

Warsaw University of Technology
The Faculty of Electronics and Information Technology
Institute of Control and Computation Engineering

FACEBOOK USER INTERESTS
EXPLORATION AND RECOMMENDATION
BASED ON FACEBOOK SOCIAL GRAPH
DATA ANALYSIS

Author:

JAKUB KRZEMIEŃ
STUDENT NO. 214616

Supervisor:

Mariusz Kamola PhD

Warsaw, 2014

Streszczenie

Eksploatacja i sugerowanie zainteresowań oparte o analizę danych z Facebook Social Graph

Dynamiczny rozwój i rozpowszechnienie Internetu a także ogromna popularyzacja serwisów społecznościowych sprawiły, że ogromne ilości informacji na temat użytkowników Internetu są dostępne na wyciągnięcie ręki. Najpopularniejsze serwisy społecznościowe udostępniają wygodne narzędzia dzięki którym można wchodzić w interakcję z użytkownikami tych serwisów, a także uzyskiwać dostęp do danych profilowych. Takie informacje jeszcze kilka lat temu były niedostępne czy też nieosiągalne. W ostatnich czasach sytuacja ta uległa zmianie co generuje nowe, niezbadane jeszcze obszary związane z analizą i interpretacją tego typu danych. Również w związku z rozwojem Internetu i ogromem informacji w nim zawartych, stało się pożądane by w jakiś sposób moderować czy selekcjonować treści dla użytkownika. Mechanizmy serwujące reklamy w serwisach WWW starają się prezentować treści, które mogą zainteresować danego użytkownika w oparciu o historię jego zapytań czy wcześniej przeglądane strony. Serwisy społecznościowe jak np. Facebook filtrują prezentowane dane, starając się pokazywać informacje jedynie od najbliższych czy najważniejszych znajomych. Sklepy internetowe próbują sprzedać dodatkowe produkty dopasowując je do poprzednich zakupów danego użytkownika. Wcześniej wspomniane nowe, niezbadane jeszcze dane stanowią ciekawą alternatywę jako podstawa działania dla tego typu mechanizmów.

Celem niniejszej pracy jest analiza obecnie znanych i używanych mechanizmów rekomendowania treści, a następnie próba stworzenia nowego mechanizmu w oparciu o dane z sieci społecznościowej, w tym wypadku Facebooka. Praca skupia się na sugerowaniu muzyków i zespołów muzycznych w oparciu o zainteresowania użytkowników.

Ze względu na subiektywną naturę tego typu rozwiązania w ramach pracy powstały także narzędzia mające na celu weryfikację otrzymanych wyników.

Abstract

Because of the dynamic evolution and propagation of the Internet and the popularization of social networks, huge amounts of information about the users of the Internet are easily accessible to anyone. The most popular social networks provide efficient and easy to use tools that allow to interact with their users, as well as access profile data. This type of information just several years ago was not available or inaccessible. This change generates new, unexplored areas for analysing and interpreting such data. Also because of the evolution and popularization of the Internet and the vast amount of data stored, it has become desirable to moderate and select the content that is being displayed to the user. Mechanisms presenting ads on websites try to present content that can interest the user based on his browsing history or previous queries. Social networks such as e.g. Facebook filter the presented information and try to show only the data related to the closest or most important friends of the user. E-commerce sites try to sell additional goods by suggesting products based on previous purchases made by the user. The aforementioned new, unexplored data is an interesting alternative for the core of such systems.

The aim of this paper is to analyze the currently known and used content suggestion mechanisms and then create a new one based on data from a social network, in this case Facebook. In particular the paper focuses on suggesting music bands and musicians based on user interests.

Because of the subjective nature of such a solution, as a part of this paper, verification tools have also been developed.

Contents

1	Introduction	5
2	Basic information	7
2.1	Introduction to the concept of a social network	7
2.2	Facebook	8
2.2.1	Facebook Social Graph and Open Graph concepts	8
2.2.2	Facebook Platform, Graph API and Facebook Query Language (FQL)	8
3	Currently used recommendation algorithms	10
3.1	Collaborative Filtering - introduction	11
3.1.1	Traditional Collaborative Filtering	12
3.1.2	Cluster Models	13
3.1.3	Search-based filtering	13
3.1.4	Slope One algorithm	14
3.2	Commercially used recommendation algorithms	15
3.2.1	Amazon - Item-To-Item Collaborative Filtering	15
3.2.2	Last.fm - Label Propagation Algorithm	16
4	Social-based recommendation algorithm concept	21
4.1	Data acquisition	21
4.2	User data analysis	25
4.3	Building the graph	34
4.3.1	Basic implementation	34
4.3.2	Refined implementation	35
4.4	Initial results	36
5	Cloudera Oryx	38
5.1	Oryx description	38
5.2	Oryx architecture	38
5.3	Oryx Collaborative Filtering algorithm	40
5.4	PMML documents	41
5.5	Apache Mahout	41
5.6	Oryx initial test	42
5.7	FB Graph data with Oryx	44
6	Results verification	47
6.1	Verification with Amazon recommendations	47
6.2	Verification with Last.fm recommendations	48
6.3	Amazon and Last.fm results comparison	49

