a features extraction system for the events and the *Facebook* profiles. Finally, we have to implement an hybrid recommendation system in which we integrate the social part.

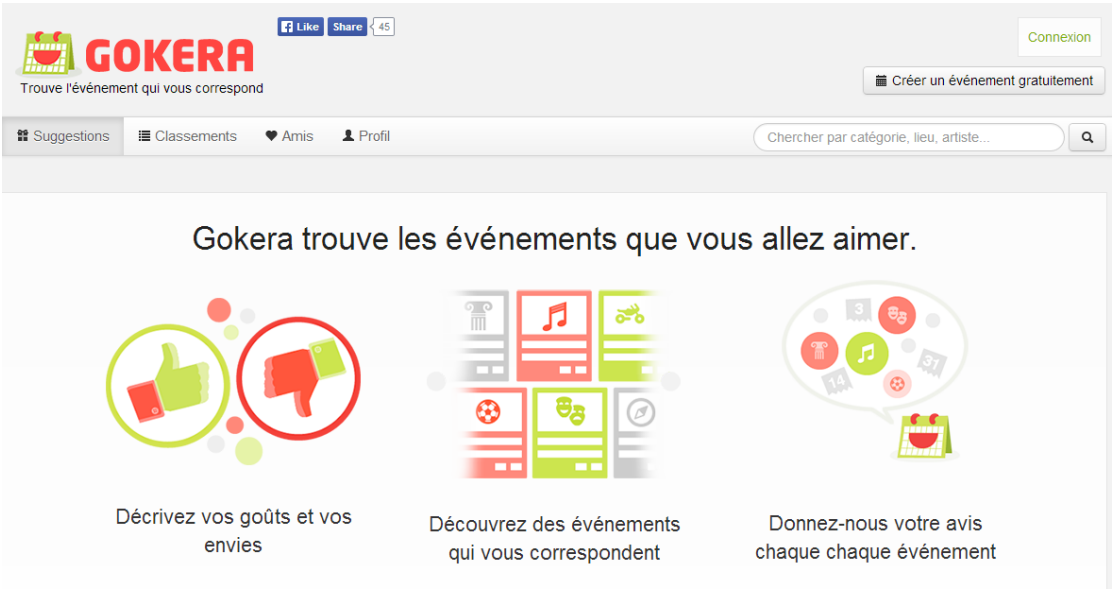

FIGURE 1.1: Main page of the website *Gokera*

1.3 Our proposal

In this project, we experiment with a hybrid recommendation using two approaches : the collaborative filtering and the content-based. The pros and cons of some existing recommended systems in which we are interested and the one we use as a basis is described in chapter 2. All the next chapters include our proposal.

The chapter 3 talks about the event analysis. We find why this step is important, how the description and the website is analyzed, why their analysis is different, what we focus our attention on and why and finally how we compute the importance of a word.

The kind of hybrid recommendation system we use has still a weakness so we experiment a way to fill it with the add of a social dimension from *Facebook*. We take into account the user's profile and those of his friends. All the details about the user's activity, the features extractions and their weight are described in chapter 4.

At this stage we have all the user's features and all the events' features, we use our hybrid recommendation system, which takes into account the *Facebook* profiles to find the products which might interest the customer. The whole system is stated in chapter 5.

The chapter 6 contains all the conception part : the database, the architecture, the class diagram and the performances. It is followed by the problems encountered. To end up, we compare the initial planning with the final one and highlight the differences between them in order to have better estimation for the next project.

At the end of the document, in chapter 9, we analyze the different results we have obtained in purpose to give some leads to improve this project.

Chapter 2

Existing recommendation systems

Recommendation systems are mechanisms which will propose to the user some products that might interest him. It will be realized on the basis of the user's predictions based on his consumption history (either purchases, ratings or searches).

In this project, the consumption history will be based on event rating like *Facebook* with "*I Like*" and "*I Dislike*" actions. There are several approaches like collaborative filtering, content-based, demographic, utility-based and knowledge-based [1]. Only the first two are presented in a simplified way because there are the most popular.[1]

2.1 Collaborative filtering

The system that uses the collaborative filtering is focused on similarities among users and products. Several versions of this approach exist but only the memory-based is presented here.

The idea is that for having a good recommendation, we must find other persons having similar tastes than the user has, in order to propose to him products that might interest him. So we are in a system where users depend on each other.

We can identify some weaknesses, mainly the user's activity. To have the most efficient system, the user must be active rating a maximum of products so that we could determine the most accurate profile. Another major problem is when a product is added, as long as nobody has rated it, it will be never recommended to other users. Finally, in the case where the user have some special tastes, the recommended products will be only slightly targeted [3]. This is mainly due to the lack of product decomposition in features.

[1] For more information, we invite the curious reader to read the article [2].

2.2 Content-based filtering

The content-based filtering approach is focused on the products decomposition in features. Through these features, we can define the similarity among the products. When a user consumes one, he will rate it in a certain way, all the features related with the product. This will allow the system to suggest to him other products with similar features, contrary to the collaborative filtering approach, the user is independent from the others.

The inconvenience of this approach is that we will face the problem of over-specialization because the user will see only the products which are similar to those already rated and so, he will not have the occasion to discover some other ranges of products [3].

2.3 Hybrid Recommender Systems

This last approach consists in a combination of several recommendation systems in order to solve their inconveniences and get better performances. Different versions exist like weighted, switching, mixed, feature combination, cascade, feature augmentation and meta-level [1]. Only the first one is presented.

The weighted approach combines the collaborative filtering system and content-based system. This combination allows to get better results because we combine the products rating of the user and all the features related as well as similar tastes of users. So the purpose is to find a way to take into consideration the features that the user likes specially and those of the similar users.

In order to recommend products in a suitable way which could contain some features liked by the user and some others a little less, we will assign a weight to each one of those. This will allow improvement in the accuracy of the recommendation system and satisfy the problem of the particular tastes and the over-specialization. Moreover this combination solves the visibility problem of the new products added.

Unfortunately, to combine two systems having the same defects will not correct them so the problem of the new user and his activity is still present.

To correct this last defect we could try to add a new dimension to this recommendation : the social dimension. The chapter 4 is dedicated to it.

Chapter 3

Event analysis

The event analysis is one of the most important part even the most. It is the main core of the recommendation because if its quality is bad, the recommended products will not be targeted, no matter what the user's activity. The event analysis is essentially based on the events available on `http://www.gokera.ch`, of the 18^{th} of December 2013. Some examples of events and some of their attributes are showed at the Table 3.1.

TABLE 3.1: Some examples of events with some attributes

object id	2284	2644	5568
name	Foire de Brent	Bretaye By Night	Bowling Challenge
website	None	None	`http://www.villars.ch`
price	0.0	65.0	5.0
# views	7	4	6
# likes	0	0	0
# dislikes	0	0	0
start date	10/01/2014	10/01/2014	10/01/2014
description	La Foire ...	Venez passer ...	Tournoi de bowling ...
category	Marché/Brocante	Rencontres/Animation	Sport

3.1 Corpus

Let's start by defining a corpus in the linguistic domain, Wikipedia has a good one[1] :

> A corpus is a large and structured set of texts. They are used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules within a specific language territory. A corpus may contain texts in a single language (monolingual corpus) or text data in multiple languages (multilingual corpus).

[1]The citation is from `https://en.wikipedia.org/wiki/Text_corpus`.

For each event we have a description and/or a website. At this time where we are writing this line, we have a total of 3140 events. Among them, we already have 798 (25,41%) which don't have neither a description nor a website. Unfortunately for those, we cannot do anything, so they won't ever be recommended.

One needs to know in this project is we interest only in French texts. Although there could be some descriptions in another language, those will be ignored. With regard to the websites, we are interested also only in French content. In our case, the corpus is monolingual. In order to clarify, I define the terminology "document" which can contain several texts, because the set of all the texts from a website is grouped in one single document.

So our corpus has a set of documents which represents a description or the content of a website for an event. But those are not added directly in our corpus because they must be preprocessed.

3.2 Preprocessing

Before giving the document to the text analyzer, some verifications have to be done.

3.2.1 Language detection

The language detection tool used is based on calculus and frequency comparison of profiles of *N-gram*. The system used is small, fast and robust and is tolerant of textual errors [4]. As stated before, we are interested exclusively in the French language. Of course, the longer is the text, the more accurate is the language detection. According to some testings, the tool is reliable as we have a small number of words if they are not outstanding. In the case where a word is special, it is preferable to have a longer text.

It is possible to have several languages inside a text, for example in the case where the word has an English connotation inside a French text or simply a literal translation. Moreover, the more elements in French there are, more chance we have to detect the French language. Having words in another language does not degrade the quality of the analysis because their origins will not be found and so they will be ignored.

3.2.2 Description

A description is accepted as a document if and only if it is not empty and it is recognized like a French text. There is no more extra operations to realize.

3.2.3 Website

The website analysis has to be done in a rigorous way. First we have to define the *HTML* tags in which we interest in. The Table 3.2 shows those we have kept as well as the reason.

TABLE 3.2: *HTML* tags kept in which we are interested in their content

Tag	Remark
,<p>	Contains in a general way texts.
<h*>	Represents a title, considered the most important element.
	Often used for the menus. We can find into important elements.
, <i>, <b>	To highlight a particular word.

For the second step, if we make the hypothesis that the website url points at the root, we can wonder "Is only one page sufficient ?". The answer is clearly no for the following reasons :

We can arrive at a page which contains:

- Only advertisings and a link which allows us to go to the desired content. It is not pertinent;
- A language menu, in this case except for the languages themselves, there is no content;
- Contents in another language but propose to switch the language with a tab, generally in a corner. The current content is not interesting because it is not in French.