6.4	Verification results	50
6.5	Conclusion	52
7	Summary	53
8	Bibliography	54

1 Introduction

Just several years ago the biggest online platforms in the world would protect the valuable information they collect. However, in time a more open approach was born, with public-facing APIs becoming open to the world and specially prepared SDKs being developed to help interact with those endpoints. One of the leaders of such a modern, open approach was Facebook. Very quickly Facebook applications became a natural element of the social network ecosystem, with friends inviting other friends to online games or social applications where users could interact with their friends content such as images or events in a new way. A new business was born with social media agencies creating specialized campaigns whose goal was to become viral and spread naturally like a virus. As a result, developers from all over the world gained access to unprecedented amounts of new data - user profiles, their interests, events they are attending, things they like, content they share and much more. Data gathered from social networks can be utilized in various ways, one of which is analyzed and studied in this thesis.

Recommendation Systems become more and more necessary for a variety of reasons - the amount of content Internet users are experiencing is growing rapidly and it has to be selected for them because the volume of information is simply overwhelming. For the same reason the attention span of users is shrinking, especially on mobile devices, and the content presented has to be interesting to the user, otherwise it will be disregarded. E-commerce platforms are very competitive and need to search for new ways to sell more products. Social networks users are very active, creating too much data, so it needs to be filtered. Popular cloud-based platforms for music and video try to interest users in new musicians or movies. Mobile and web ads systems need to present ads that can be interesting for the user, otherwise the click-through rate will be very low. These several examples showcase a wide variety of applications for Recommendation Systems. However, these Systems work based on algorithms and mechanics that have been developed before the age of Web 2.0 and the aforementioned freedom in access to social network data.

The aim of this paper is to analyze the current Recommendation Systems and algorithms and to create a new one based on data from a social network, in this case Facebook. In particular the paper focuses on suggesting music bands and musicians. Because suggesting content is a highly suggestive process, additional verification mechanisms were proposed and introduced. Additionally, a popular open source Recommendation System, Oryx, is used to compare results.

The paper is divided into several chapters:

- Chapter 2 - Basic information about social networks and Facebook
- Chapter 3 - An analysis of currently used Recommendation Systems
- Chapter 4 - Detailed information about the developed Recommendation System
- Chapter 5 - An analysis of a popular open source Recommendation System, Oryx
- Chapter 6 - A description of proposed verification mechanisms, and the results of this verification

2 Basic information

The success of social networks, understood as online platforms, is a phenomenon that created new sources of unprecedented quantities of social data. This allows for development of new concepts, one of which being the topic of this thesis, that could have not been possible just a few years ago. Internet turns out to be the perfect environment for a concept originating from social behavior and network science.

2.1 Introduction to the concept of a social network

A social network is a theoretical structure emanating from sociology, psychology and statistics. The most basic description defines a set of actors and ties between these actors. 'Actors' do not have to necessarily be individuals - in a social network this term is also applied to groups, organizations and whole societies.

The origins of social networking understood as an online platform which is being used to create virtual social networks can be dated back to the early 1970s [1]. From digital bulletin boards the idea evolved and made its appearance in the World Wide Web during mid 1990s. The basic concept back then was to bring people together to interact, for example via simple publishing tools, predecessors of blogs, or through chat rooms. In the next couple of years the focus shifted to user profiles, where people listed their personal information and could look for users with common interests. The first online platform with such functionality was SixDegrees.com, named after the concept of six degrees of separation - the theory suggesting that any two people can be connected through a chain of at most five acquaintances. With the evolution and propagation of Internet, more social networking sites emerged and quickly gained popularity, e.g. Friendster.com , MySpace.com, LinkedIn.com or Facebook.com.

2.2 Facebook

In the winter of 2004 TheFacebook.com was launched from the dorm room of a Harvard sophomore, Mark Zuckerberg [2]. Its initial focus was similar to other social networks, with the addition to exclusivity - users were only able to register when using a harvard.edu e-mail domain, restricting its initial user base to students of the Harvard College. From that moment Facebook grew to over 1 billion users, as of October 2012 [4].

Many factors contributed to this astounding user growth. A few of additions to Facebook were especially important for the purposes of this thesis:

- Facebook Social Graph and Open Graph - features allowing to connect users to entities existing both inside and outside of the Facebook universum,
- Facebook Platform, Graph API and Facebook Query Language (FQL) - technological features allowing developers to interact with objects within the graph of Facebook connections.

2.2.1 Facebook Social Graph and Open Graph concepts

Social Graph lies in the core of Facebook - described as "the global mapping of everybody and how they're related" it's a representation of people and their connections. These connections reach far beyond Facebook - Open Graph allows any developer to build on top off the social graph and define new social graph objects, e.g. users can log-in to third party websites using their Facebook credentials or interact, for example 'like', with virtually any object created by the developer.

These features, being one of the reasons for Facebooks popularity, are the source of information for this thesis, which focuses on users interests, described as connections with objects from the Social Graph.

2.2.2 Facebook Platform, Graph API and Facebook Query Language (FQL)

Another reason for the phenomenon of Facebook is its Platform, which allows developers to create their own services and applications that can interact with Facebook and access data provided by Facebook. Two common ways of interacting with Facebook data are available:

- Graph API - the basic element of the Facebook Platform. This API allows to read and write data to and from Facebook. It gives a consistent

view of the Social Graph with a clear representation of all objects and connections within the Graph. Every object is represented by its ID and a set of attributes. Each of these objects has also a set of relationships, or connections, to other objects, also accessible through their IDs (e.g. a user is connected to his photo albums). Connection types vary for object types, for example users can be connected to their friends and an event can be connected with its attendees. Accessing Graph objects is done with HTTP GET requests and all responses are JSON objects. Facebook SDKs, although their use is not required, allow for easy integration. Official SDKs have been released for JavaScript, PHP, iOS and Android. Countless third party SDKs are also available, ranging from Flash to Ruby.

- Facebook Query Language (FQL) is a SQL-style interface that enables developers to query the Graph API, with features designed specifically for larger data-sets, e.g. multi-queries. Queries are performed with HTTP GET requests and all responses are JSON objects. Despite features like subqueries, typical ORDER BY and LIMIT clauses etc. FQL is rather basic and lacks many features typical for SQL, for example a FROM clause can contain only one table. Response times for large data-sets are much better than when using Graph API though still easily reach 30 seconds or more, making it important to optimize data fetching and analysis logic.

While the Graph API represents data in the form of a network, FQL accesses data logically described as tables, which allows for two different approaches - depending on the data required, it is more natural to use one or the other. Also, some data is only accessible via the Graph API while other only via FQL. Finally, response times vary, depending on the amount and type of data being fetched from the API. That is why for the purposes of this thesis both options have been used.

3 Currently used recommendation algorithms

The task of any Recommender System is to try to rank given elements with a rating which measures the potential usefulness of those elements for the user, producing a rating as similar as possible to the rating that would be given to those elements by that user - for elements that haven't been yet considered by that user. In other words such systems try to predict and suggest items that the user could be interested in buying, music that the user could be interested in listening to etc.

The need for Recommendation Systems came from noticing the fact that users tend to rely on recommendations provided by others when making daily, routine decisions - for example, people read movie reviews prior to selecting a movie to watch. In the mid-90's, when e-commerce started developing rapidly, such systems became especially necessary. In recent years because of the growth of the Internet and the amount of choice an Internet user has, such systems prove to be particularly useful. Hugely popular Internet services such as Amazon, YouTube, Yahoo, NetFlix, Last.fm, IMDb, among others, rely heavily on Recommender Systems for a variety of reasons - a Recommender System can play many roles for the service provider.

The most popular functions of a Recommender System are[5]:

- Increasing the number of items sold - both for commercial and non-commercial services this is the most typical use-case. For non-commercial services this can be e.g. the number of articles a user reads.
- Selling more diverse items - Recommender Systems allow services to popularize less known items.
- Increasing the user satisfaction - relevant recommendations can increase the positive perception of the service.
- Increasing the user loyalty - Recommender Systems gather information about the users behavior over time and with more information the results can be more precise. This means that the longer a user interacts with the system, the bigger the likelihood that the results of the Recommender System will be meaningful, providing the user with valuable information, which will result in a better returning users rate.
- Better understanding of users needs - as mentioned previously, Recommender Systems gather information about user behavior. This infor-

mation is a valuable asset that can be used by the service provider for a variety of purposes.

It is important to remember that Recommender Systems often operate in a highly challenging environment:

- In the beginning, when new users or customers are created, the knowledge about those entities is very limited. Sometimes the Recommender System needs to produce results based on just a few ratings or purchases.
- On the other hand users or customers that have been actively using the product for a long time have generated a lot of data, possibly hundreds or thousands of ratings or purchases.
- The input data is constantly changing with every user action. The new data should be immediately taken into account by the Recommender System.
- The data-set which needs to be analyzed by the Recommender System is colossal, for example a renowned retailer can have millions of customers and even more unique items on sale.
- Typically the results have to be of very high-quality, otherwise the sole purpose of a working Recommender System is not met. On the other hand, those results need to be returned in realtime to ensure the user experience is flawless.

3.1 Collaborative Filtering - introduction

The motivation for collaborative filtering comes from the idea that people often get the best recommendation from someone with similar taste. Collaborative filtering explores techniques for matching people with similar interests and making recommendations on this basis.

That is one of the reasons why, as mentioned previously, typical applications of collaborative filtering often involve very large data sets. Collaborative filtering has been successfully used with various data-sets and various use-cases: e-commerce, finance, search engines, internet ad distribution networks, loyalty platforms and in other web applications that strive to deliver a personalized experience to the end user.