Although most websites are not concerned, we have to maximize our chances to parse the content of documents which could represent an event and so, find a solution.

Before proposing a solution, the Figure 3.1 is a representation of a website. In this illustration, a particular website has been decomposed in all of its links (that point to a textual resource) as a directed graph (that can be connected in certain cases). Links represented with orange circles are links which point out the domain (in nearly all the cases, this breaks the connectivity property) whereas those in white point at internal resources.

To start, we consider it is extremely important to keep only links that point at the same domain and at textual content (an easy way to find it is to use the field content-type of the *HTTP*[2]). This is for two reasons : the first, the pertinence of the information will diverge and secondly, if $depth \to \infty$, we will visit the entire web. All the while taking into account the last remark, if we restrain ourself to stay in the same domain, even if

[2]For more information, you can consult the specification of the protocol *HTTP*, `http://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-01.html`.

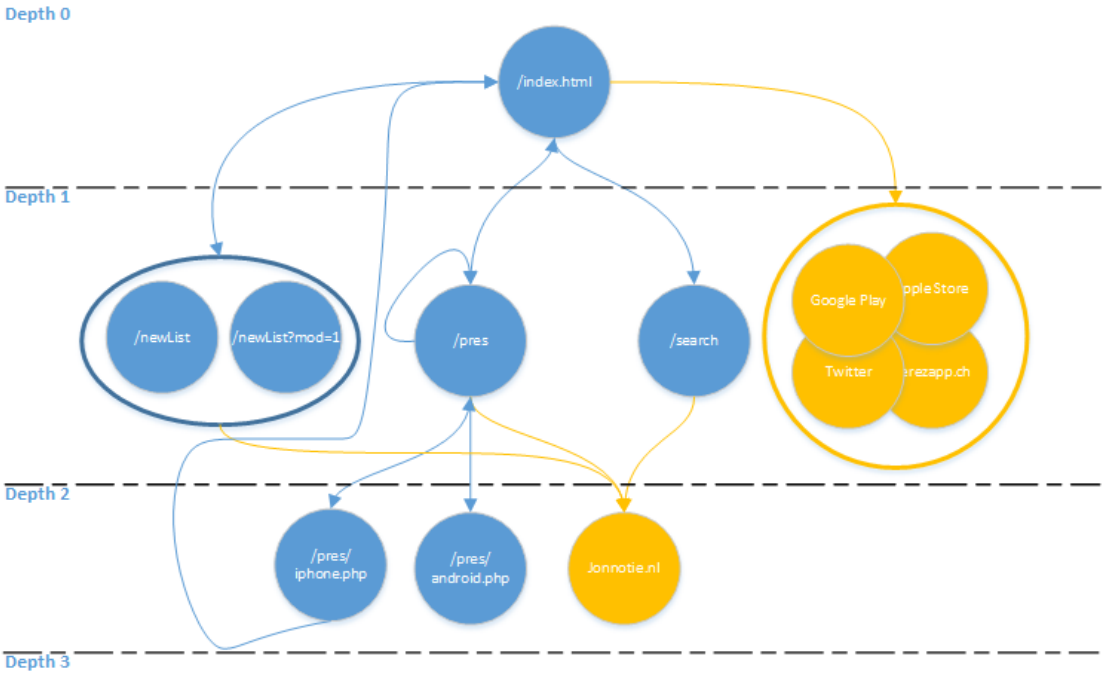

FIGURE 3.1: Representation of the website `http://www.cartor.ch` as a directed graph

depth $\to \infty$, we will have, anyway, a finite set of urls.

The next step consists in transforming all relative links (generally those which stay in the same domain) in absolute links. We have to be careful how a link is represented :

Let *xxx*, an existent website page. It can be represented as the following forms :

1. *xxx* or *./xxx*, in this case we have to add *xxx* to the parent;
2. *../xxx*, we have to add *xxx* to the grandparent. Be careful if there are some "..";
3. */xxx*, the page is at the same level as root.

Second to last step concerns the url rewriting and the parameters *GET*. For the first, the main problem is :

Let the */xxx*, is it a file or is it a folder which points in its own index ?

Unfortunately, there is no way to know it without doing a query to the server. If we send a query to the server, we will know explicitly if it is a folder because it will be represented by the format */xxx/* or by */xxx* if it is simply a file.

For the second one, if we observe again the Figure 3.1, we are aware there are two pages grouped : /newList and /newList/?mod=1. In some cases, the content of the different pages can be the same but in this precise case, it is not. In this example,

the difference is not flagrant but we might have an url like "/display?id=x" or "/display?include=xxx" in which their content change radically according to the parameter. So the solution is to keep those parameters. Concerning the anchors (the symbole # in the urls), they must be removed in order to clean the url and because they do not contain significant information.

Now we have the links which point at the same domain and at textual content. But there is still a problem : if we want to parse all the website until a specified depth, we will end in an infinite loop, extracting the urls of the same pages and revisit them again. In order to converge at a finite set of urls, we have to store each visited url, after "cleaning" them (remove the anchors, put the "/" if it is a folder, etc.).

To finish, we have to create an hash table in which the hash function is defined such as $h(x) = x$. With this method, we solve the problem and have a list of urls but they are not unique. Because if we have /newList/ and further another /newList/index, we have no chance to know that /newList/ redirects into the index page (defined in the server configuration) and inversely, from /newList/index we cannot know it is the index page for the folder /newList/. So unfortunately, we have to accept a slight redundancy but in generally they are special cases.

Another alternative would be to hash the content of each page, store their hashes and compare them with the others. The disadvantage of this technique is that we have to get the content of all visited pages and this operation is usually slow. Moreover, it is not perfect because if a page contains some dynamic content (current time, time of generation, some advertisements, etc.), the product hash will not be the same. For those reasons, we have preferred to keep the first solution.

3.2.4 To add to the corpus

At this stage, we have descriptions as well as the content of the website that can be added to the corpus. But before adding them, we analyze their textual content to remove some parts of it, in order to decrease noisy text and so to improve the reliability.

The parts to remove are essentially the symbols, punctuations, sentence tags and all the unknown words.

Here is an example of non desired content : . # ! ... ? + " * % & / ( ) ; : _

3.3 Text analysis

We have a corpus which contains the set of documents related to each event. At this step, we have to find the most important words for an event, mix them between the description and the website if necessary, in order to extract our features for the recommendation.

Several ways exist for this but we have opted for the approach *(tf-idf)*. There are some existing different implementations but we have chosen this proposed at the reference [5].

3.3.1 Tf-idf

Tf-idf is a statistical method which allows to calculate the importance of a world inside a document inside a corpus. It is decomposed in two parts : the frequency calculus of a term and this for the inverse document frequency. The *(tf-idf)* is the multiplication between the two intermediate results.

To compute the frequency, we use the Equation 3.1 :

$$tf_{ij} = \frac{n_{i,j}}{\sum n_{k,j}} \tag{3.1}$$

Where $n_{i,j}$ is the number of occurrences of the term t_i in the document d_j and $\sum n_{k,j}$ the total number of occurrences of all terms in the document d_j.

For the inverse document frequency, we calculate it with the Equation 3.2

$$idf_i = \frac{\log |D|}{|\{d : t_i \in d\}|} \tag{3.2}$$

Where $|D|$ is the total number of documents within the corpus and $|\{d : t_i \in d\}|$ the number of documents which contains the term t_i.

To finish, *(tf-idf)* is defined with the Equation 3.3

$$(tf\text{-}idf)_{ij} = tf_{ij} \times idf_i \tag{3.3}$$

3.3.2 Features extraction

We have now all the necessary tools for the features extraction for the events. Before computing them, we have to define the terms to take into consideration[3]. We have decided keep only the nouns and the proper names. In a general way, the rest is not pertinent. The only case where we could hesitate would be this of the adjectives. Let's take the following example :

The little catholic church in the village has been painted in red.

In this example, we can note that the adjective "catholic" could be interesting but not "little". The case where adjectives could be taken into consideration is quite rare and as we cannot distinguish which of those are pertinent with this methods, we do not keep them.

[3]The full list of the different kind of words is available on `http://www.ims.uni-stuttgart.de/institut/mitarbeiter/schmid/french-tagset.html`.

Once the importance of all terms of an event is computed, we have to keep the $k, k \in \mathbb{N}^*$ most important. For this, we sort them in ascending order by frequency. The smaller the frequency is, the rarer is the term and so, more important. In the case where the feature is found in the description and in the website content, we keep only the feature with the highest *idf*.

At the moment, we add at the corpus the description and the website content if present. In fact, this affirmation is slightly false. We have opted for an approach which defines that if we already have k or higher key words within the description, we will not take those of the website. On the other hand we add the set of key words of the website. This approach highlights the description but has for consequence to create a side effect because if we have exactly $k-1$ key words in the description, we have high chances (if the website is not in French and it has some contents) to decrease the quality of the features of the event due to a high likelihood to add a large content.

The other imagined way was to define a function which returns the number of words to fetch in the website during the parsing, in order to gain time. It would have a similar form than this proposed in 3.4 :

$$f(W_i) = \frac{\max D}{W_i} + \bar{D} \tag{3.4}$$

Where D is the set of descriptions (in key words), W is the entire text of the website and i the event.

This approach was dropped because when we realized some statistics (with 5180 events at this time), Figure 3.2, we had noticed that in a general way, the content of the descriptions is poor (length generally low) and the average was at $7,59$ key words per event. Moreover, the maximum key words extracted was at 84. This is the reason why we didn't keep this approach and we think that the first one is more adapted at this situation.

Other interesting information we have drawn from those statistics :

- $66,91\%$ of the descriptions (2101) are in French;
- Over the 1873 specified websites, only $95,46\%$ of them (1788) are exploitable (valid), independently of the language;
- Among the $95,46\%$ of websites, $54,44\%$ (974) have some French contents with depth at 1;
- Altogether, there are $27,68\%$ of events (869) which have a description and a website with French content.

By adapting those statistics without the empty events, we have respectively : $89,7\%$ (French descriptions), $76,35\%$ (valid website with French contents) and $37,11\%$ (description and website with French contents).

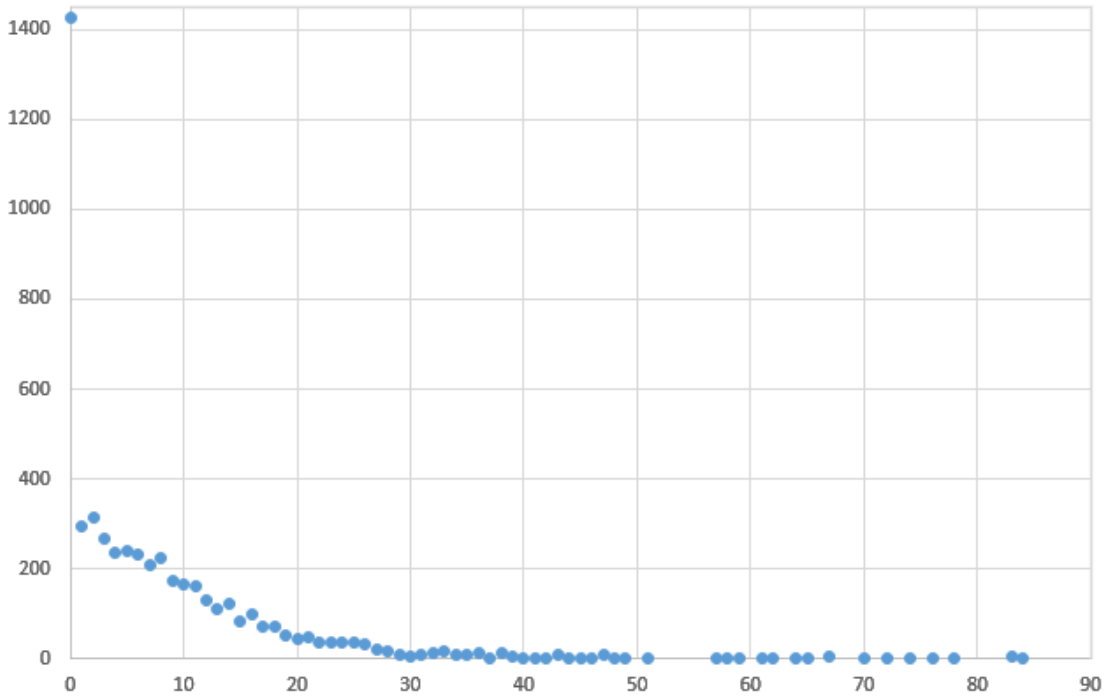

FIGURE 3.2: Histogram of the number of key words in the description, per event.

If we interest in key words extraction, we could say that the average of key words extracted in the description is $10,88$ and if we add the website content with a recursion defined at 1, we have an average of $367,68$. So, we are aware of the content of a website is not negligible.

To finish, we define one or two extra features that those extracted. They are from the category of the event and some of them have two categories. In total, the number of features of an event is in the domain $[1; k+2]$.

For the recommendation, we know where the feature is from and we apply a factor to it, on its *(tf-idf)* in order to weight their origin (description, website or category).

Chapter 4

Facebook analysis

Nowadays the social network *Facebook* is used by many people over the world. It contains a lot of information about their users and it is for this reason that we have considered it could be interesting to include this social dimension in our recommendation system.

4.1 Social dimension

The hybrid approach of the recommendation system used in this project, described in chapter 5, let's appear a not very reliable recommendation when it is a new user or when he is not active. If this kind of users has not still rated events stored in our database, it would be impossible to establish an efficient recommendation.

To satisfy this problem, we collect the *Facebook* profile of the user and look for elements which the user interests in. We have defined the area of research will be centralized on the groups and fan pages, liked by the user.

4.2 Facebook Graph API - Facebook Query Language

Facebook makes available to developers the *Graph API* which allows to have access to the user's data and his friends' data. We can access it in two ways : with *GET/POST* or with their language *FQL*[1]. The second one has been chosen because it allows to create more accurate queries and offers the possibility to mix several queries inside one.

We must consider that the user can define the privacy settings on his data and that *Facebook* imposes some limitations on their *Graph API*, for example the access to the user's friends list. The Figure 4.1 shows the principle.

[1]More information about these methods on `https://developers.facebook.com/docs/graph-api/reference/` and `https://developers.facebook.com/docs/reference/fql/`.

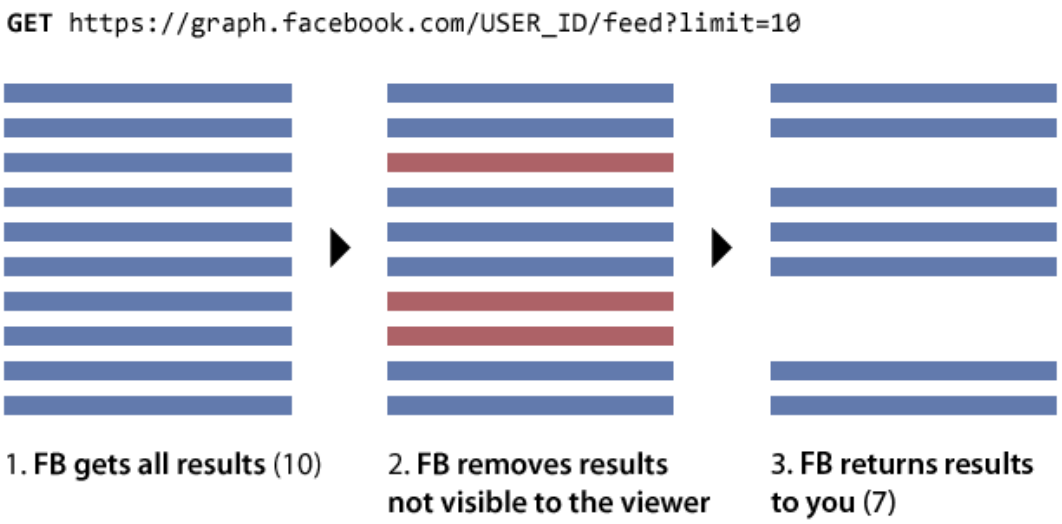

FIGURE 4.1: Representation of a query processing with the privacy settings on *Facebook Graph API*. The method used is with *GET* but it has the same behavior with *FQL*. Source : `https://developers.facebook.com/blog/post/478/`.

To define the permissions we need, we have to create a *Facebook* application in which we precisely state what we want to access. Afterwards, when the user will be connected to it, *Facebook* will ask him to accept that the application has an access to his data we have defined. If he accepts, we will be able to generate a token that we will send to *Facebook* when we will submit queries[2]. Otherwise, the social dimension will not be taken into consideration for the recommendation.

4.3 Fetching data from a Facebook user

First we collect the current user's profile and his friends' profiles. We interest in his friends too because if the user is not very active, the defect of the recommendation system will be still present. So we fetch also his friends' data because there is a high likelihood that the current user has at least one friend who has a strong activity.

For each profile (friends and current user), we interest in groups and fan pages liked and we extract the related features in the same way in the chapter 3. Then, we focus only to the friends who have the same features we have defined thanks to the events analysis. With a real profile of *Facebook*, we have obtained $29,08\%$ features related (1526 over 5248). Among these, we have to take into account that some features are insignificant.

[2]A global view of permissions is available on `https://developers.facebook.com/docs/reference/login/#permissions`. For more information about the permissions by field, you can consult the following link `https://developers.facebook.com/docs/reference/fql/`.

Even if we have a good correlation with the features found, we have a lot of noise. To improve the quality of the features extraction, we can add a synonym dictionary for all the features we have defined and use it to extend the area of relation detection between a word and a feature. With this additional system, we will obtain a more efficient additional system but unfortunately we could not try this system due to a lack of time.

At this step, we cannot keep the features of all friends because the main target is the current user. We define the k most active friends. To compute the activity of a user, we refer to all of his posts written by the user. We fetch only the posts on the user's wall because it's not possible to have access to the messages posted in another wall (friends or others). To calculate the most accurate activity we fetch all the posts which can be an accepted friend request, a page/group he likes, an announce of his relationship, simply a message etc.

To estimate the activity, we start calculating the average of the intervals between each post, from the date of the first one to the current date, with the Equation 4.1 :

$$\overline{X} = \frac{today - first\_activity}{|Messages|} \tag{4.1}$$

Where *today* and *first_activity* are dates represented by the format *timestamp* (number of seconds since the 1^{st} of January 1970) and $|Messages|$ the total number of messages.

Then, we calculate the absolute deviation by month with the Equation 4.2 :

$$e_m = \frac{1}{n} \sum_{i=0}^{n} |\overline{X_i} - \overline{X}| \tag{4.2}$$

Where $n, n \in \mathbb{N}^*$ represents the number of months since the first activity, $\overline{X_i}$ the average of the user i and $\overline{X}$ the average of the averages of the intervals among friends' posts.