To ensure that the provided examples operate within the same context, all

are described from the perspective of a successful e-commerce platform - a platform that is used by a very large amount of users and contains a huge set of items available for purchase. Because of the volatile nature of online shopping it is also important the recommendations are fetched instantly but at the same time their quality must be high.

3.1.1 Traditional Collaborative Filtering

As described by the pioneers in this field, recommendation researchers and team leaders in Amazon.com [6], a traditional collaborative filtering algorithm perceive a customer as an N -dimensional vector of items, where N is the number of all items available. The vectors components can be positive or negative - if an item was purchased or rated positively, the component is positive. For negative ratings the component is negative. Because some items can be very popular, such best-sellers are taken into account by dividing the appropriate vector component by the amount of users who have purchased or rated the item. Most users vector is very sparse since not many items have been rated or purchased, which helps with the computation speed of the algorithm. Recommendations are found by selecting several users that are the most similar to the user requesting recommendations - that similarity is typically measured by the cosine of the angle between two vectors of the users that are being compared, although other techniques can also be applied. Once the similar users are found, the algorithm needs to select the appropriate items to recommend - usually simply the items most popular among the group of similar users are suggested.

One of the bigger issues with such an approach is the inflicted computational expense - in theory in the worst case the algorithm needs to check all users and all items so the processing expense would be $O(MN)$ where M is the number of all users and N is the number of items available. As mentioned previously, the vector for most users is very sparse since most users rate and purchase few items. That is why examining every user can be expected to actually cost approximately $O(M)$. However, a very limited group of users can be considered 'power users' who purchased and rated a big chunk of the items available, so the final cost can be expressed as $O(M + N)$. For a large data-set that means a heavy performance penalty, as well as limited scalability. To enhance the speed and responsiveness, the data-set needs to be reduced. M can be reduced by selecting users at random or by not taking into account users with few ratings and purchases. N can be reduced by discarding best-sellers or unpopular items. The items that need to be examined can also be narrowed down for example by the category of the item.

Each attempt to reduce the data-set unfortunately also means that the final recommendations quality is worse. Selecting users at random means that when the recommendation is computed and the algorithm searches for users similar to the user that is requesting recommendations, the selected group will not be very similar to the user. Selecting items from a category means that the recommendations will also belong to that category, which hurts item discovery. Removing popular and unpopular items means those will not be suggested and users who purchased them will not receive recommendations - which obviously can happen often for the most popular items.

3.1.2 Cluster Models

Cluster models are created by a Recommendation System by grouping users into separate clusters, with each cluster containing users that are the most similar to each other - searching for similar users becomes a classification problem. Once the Recommendation System has the whole user base divided into segments, ratings and purchases from a given cluster can be used to generate recommendations for users from the same cluster. The clusters can be generated manually but usually an automated algorithm is used. As mentioned previously, Recommendation Systems typically operate on very large data-sets. That is why usually to generate the clusters, some form of a greedy cluster generation is used - for example initially clusters are created by selecting users at random and then running a necessary amount of iterations to match users to those clusters, additionally merging segments or generating new along the way. Sometimes it is also necessary to reduce dimensions or introduce sampling, if the data-set is too complex. Sometimes users can be grouped into several segments at once, with an additional metric of the strength of the relationship that is used for generating recommendations. This approach performs well in typical online scenarios since the most expensive computation is done offline while the online comparison requires only checking a certain amount of segments. However, the final quality of the provided recommendations is poor. The segments contain many users which are all treated similar so the recommendation results are averaged. To make the results more accurate more segments are required, which has negative impact on the online speed of the Recommendation System.

3.1.3 Search-based filtering

Search-based filtering (or content-based filtering) understands the problem of creating a recommendation as a search for items that are somehow related. With this technique, the Recommender System tries to generate a search

query that could find items that have something in common with the items rated or purchased by the user. As an example, for a user that bought a DVD Collection, the Recommender System can recommend other popular movies from the same genre, starring the same actors, other movies directed by the same director. The first problem with such an approach is the complexity of the queries. If the Recommender System needs to give a recommendation to an active user with hundreds or thousands of ratings and purchases, the search query can be built only on a subset of the data, or some averaged-out data - which affects the final quality of the provided recommendations. The second issue is that a search-based Recommender System fails to deliver items for discovery - the results are either very general, e.g. the most popular movies from a given genre, or too limited, e.g. movies from the same director.

3.1.4 Slope One algorithm

Slope One is a very basic recommendation algorithm which can be quickly implemented, and yet often gives results very similar to more complex counterparts [3]. The basic idea behind the algorithm can be easily explained with a simple illustration, as presented in Figure 1.