We keep only the $k, k \in \mathbb{N}^*$ friends with the smallest e_m. Then we sort them in ascending order because we are interested in the most active friends, so the smallest time between each post (e_m).

Finally, we assign the weight of the user's features at 1.0 and for his friends' features, according to the Equations 4.3 and 4.4 :

$$X_k = \{\overline{X_i} : 0 \leq i \leq k\} \tag{4.3}$$

$$Weight(f) = \frac{\overline{X_f}}{max\{X_k\}} \tag{4.4}$$

Where $\overline{X_f}$ is the average of the intervals between each friend's post and X_k is the set of the average of averages of the k most active friends. Finally, the weight is defined by the normalizations of $\overline{X_f}$ by $max\{X_k\}$.

Chapter 5

Social recommendation system

As stated before, it exists several recommendation systems and we are interested in the approaches collaborative filtering and the content based. The first is based on similarities among users and the second on the decomposition in features of a product. If we mix those systems, we fix nearly all their weaknesses but it remains still one : if a user is new or inactive, the recommended products will not be targeted. This is the reason why we experiment to integrate the social dimension described in chapter 4. The approach described below is widely inspired by the reference [2].

The methodology we use is the following :

- To create the basic matrices related to the user's rating, features of products and user's features from Facebook;
- To compute the user-feature matrix;
- To calculate the feature frequency, user frequency, inverse user frequency and finally the weighted user-feature matrix;
- To compute the user similarity matrix;
- To find the k nearest neighbors for a particular user.

Before detailing each step, let's define $\mathcal{U}$ the set of users, $\mathcal{F}$ the set of features and $\mathcal{E}$ the domain of events. To simplify the example of our approach, we define that $|\mathcal{U}| = 3, |\mathcal{F}| = 5, |\mathcal{E}| = 5$ but in fact, we have $|\mathcal{F}| = 5248$ and $|\mathcal{E}| = 3140$.

5.1 Basic matrices

Those 3 matrices are trivial to define because there is no calculation. The only thing to do is to get the information and fill them up. The user-event matrix R has a size of $|\mathcal{U}| \times |\mathcal{E}|$, Table 5.1. In our case, if there is a rating, its value is in the domain $\{x : 0 < x < 6, x \in \mathbb{N}\}$.

	$\mathcal{E}_1$	$\mathcal{E}_2$	$\mathcal{E}_3$	$\mathcal{E}_4$	$\mathcal{E}_5$
$\mathcal{U}_1$	1	2	4	5	3
$\mathcal{U}_2$	3	2	5	-	-
$\mathcal{U}_3$	1	4	-	-	3

TABLE 5.1: Matrix R, user-event

The event-feature matrix F has a size of $|\mathcal{E}| \times |\mathcal{F}|$, Table 5.2. The features are weighted and the value is in the domain $]0;1]$ (more details in chapter 3.3.1) but it could be higher if we decide that the feature from a website or description should be more important and use a factor.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\mathcal{E}_1$	0.2	-	0.8	-	-
$\mathcal{E}_2$	-	0.8	1	-	0.3
$\mathcal{E}_3$	1	-	0.3	-	-
$\mathcal{E}_4$	-	0.2	-	0.4	-
$\mathcal{E}_5$	-	-	-	0.8	-

TABLE 5.2: Matrix F, event-feature

The social user-feature matrix S has a size of $|\mathcal{U}| \times |\mathcal{F}|$, Table 5.3. The features are weighted and the value is in the domain $]0;1]$. More details in chapter 4.3.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\mathcal{U}_1$	0	0.4	0	0.8	0.3
$\mathcal{U}_2$	1	0	0.5	0	0
$\mathcal{U}_3$	0	0.2	0	0	0.6

TABLE 5.3: Matrix S, user-feature *Facebook*

5.2 User-feature matrix

With the matrices R and F, we are able to compute the correlation between a user and a feature. We take into account only ratings greater than P_t in where $P_t \in \mathbb{N}^*$. The user-feature matrix P has a size of $|\mathcal{U}| \times |\mathcal{F}|$, Table 5.4, and is defined by the Equation 5.1 :

$$P(u,f) = \sum_{\forall R(u,e) > P_t} F(e,f) \tag{5.1}$$

where $u \in \mathcal{U}, e \in \mathcal{E}$. In our case, we define $P_t = 2$.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\mathcal{U}_1$	1	0.2	0.3	1.2	0
$\mathcal{U}_2$	1.2	0	1.1	0	0
$\mathcal{U}_3$	0	0.8	1	0.8	0.3

TABLE 5.4: Matrix P, user-feature from events

5.3 Weighted user-feature matrix

Actually, with the matrix P we know only the features related to an event rated by a user. We do not take account the rating of the user and this is the reason why we have to compute the feature frequency, user frequency and inverse user frequency matrix to find the features which better describe a user and those which better distinguish him from the others.

The feature frequency matrix FF is the total weight to which a user u has an interest, the features from the event and *Facebook*. Remember, a user u and a feature f are correlated if and only if $P(u, f) > P_t$. Those from *Facebook* are all taken into consideration. FF has a size of $|\mathcal{U}| \times |\mathcal{F}|$, Table 5.5, and is defined by the Equation 5.2 :

$$FF(u, f) = k1 \cdot P(u, f) + k2 \cdot S(u, f) \tag{5.2}$$

where $u \in \mathcal{U}, e \in \mathcal{E}$ and $k1, k2 \in \mathbb{R}^*$. $k1$ and $k2$ are factors to point out which feature weight is more important. In our case, $k1 = k2 = 1$.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\mathcal{U}_1$	1	0.6	0.3	2.0	0.3
$\mathcal{U}_2$	2.2	0	1.6	0	0
$\mathcal{U}_3$	0	1.0	1	0.8	0.9

TABLE 5.5: Matrix FF, feature frequency

The user frequency matrix UF is the number of users in which the feature f occurs at least once. It has a size of $1 \times |\mathcal{F}|$, Table 5.6.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\sum u$	2	2	3	2	2

TABLE 5.6: Matrix UF, user frequency

We can compute the inverse user frequency matrix UIF which has a size of $1 \times |\mathcal{F}|$, Table 5.7. It will define the feature regarding the number of users who has rated it. Higher is this number, lower is the importance. UIF is defined by the Equation 5.3 :

$$IUF(f) = \log_{10} \frac{|\mathcal{U}|}{UF(f)} \tag{5.3}$$

where $|\mathcal{U}|$ is the total number of user and $f \in \mathcal{F}$.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\sum u$	0.176	0.176	0.0	0.176	0.176

TABLE 5.7: Matrix IUF, inverse user frequency

Finally, we have all the information to calculate the weighted feature matrix W, which has a size of $|\mathcal{U}| \times |\mathcal{F}|$, Table 5.8. This will allow us to find the features which better describe a user u and those which better distinguish him from the others. W is defined by the Equation 5.4 :

$$W(u, f) = FF(u, f) * IUF(f) \tag{5.4}$$

where $u \in \mathcal{U}, f \in \mathcal{F}$.

	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$
$\mathcal{U}_1$	0.176	0.106	-	0.352	0.0528
$\mathcal{U}_2$	0.387	-	-	-	-
$\mathcal{U}_3$	-	0.176	-	0.141	0.158

TABLE 5.8: Matrix W, feature weight matrix

5.4 User similarity matrix

At this stage, only the content-based part has be done and now we have to determine the similarities among users (the collaborative filtering part). For this, we use the cosine-based approach which define the similarity between 2 users (u, v) and a feature $f \in \mathcal{X}$. The symmetric user-user matrix UU has a size of $|\mathcal{U}| \times |\mathcal{U}|$, Table 5.9, and is defined by the Equation 5.5 :

$$UU(u, v) = \frac{\sum\limits_{\forall f \in \mathcal{X}} W(u, f)W(v, f)}{\sqrt{\sum\limits_{\forall f \in \mathcal{X}} W(u, f)^2} \sqrt{\sum\limits_{\forall f \in \mathcal{X}} W(v, f)^2}}, \mathcal{X} = \mathcal{F}_u \cap \mathcal{F}_v \tag{5.5}$$

where $\mathcal{F}_u = \{f \in \mathcal{F} | P(u, f) > 0\}$, $\mathcal{F}_v = \{f \in \mathcal{F} | P(v, f) > 0\}$ and $u, v \in \mathcal{U}$.

The most similar neighbors of user u are those with the highest value. In the Table 5.9, it would be for $\mathcal{U}_1$ the user $\mathcal{U}_2$ then $\mathcal{U}_3$.

	$\mathcal{U}_1$	$\mathcal{U}_2$	$\mathcal{U}_3$
$\mathcal{U}_1$	-	1.0	0.823
$\mathcal{U}_2$	1.0	-	-
$\mathcal{U}_3$	0.823	-	-

TABLE 5.9: Matrix UU, user-user similarity matrix

5.5 k nearest neighbors

Now we have the similarity user-user matrix, the last step will be to find the most recommended products. First, we have to keep only the $k, k \in \mathbb{N}^*$ nearest groups of neighbors. We keep groups because in the case that $k = 1$ and $\mathcal{U}_1 = \mathcal{U}_2$, $\mathcal{U}_1$ and $\mathcal{U}_2$ have the same importance and we do not want to take only one of them. Those steps are decomposed in 6 stages. For the example we take $k = 2$ and look for a recommendation for $\mathcal{U}_2$.

1. We get the nearest groups of neighbors of $\mathcal{U}_2 : \{\mathcal{U}_1\}$;
2. We get the events in the neighborhood that $\mathcal{U}_2$ has not rated : $\{\mathcal{E}_4, \mathcal{E}_5\}$;
3. For each event, we get theirs features : $\mathcal{E}_4 : \{\mathcal{F}_2, \mathcal{F}_4\}, \mathcal{E}_5 : \{\mathcal{F}_4\}$;
4. We compute their frequency in the neighborhood : $fr(\mathcal{F}_4) = 2, fr(\mathcal{F}_2) = 1$;
5. For each item, we add its features frequency finding its weight in the neighborhood : $w(\mathcal{E}_4) = 3, w(\mathcal{E}_5) = 2$;
6. Finally we keep the $k2, k2 \in \mathbb{N}^*$ events with the highest score of step 5 which will be the recommended events. In our case $k2 = 1$, we keep only $\mathcal{E}_4$.

Chapter 6

Conception

In this section, the whole conception and corresponding technical details are described. First, we talk about the database and the DBMS used (database management system) and then about the architecture of the program. We continue with the class diagram and finally, we evoke the performances.

6.1 Database

The *DBMS* used in this project is *MongoDB*. It is a *NoSQL* database and has very good performances with big data. In our case, there is a high likelihood to have a lot of information for a recommendation system. *NoSQL* databases are more adapted for this kind of project than *SQL*, specially for performances and scalability.

The Figure 6.1 shows the relational model used. If we start off by the user, we can see he can have some associated features which come from *Facebook* and he has the possibility to rate an event. An event has a category and several features with a weight. Some entities have a field *external_id* which corresponds to the *id* of *Facebook* for the user or the *id* used in the website `http://www.gokera.ch`. The user also has a token from *Facebook*, given to the Graph API when we send queries.

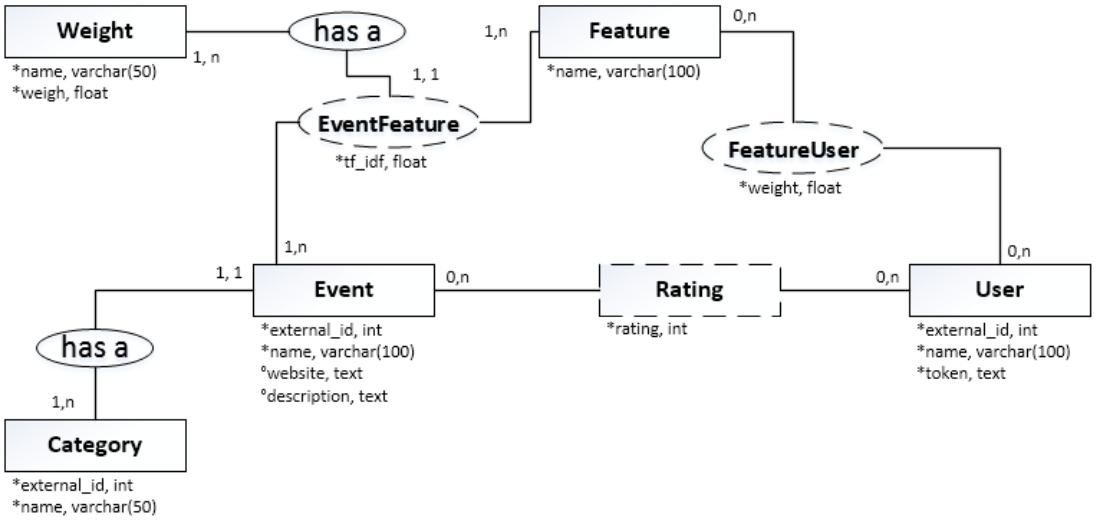

FIGURE 6.1: Relational model of the database

6.2 Architecture

The project is written in *Python 2.7* and uses the web framework *Django* 1.3. The final purpose of this project is to be used in a web environment so it is necessary to realize it in a website. The kind of the database used is *NoSQL* and the DBMS is *MongoDB* (more details in chapter 6.1).

Django is based on applications we insert into it. We have developed those which are :

1. Main, which displays the login page and the action page;
2. Event, which allows to add an event, show and rate it;
3. Events analysis, which contains all the part related to the event analysis : website parser, tf-idf, *Treetagger*. This part is multi-threaded;
4. FBGraph, which provides the *Facebook* login and the token management. Moreover it extracts the *Facebook* features for a user. A part is multi-threaded;
5. Recommendation, which is our recommendation system which takes into account the event analysis, *Facebook* features extraction.

All those applications always have the same structure : the views, models, functions and classes. Only the views are present in all the applications. The Figure 6.2 shows the architecture.

At the root of the project, there is the *App config* that contains all the specifics parameters, for example the depth of the website analysis, the weight of the features from the description etc. There are also the unit tests for the tf-idf and urls-extractor. *Settings, Manage, Tests* are files of *Django*.

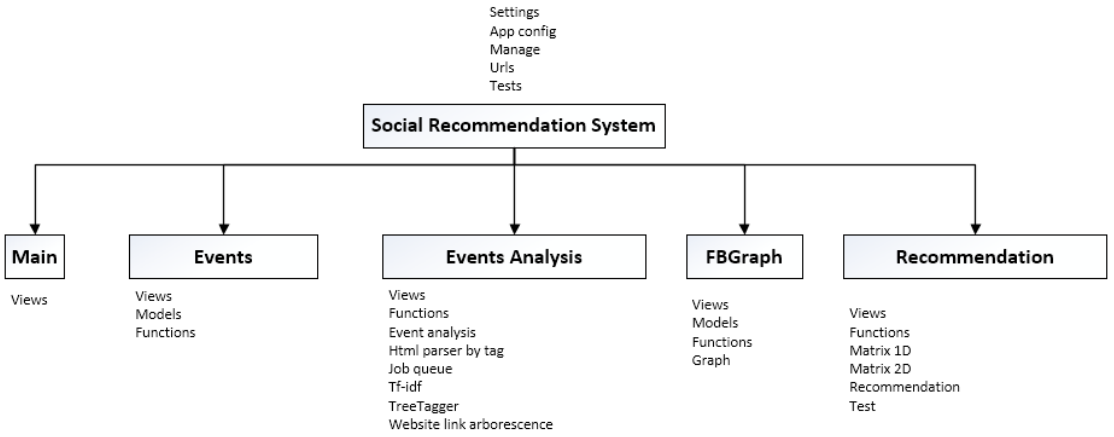

FIGURE 6.2: Architecture of the project

The event analysis is the biggest application. It contains the *TreeTagger* used for the text analysis. It is a tool that works for several languages which contains the French. This tools is very slow but it works with French. If the language was English, we would have used the library *NLTK* which is faster[1]. Another tool we use is for the natural language of a text. It is called *Guess Language* [2] and it can detect 60 languages. It uses heuristics based on the character set and trigrams in a sample text to detect the language. Longer the text is, more accurate is the detection. *HTML parser by tag* allows us to extract what we want in a website content by specifying tags. With this, we use also *Website link arborescence* which extracts all the "unique" urls that points at the same domain. Finally, we have a job queue we use for the multi-threaded part because the insertion of features inside the database is not thread-safe.

The *Facebook Graph* application uses some libraries which allow us to realize the login and the queries. For this, we use the libraries *PyFB*[3] and *FacePY*[4].

The Recommendation application uses the famous scientific computing library *NumPy*[5], the matrix classes and some mathematical operations. Those operations are also used in the event analysis. There is also a unit test in the file *Test*.

[1]More information on `http://www.nltk.org`.
[2]Available on `https://pypi.python.org/pypi/guess-language`.
[3]Available on `https://github.com/jmg/pyfb`.
[4]Available on `https://github.com/jgorset/facepy`.
[5]Available on `http://www.numpy.org/`.

6.3 Class diagram

The project is decomposed only in 4 packages, because the *main* application doesn't have any logic.

In Figure 6.4 shows the class diagram for the package *FBGraph* and *Events*. In the *Events* package there are only models from the database. Those classes are used everywhere. For the package *FBGraph* we have create the last model, *User*, and have defined a class for the posts, pages and groups from *Facebook*. There is also a *Graph* class that refers to the *FBGraph API*. The last class, *FBUser* allows us to represent a user with all his pages, groups and messages and we can compute its activity for the *Facebook* analysis (more details in chapter 4).

The event analysis conception is showed in Figure 6.5. It is decomposed in three parts : text analysis, website parsing and job queue. For the last one, it is just to allow to multi-thread the event analysis because the feature insertion into the database is not thread-safe. The website parsing has a class (*HTMLParserLink*) to get the "unique" links (more details in chapter 3.2.3) and uses the class *TreeNode* to represent a node in the url graph. *HTMLParserByTag* is another website parser that extracts the content of specific tags. The rest is dedicated to the text analysis and its comprehension is quite trivial.

The last package, the recommendation one, represented in Figure 6.3, has simply a class which uses personal classes *Matrix1D* and *Matrix2D* and which performs all the recommendation computations, described in chapter 5. Those classes have columns and rows represented by a list of object (*User, Event, Feature*) and they allow us to access to their value in a simplified way.