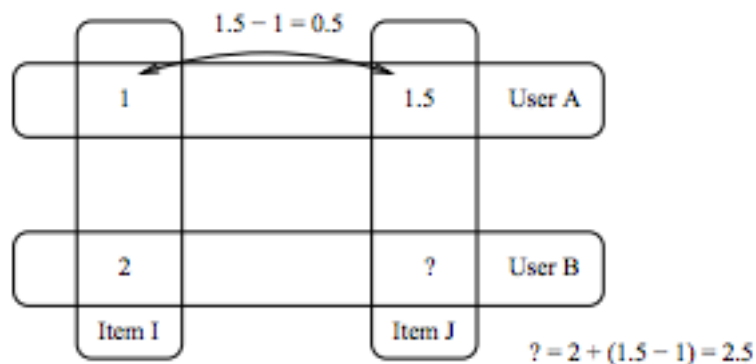


Figure 1: Slope One concept

Given two users - A and B, and two items, I and J, to predict users B rating of item J one must find the average difference between ratings of items I and J and simply add it to users B rating of item I. To further enhance the results a weighted average is used, so that the number of users that rated a given item is taken into consideration. This means that for Slope One to work, one must store only two matrices - one with average rating differences

between each pair of items and one with rating counts for common ratings for each pair of items.

3.2 Commercially used recommendation algorithms

As an example of currently used, widely popular recommendation algorithms, two will be described - Amazon Item-To-Item Collaborative Filtering and Last.fm Label Propagation Algorithm. Both are used in challenging environments and have been developed to serve demanding purposes and both are important elements of the platforms they are a part of.

3.2.1 Amazon - Item-To-Item Collaborative Filtering

Amazon uses their Item-To-Item Collaborative Filtering [6] as one of the most important marketing tools. Amazon has integrated recommendations into nearly every part of the purchasing process from product discovery to checkout. When browsing Amazon.com one will find multiple panes of product suggestions; navigating to a particular product page will present areas promoting items “Frequently Bought Together” or other items customers also bought. Recommendations are even included within the checkout pages on Amazon.com, functioning similarly to a supermarket checkout line with impulse items - but here they are targetted specifically for each customer.

Being one of the biggest e-commerce retailers in the world, generating recommendations that are up-to-date and convert well into purchases, is a huge challenge for Amazon, yet the algorithm behind it is not too complex and bears resemblance to the previously described Slope One algorithm (section 3.1.4).

The algorithm matches purchased and rated items and finds most-similar items by checking what items tend to be bought together by customers - this information is stored in a item-similarity table. But instead of creating a product to product matrix, which would be inefficient since most of the products do not have customers in common, the following iterative algorithm is used, as shown in Listing 1.


```
For each item in product catalog , I1
  For each customer C who purchased I1
    For each item I2 purchased by customer C
      Record that a customer purchased I1 and I2
  For each item I2
    Compute the similarity between I1 and I2
```

Listing 1: Item-to-Item iterative algorithm

This iterative approach finds the similarity between a product and all other products that were purchased along with it, using a simple cosine measure:

$$\text{similarity}(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|}$$

Although this calculation is expensive, it is done offline. With the item-similarity table in place, the algorithm finds items most similar to items that were purchased or recommended by a given user, and then recommends the most popular ones. This performs very well since users typically do not have too many purchases and recommendations.

3.2.2 Last.fm - Label Propagation Algorithm

Last.fm is a music online platform, founded in the United Kingdom in 2002. Last.fm gathers information about music listened to by people from all around the world thanks to a process named 'audioscrobbling' - Last.fm builds a detailed profile of each user's musical taste by recording details of the tracks the user listens to, either from Internet radio stations, or the user's computer or many portable music devices. This information is transferred ("scrobbled") to Last.fm's database either via the music player itself (from services such as Deezer, Spotify, Rdio, Amarok and many other) or through a plugin installed into the user's music player. The data is used to build the user's profile automatically, personalize content and suggest new artists. By April 2011, Last.fm had reported more than 50 billion scrobbles.

Working with such a large dataset was one of the challenges that Last.fm engineers faced when building a recommendation mechanism. The final algorithm was built in a way that could easily utilize MapReduce. MapReduce is a highly scalable programming model and associated implementation for processing large data sets in parallel with a distributed algorithm in a cluster, described by Google in 2004 [10] and heavily used in a variety of the most

important Google products. In case of Last.fm a popular open-source Java implementation is used, Hadoop MapReduce.

To generate valid recommendations Last.fm builds a graph where the vertices are users and artists, whereas the edges are created when a user expresses some form of interest with a given artist - e.g. listens to a track or starts following that artist, as presented in Figure 2. When making random walks in such a graph from a given user, if an artist is visited often and there are many short paths to that artist, the artist should be selected as a good recommendation, as presented in Figure 3. A graph random walk is equivalent to the label propagation algorithm which belongs to a family of algorithms that can be easily coded with MapReduce.