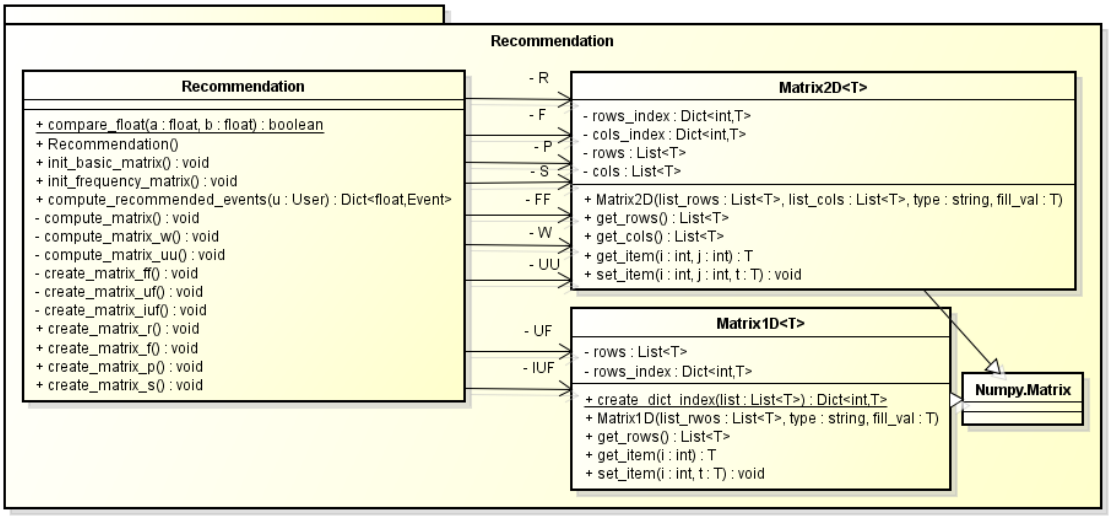

FIGURE 6.3: Class diagram of the package *Recommendation*

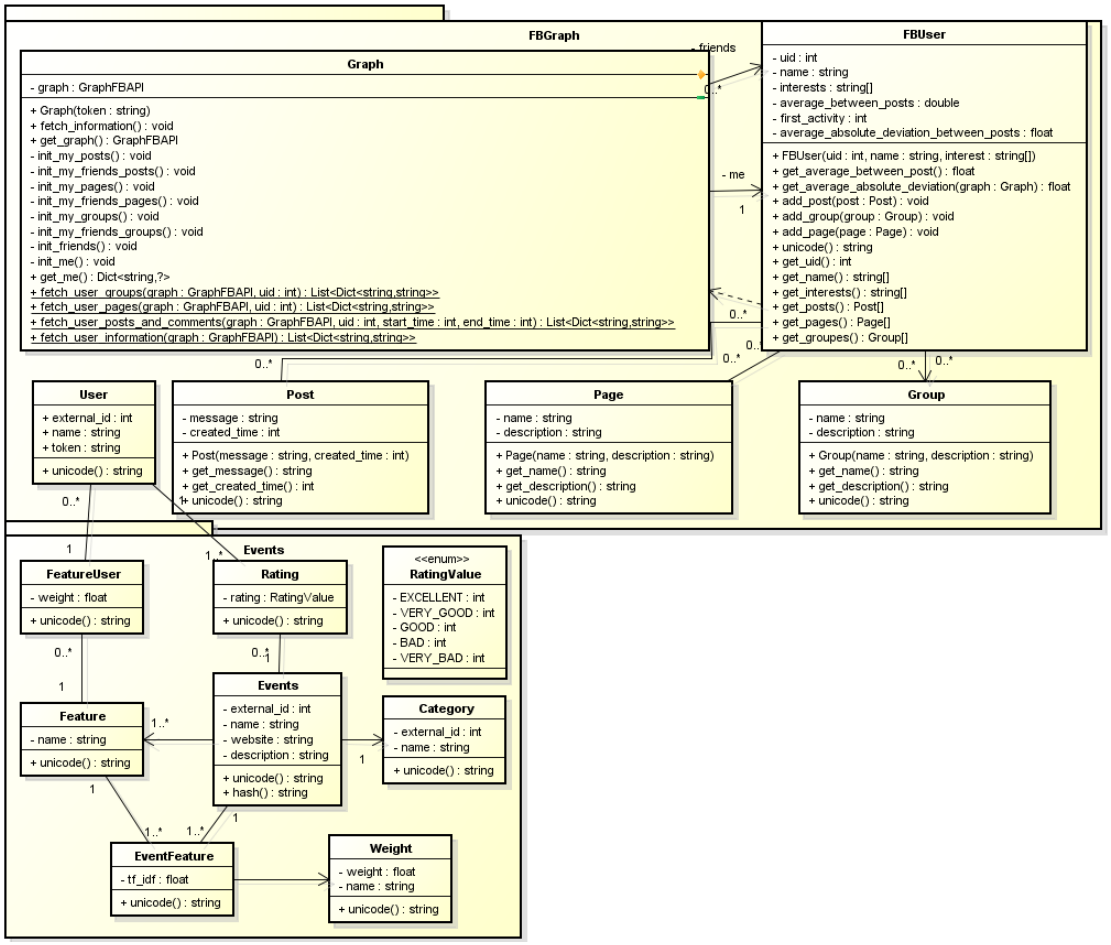

FIGURE 6.4: Class diagram of the package *FBGraph* and *Events*

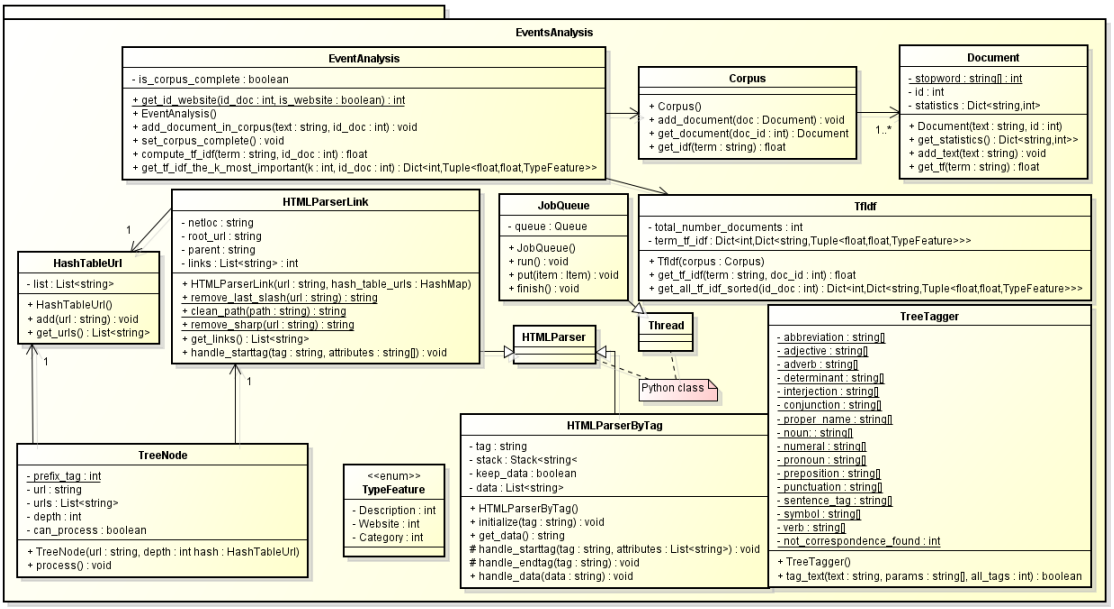

FIGURE 6.5: Class diagram of the package *EventsAnalysis*

6.4 Performance

In a general way, the application is very slow. Using a web framework is not the fastest way to display pages but this is not a big deal comparing to the next other performances.

The event analysis is decomposed in four main steps : to fulfill the corpus with description and website content, to compute the *(tf-idf)* of each term for each document, to find the k most important features and finally to insert them into the database. Step four is quite fast because it is simple database insertions and it is not possible to multi-thread it. Steps two and three are slow because there are a lot of data and we need to calculate several things for each term in a document. This part is multi-threaded and it is helpful.

The first step is a very long process. First, parsing a website with *Python* is quite slow and we look for all unique urls which point at the same domain for every website. As there are thousands of queries, this part is very slow. Moreover, we have to analyze the text of the description and the website content. As we treat French texts, we use *TreeTagger* and it is a very slow tool. To just analyze a simple sentence of several words it needs approximately one second. For longer sentences, this time is longer and if you parse the content of a website even much longer.

This part is also multi-thread but there could be several ways to do it : to split the events in n part where $n = \sharp Threads$, or for each website analysis, start n threads to parse all the urls but you have to take into account the commutation of context. For the first, if unfortunately you have one thread which has all the websites with the longest contents, the others will wait on it and so will waste time. Actually we use the first way but it could be imaginable to use thread pool and try the second way which should maybe be faster.

The machine we used was a *Windows 8.0* with a processor E8400 which has two cores at 3.0 Ghz, 4 GB of RAM. If we use only one core, the event analysis will take approximately thirty hours and using the both cores, approximately twenty-five hours. We can imagine that maybe with a computer having sixteen cores, we could improve the time to ten hours.

The *Facebook* analysis is also very slow for the same reasons as the event analysis. It uses *TreeTagger* to extract feature from the groups and pages. The number of texts to analyze is quite large because if we have two hundred friends, and each of them likes three hundred groups and pages, *TreeTagger* will have a lot of work. Of course, this part is multi-threaded but there is still one problem with the activity computation : for each user we have to quantify the number of messages posted per month and for this, we use one query. *Facebook Graph API* fixes a limitation at six hundreds calls per ten minutes and this is a problem. To avoid this inconvenient, between each query we wait one second but there is a clever way to experiment : we can use multi-queries as we use for fetching information and we can send maximum sixty queries inside one and it is considered like one. So with this technique we can realize thirty-six thousand queries

in ten minutes but it needs a more complex management because we have to fulfill the multi-queries every time.

We test this part with the first way and we have approximately thirteen hours and thirty minutes for one core and for both cores eight hours. But it was only for one user. Now let's imagine we have to analyze the *Facebook* for each user who connects to the website for the first time, it will not be possible with this way.