Label propagation algorithm step by step:

- Start with a partially labeled graph, in our example user nodes are labeled with songs that they listened to.
- Each label has an associated weight, in our case it is the count of how many times a track was listened to, as presented in Figure 4.
- Iterate as presented in Figure 5:
 - Propagate labels to adjacent nodes,
 - Accumulate and renormalize the received labels at each node,
- Final labels include unknown items.
- After running enough iterations or reaching convergence, new labels at user nodes are recommendations whereas new labels at item nodes are similar items.
- It is important to disregard hubs - very popular items or users that listen to everything. This is achieved by abandoning the random walk after a given number of steps, which can be mimicked in the label propagation algorithm by propagating a dummy label whose weight is based on the degree of the source node. In the final output, the dummy label is ignored.



Figure 2: Last.fm graph (with user U and music track t)



Figure 3: Last.fm graph (many short paths from U to t, t should be recommended)

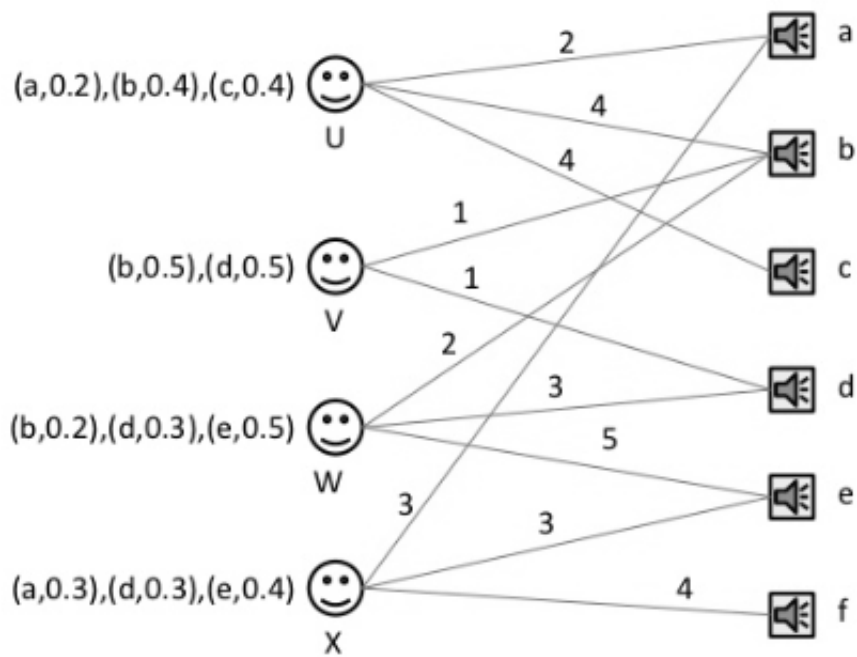


Figure 4: Last.fm label propagation algorithm graph - edge weights are counts of how many times a track was listened to, each user is labeled with the tracks he listened to and the proportion of the distribution of his interest between those tracks.

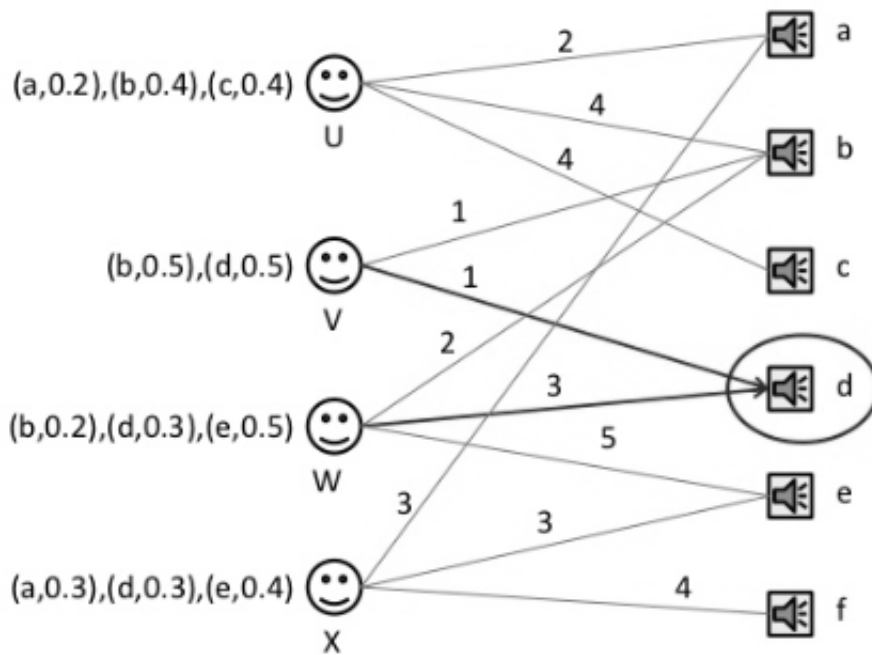


Figure 5: Last.fm graph: label propagation algorithm - accumulation and renormalization. For node d: $1 \times (b,0.5), (d,0.5)$, $3 \times (b,0.2), (d,0.3), (e,0.5)$ gives $(b,0.275), (d,0.35), (e,0.375)$. In the next iteration these labels will propagate to user V who will receive a recommendation of track e with a weight of 0.375.