For the recommendation, the execute time is around five minutes if we use only one core. The reason is that we have approximately 65′000 links between events and features. When we iterate over all those objects, for each of them we have to query the database to know the associated weight. The rest is quite fast because it is simply multiplications with values of big matrices. Instinctively, we can think that multi-thread it would be faster, but for some reasons we don't know, the execution time is multiply by a factor of two or four depending the number of threads. We didn't get time to find the reason of this behavior but we made a hypothesis : *Django* queries work with lazy initialization and make a request each time we want to access to an object, so it could be possible that accessing to the database is mono-thread.

Chapter 7

Problems encountered

At the beginning of the project, we have defined that the website and the entire project will be realized in *C++*, especially for performance and ability reasons. The web framework used was *Wt*[1] based on the famous library *Boost*[2]. *Wt* is based on a set of components that could be easily integrated into a web page, as well as on a signals systems.

The encountered problem was the connexion to *Facebook*, based on an *OAuth* system. Some exchanges must be established via requests using *GET* in *HTTPS*. Unfortunately, during the project, it was impossible to get responses in *HTTPS* whose their body wasn't empty and their status different than -1. Several tentatives were carried out during the compilation of *Wt* in order to find a solution :

- To use different versions of *Boost* : 1.53 and 1.54, in 32 and 64 bits;
- To recover the last source codes of *Wt* on the official *github*[3];
- To combine different optional libraries of *Wt* with *OpenSSL*;
- To compile with different versions of the *Microsoft* compiler, especially version 2010, 2012 and 2013.

Unfortunately none of those solutions didn't allow us to solve the problem. After one week of hard work, the language *C++* and the library *Wt* were dropped out and it is the language *python* as well as the web framework *Django*[4] which was chosen.

After our last decision, I could discuss with the developer of *Wt*, *Koen Deforche*, who explained to me that the problem came probably from *OpenSSL*, especially the folder that contains the certificates as its location is not standardized and it is the operating system which decides where to place it.

[1]For more information, you can consult the following website `http://www.webtoolkit.eu/wt`.
[2]Set of *C++* libraries intend to cover large domains. More information on `http://www.boost.org`.
[3]Available on `http://wwww.github.com/kdeforche/wt`.
[4]More information on `https://www.djangoproject.com`.

When we enter the command in a shell

"*openssl version -d*"

the output shows the location of the folder

"/usr/local/ssl"

which doesn't exist in *Windows*. A solution would be to compile *OpenSSL* and change the parameter *openssldir*. Unfortunately, this solution hasn't been tested because we didn't have enough time.

Chapter 8

Planning

In this part, we describe the initial planing we have establish at the beginning of the project and the final one. Then we describe the differences between those and make findings to take into account for future projects.

8.1 Initial planning

The Figure 8.1 shows the initial planning at the beginning of the project. We have decomposed the project in five iterations : the analysis, features and database, *Facebook*, social recommendation system and documentation.

The analysis is one of the longest iteration because we have to conceptualize the project, what we need and what we can do, still in *C++*. Then there is the second iteration that contains the whole events analysis with the storage. In the middle of the project, there is the *Facebook* part that is not supposed to be long to realize it and then, the recommendation part that is supposed to be the longest one. Finally, there is a last iteration for the documentation.

8.2 Final planning and findings

The Figure 8.2 shows the final planning. The first iteration was nearly the same, except that there was more time for the report. But at the end of this iteration, we had a problem and we changed the language we used. All the encountered problems are described in the chapter 7. So we have lost some time to realize the website in *Python* but we have respected the end of the iteration.

The second iteration has lasted longer than we thought because we didn't estimate correctly the duration of the tests and during this period, we had access to a server with sixteen cores only during a short time. It was very important to test the system in order to run the events analysis on it to have some data to test the recommendation system. Unfortunately we still had some problems but we couldn't focus only on it and so have

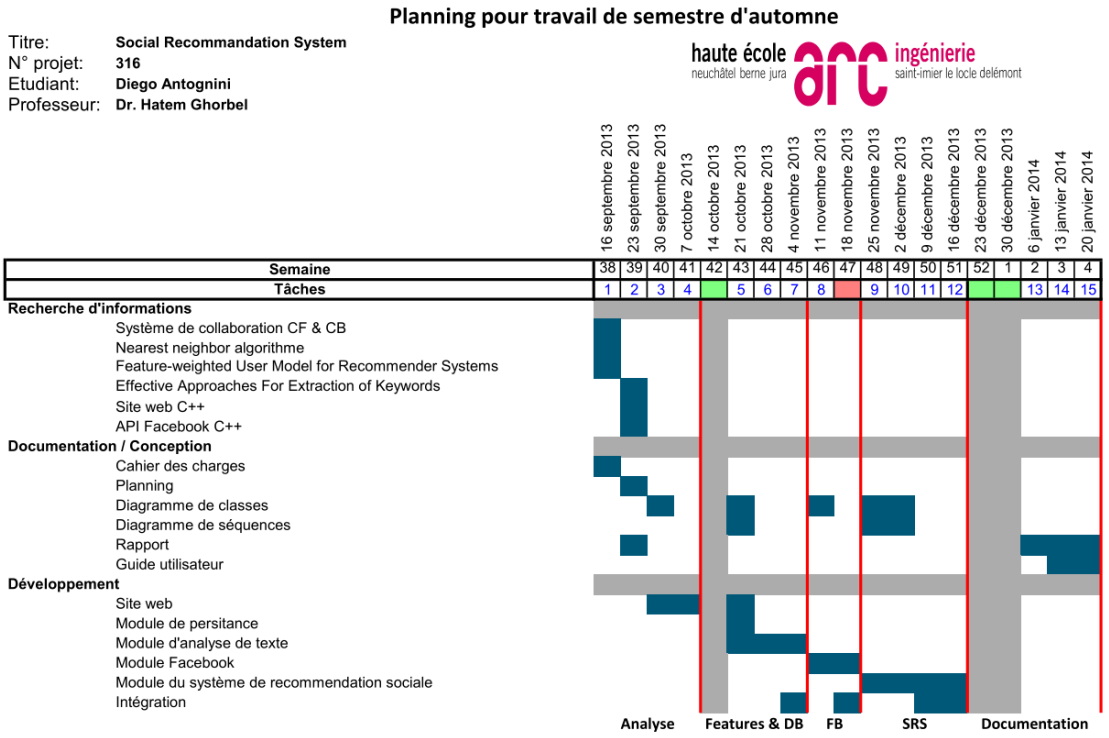

FIGURE 8.1: Initial planning

started the third iteration which was as we thought.

The social recommendation system iteration was shorter than expected because it was simpler than we thought. We have profited of this time to run our scripts on a computer to have some data to test the entire system. The last of those scripts are quite long, more details in chapter 6.4.

The last iteration regarding the documentation was shorter because the report had been written during all the project, in order to gain time at the end of the project to have time in the case where we had some major issues.

Finally, the sequences diagram has not been realized because we didn't think that was useful.

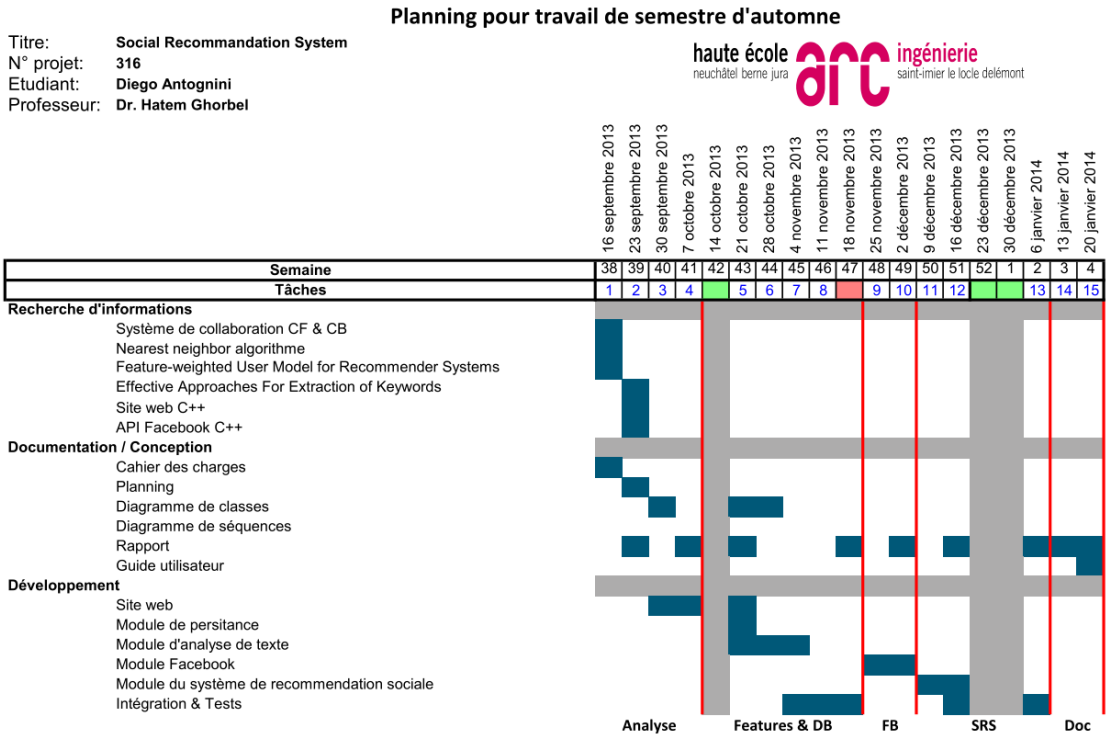

FIGURE 8.2: Final planning

Chapter 9

Discussions

This section contains the different results we have obtained and we discuss about them. We give also some leads to improve those whether it be about their accuracy or about the time to compute it. We start with the event analysis, then continue with the *Facebook* analysis and finish with our social recommendation system.