4 Social-based recommendation algorithm concept

The amount of social data that is now openly available to any developer allows to research freely information that a few years ago was accessible to only a given few that were running the biggest social or e-commerce platforms in the world. The general idea behind the experimental algorithm that was created and implemented for the purpose of this thesis is to collect, explore and analyze social data that is accessible through the Facebook Social Graph, using it as the basis for a recommendation engine. To allow for verification of the results, the developed algorithm focused on musical interests of the users. However, thanks to the amount of different interests a Facebook user includes in his profile, it is in theory possible to use the same algorithm in other areas, especially in the entertainment field and with fast-moving consumer goods.

In order to test and verify such a concept, several web applications have been developed. An overview of the overall architecture is presented in Figure 6.

4.1 Data acquisition

Thanks to the well documented Facebook SDK it is not a complicated task to acquire the necessary data. The first step is registering a Facebook application, which supplies the developer a public and private API key that can be used for all calls to the Facebook API. After that all that is required is a simple script which works as a web application, which asks users to log in with their Facebook credentials to the previously created application and to allow access to all essential information, which then has to be fetched with appropriate FQL and/or Graph API queries and saved in a database for further analysis. Thanks to the social graph, a single user can give access to valuable information of all their friends - unless they opt-out of this feature, which is enabled by default. Initial tests showed that over 90% of users did not change this setting. The script first fetches unique identifiers of all friends of the user that logs in with his credentials and gives appropriate permissions to the script, and then performs FQL queries for each friend, resulting in thousands of records per one real user of the application.

There are many sections of a Facebook user profile that can be filled with likes for specific entities. The possible sections are ranked in Figure 7.