9.1 Event analysis

The most important thing when we analyze events, is to have a maximum of information about each of them. In our case, we interested in category, description and website. Among the 3140 events, we got already $25,41\%$ of them with an empty description and no website. This means that none of them will be recommended. $66,91\%$ of the events have a description detected in French and $51,97\%$ have a website with French content. Finally $27,68\%$ of the events have a description and a website with French content. If we remove the empty events and adapt the statistics, we have respectively : $89,7\%$, $76,35\%$ and $37,11\%$. Moreover, we have an average of $367,68$ key words per event. Those last results are quite satisfactory but higher will be these statistics, more accurate will be the event analysis.

In a general way, analyzing French texts with *TreeTagger* is very slow, approximately one second for a short sentence. This is very problematic because we use it for the description and website analysis and their content could be very long, especially for the website. Unfortunately, *Python* has only a library for English texts analysis. So, it could be worthwhile to find another way to analyze French text. As the event analysis is not an operation realized each day, we can imagine use it for this part on a machine with a high number of cores. If we really want to be efficient, rather running each time the entire event analysis, it could be interesting to implement a system of re-indexing when we add a new event.

Currently, the parts of the sentence in which we interest in are the nouns and the proper names. But sometimes, we can miss some important words which could be an adjective or another kind of word, or we can have some noise when we detect a proper

name. To improve the quality of the features extraction, it could be interesting to analyze deeply the semantic of a sentence but maybe it would take so much time. Another extra way would be to use an additional system with synonyms and group the features which mean the same thing. At this moment, the noise of the feature extraction is quite high and it is really necessary to find a better way.

For the websites, to detect the uniqueness of a page for our hash table, we use the url as a key and we can have different urls for the same page. So we have only a little redundancy because those pages, accessible by different links, are not very common.
An alternative would be to use the hash of the page content as key but it could be more accurate and maybe we can gain some time because we avoid some text analysis. But the content of a page could be different depending on some dynamic elements inside, like the time, some advertisements, etc. Moreover, we have to fetch the entire content of each page and so, loose some time. However we could mix those two approaches and have only the problem of some dynamic contents but those are quite rare. For the time, it depends on the language but in *Python*, to request a web page is quite slow.

During the description analysis, if we have more than k key words we do not parse the website. We use this technique to highlight the description but we have a side effect if we have exactly $k - 1$ key words and add a large content from the website. To avoid this, we could define a function which defines the number of key words to take in the website according to the number of key words in the description.

9.2 Facebook analysis

In a general way, we have an acceptable correlation between the features extracted from *Facebook* and those from the event analysis. In fact, $29,08\%$ of all the features have been found in a *Facebook* profile. Among these, we have to take into account the features we have extracted which are insignificant. We have the exact same problem that the event analysis in term of noise for the features and speed so the solutions would be the same. The execution time for this part is very important because this analysis is supposed to be realized in a few seconds.

Our area of information research in *Facebook* is focused on the pages and groups liked by the user. It could be interesting to extend it by the *Facebook* events in which the user has participated or his activities, books, messages for example.

In any case, whether the user is active or not, we collect data on his friends and define a weight for the feature depending their activity in relation to the k most active friends. The best way would be to represent the events of *Gokera* on *Facebook* and see which of them his friends have participated and use a system similar to a recommendation based on collaborative filtering to define the weight. Another way which will decrease the execution time is rather to compute the activity of all his friends, we take only friends who have already participated at an event.

9.3 Social recommendation system

Our social recommendation system is slowed down only by the creation of the matrix event-feature because in our case we have $65'000$ links to query each time we want to compute the recommendation and because of this, it cannot be computed in few seconds. The solution would be to store and initialize the basic matrices in the program when we start the server and update them several times per day. This will allow to recommend events for a user in seconds.

Chapter 10

Conclusion

Through this project, we have created an entire recommendation system focused on a single user, exploitable directly in a website. We have realized each part of it, especially the event analysis, the analysis of the *Facebook* profile of a the current user and a hybrid recommendation based on the feature-weighted approach mixing the collaborative filtering and the content-based.

The events we use (3140), from the website *Gokera*, have enough information to work with and if we take into consideration the description and/or the website of each event, we reach an average of $367,86$ key words per event. In term of performances, the event analysis is too slow mainly because we analyze French content with an external tool which is slow. Moreover parsing a website is not fast neither. For the efficiency, we can say that we have the most important key words but we have also a lot of noise due to simple features extraction.

With the *Facebook* profile of the user we have tested, we found $29,08\%$ of correlated features and we have to take into account the features which are insignificant (due to the noise). But even if the user is inactive, we have found a way to find features from his friends data. Actually, we can not say if this system is relevant because we could not try it with some other users, mainly due to the execution time, for the same reason than the event analysis.

Finally, our recommendation system we have realized, seems to fill the problem in theory, if a user is new or does not have a history of ratings. In practice, we cannot say if the final system is relevant because we do not have a set of data in which we can remove values and try to find them. For this, we will participate at the *ESWC-14 Challenge: Linked Open Data-enabled Recommender Systems*[1] where we would be able to make some conclusions about the pertinence of our system. In term of performance, it would be capable to compute the recommended events in several seconds.

[1] More information on `http://2014.eswc-conferences.org/important-dates/call-RecSys`.

Appendix A

Product requirements document

Filière informatique

haute école arc ingénierie
neuchâtel berne jura saint-imier le locle delémont

Cahier des charges pour travail de semestre d'automne

Titre:	**Social Recommandation System**
N° projet:	**316**
Etudiant:	**Diego Antognini**
Professeur:	**Hatem Ghorbel**

Situation initiale

Les systèmes de recommandations sont utilisés dans la plupart des grands sites web, autant au niveau consommation qu'au niveau social. Les plus connus sont notamment *Amazon*, *eBay*, *Last.fm* pour la première catégorie puis *Facebook*, *Google +*, *Twitter* pour la seconde.

Le point faible de ces systèmes est qu'un utilisateur n'ayant encore rien visité/consommé verra que les produits recommandés sont peu ciblés. L'idée est de rajouter une nouvelle dimension sociale et de récupérer tout à ce quoi l'utilisateur s'intéresse. De plus, le système à réaliser sera principalement pour des évènements, étant sociaux, donc cette nouvelle dimension devient intéressante.

Actuellement, les systèmes de recommandations par groupes sont rares et peuvent être intéressant pour un ensemble de personne souhaitant aller à un événement qui convient à tout le monde.

Buts du projet:

Le principal but du projet est de réaliser un système de recommandation sociale hybride (collaboratif et basé sur le contenu) pour les événements recensés sur le site *Gokera*[1]. A partir du profile *Facebook* et de ses appréciations des différents événements proposés (*I like & I don't like*), le système sera capable de recommander ceux étant le plus compatible avec l'utilisateur. Le même système devra pouvoir fonctionner avec un groupe d'utilisateur.

Un prototype de site web sera réalisé afin de mettre en place le système.

Démarche proposée:

La démarche proposée est la suivante :

1. Se documenter sur le fonctionnement des systèmes de recommandations *CF*[2] et *CB*[3] ;
2. Lire et comprendre l'algorithme proposé dans l'article scientifique[4] ;
3. Réaliser un prototype de site web en C++ ;
4. Extraire les *features* (caractéristiques thématiques) lors de l'insertion d'événements ;
5. Récupérer les informations nécessaires *Facebook* de l'utilisateur ;

[1] Voir le site www.gokera.com
[2] *Collaborative Filtering*
[3] *Content-based*
[4] L'article se nomme : *Feature-weighted User Model for Recommender Systems*

6. Remplir les différentes matrices selon les points 4 et 5 ;
7. Implémenter le système de recommandation hybride du point 2.
8. Implémenter le système de recommandation par groupe (optionnel)

Contraintes:

Pour des questions de performance, il serait bien de réaliser le système en C++.

Les directives de travail sont détaillées dans le document annexe.

Neuchâtel, le mardi 01 octobre 2013 / TVO

Les parties ci-dessous déclarent accepter le contenu du présent cahier des charges.

Un exemplaire est remis à chaque partie.

Etudiant :	Professeur:	Mandant:
.....	.....	.....

Distribution Mandant
Etudiant concerné
Filière informatique
Expert

Bibliography

[1] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, pages 331–370, 2002. URL `http://dl.acm.org/citation.cfm?id=586352`.

[2] Jeffrey David Ulman and Anand Rajaraman. Recommendation systems. *Proceeding UM '07 Proceedings of the 11th international conference on User Modeling*, pages 277–309, 2011. URL `http://infolab.stanford.edu/~ullman/mmds/ch9.pdf`.

[3] Alexandros Nanopoulos Panagiotis Symeonidis and Yannis Manolopoulos. Feature-weighted user model for recommender systems. *Proceeding UM '07 Proceedings of the 11th international conference on User Modeling*, (12):97–106, 2007. URL `http://dl.acm.org/citation.cfm?id=1419416.1419433`.

[4] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994. URL `http://odur.let.rug.nl/~vannoord/TextCat/textcat.pdf`.

[5] Jasmeen Kaur and Vishal Gupta. Effective approaches for extraction of keywords. *IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 6*, pages 144–148, 2010. URL `http://ijcsi.org/papers/7-6-144-148.pdf`.