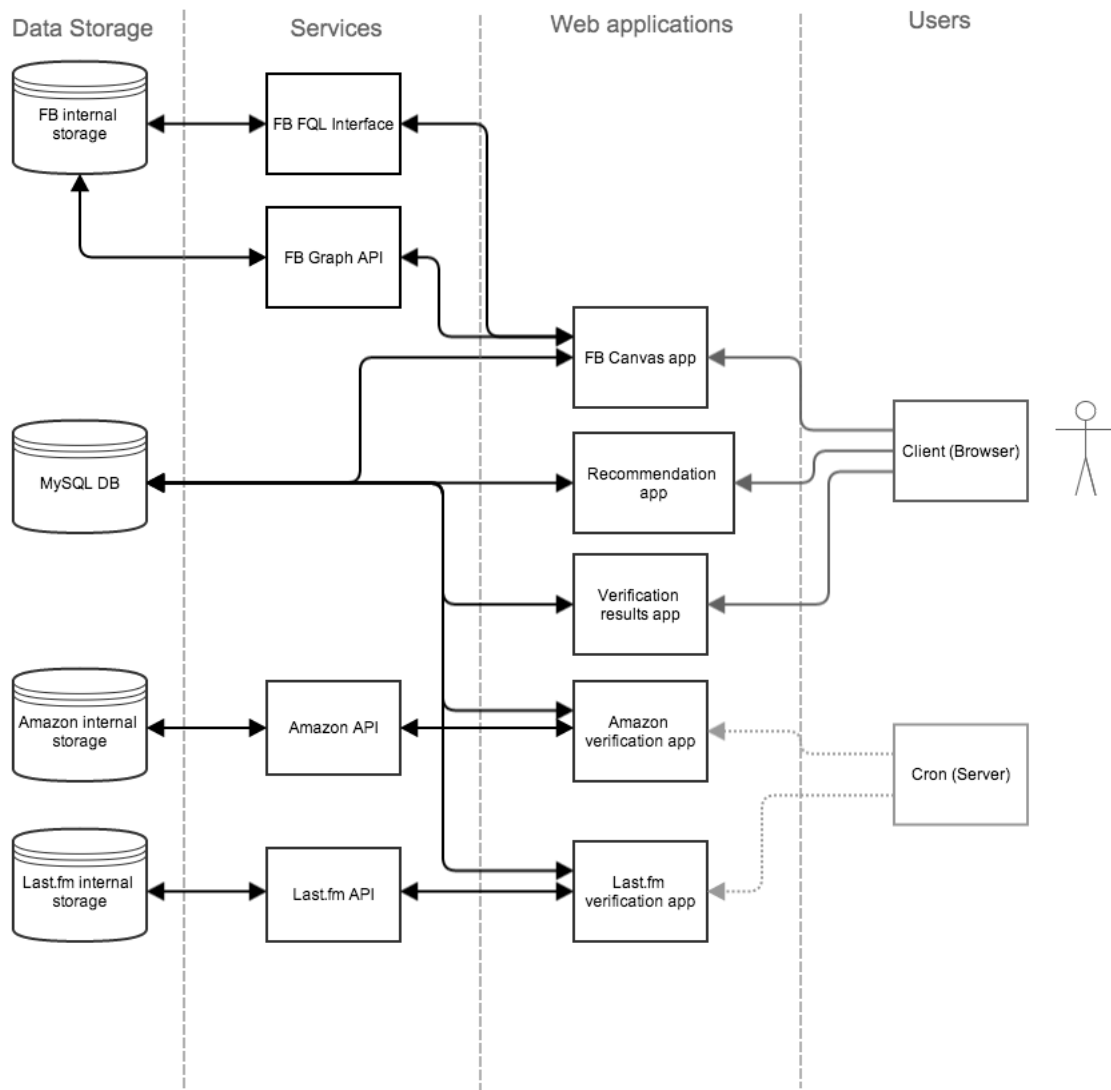


Figure 6: Application architecture

As an initial field of interest for this thesis the 'Music' section was analysed, because it is a popular interest among Facebook users and because it can be subjectively verified manually.

Many platforms are supported with the official Facebook SDK. For this thesis PHP was chosen as the primary language and the data was stored in a MySQL database, which is not optimal in terms of performance and scalability but is sufficient for the experiments described in this paper.












<input checked="" type="checkbox"/>	 About
<input checked="" type="checkbox"/>	 Photos
<input checked="" type="checkbox"/>	 Friends
<input checked="" type="checkbox"/>	 Places
<input checked="" type="checkbox"/>	 Music
<input checked="" type="checkbox"/>	 Movies
<input checked="" type="checkbox"/>	 TV Shows
<input checked="" type="checkbox"/>	 Books
<input checked="" type="checkbox"/>	 Likes
<input checked="" type="checkbox"/>	 Events
<input checked="" type="checkbox"/>	 Groups

Figure 7: Basic user profile sections

The information that was gathered from each user:

- Basic information about the user:
 - Age and gender,
 - Number of friends,
 - Number of interests,
 - Number of events to which the user has RSVP'd "attending";
- User's friends, understood as edges of the social graph - each database record stores two unique identifiers representing two vertices of the graph;
- User's interests, understood as vertices of the social graph - each database record stores the users unique identifier, the interest unique identifier and its category (when creating a fanpage, which becomes a separate social graph entity, the administrator has an option to choose a category for the page from a closed tree of hundreds of items). Even though all data is gathered from the "Music" section, not all interests belong to categories related with music. A "GROUP BY" query listed in Table

1 shows how many different categories are being fetched just from the "Music" section. That is why the input data is filtered and only entities from the "Musician/band" category are used for further analysis;

- Basic information about each artist:
 - Name
 - Music genre
 - Current location
 - Current number of all likes (popularity);

4.2 User data analysis

The characteristics of the basic data that was gathered from user profiles is worth a quick look.

- The number of friends per user is depicted on the histogram - Figure 8. Three groups of users can be observed:
 - typical users who have between 150 and 600 friends - approximately 80% of all users,
 - power users who have between 650 and 1000 friends - approximately 15% of all users,
 - single individuals who have either a very small (0-100) or a very high (1100-1700) friends count - approximately 5% of all users.
- The number of interests per user is depicted on the histogram - Figure 9. Because a typical exponential characteristic can be observed, the histogram is presented with a logarithmic scale. Very few users have more than 40 interests linked with their profile. Several heavy users can be observed with an interest count in the range of 70 to 145. Additionally one individual with approximately 290 interests exists, though he is not visualized on the histogram. An individual FB Graph query shows that this user is a popular music DJ who heavily uses Facebook, which explains the amount of interests listed.
- The number of events to which a user has RSVP'd 'attending' is depicted on the histogram - Figure 10. Similarly to the number of interests, a typical exponential characteristic can be observed, which is why the histogram is also presented with a logarithmic scale. Very few users have more than 9 events marked as 'attending'. Several heavy users can be observed with an event count in the range of 17 to 18. The individual heavy user with over 290 interests belongs to that group but does not stand out as with the interests count.

The analysis of the individual heavy user - musical DJ - suggests the three characteristics might be uncorrelated. That user has the most interests, but another user has the most friends. Similarly, there are several users that have more events marked as 'attending' in comparison to that user. To support this thesis and to investigate the possible relationship between the characteristics, three additional diagrams are presented below:

- Figure 11 - a scatter plot presenting the relationship between the number of friends and the number of interests linked to the users profile (users without friends or without interests are not taken into account),
- Figure 12 - a scatter plot presenting the relationship between the number of interests and the number of events to which a user has RSVP'd 'attending' (users without interests or without events marked as 'attending' are not taken into account).
- Figure 13 - a scatter plot presenting the relationship between the number of friends and the number of events to which a user has RSVP'd 'attending' (users without friends or without events marked as 'attending' are not taken into account).

For each of those relationships covariance and Pearson's correlation coefficient was calculated:

- Friends and interests:

- $\sigma(f, z) = 1625.7355$

- $r_{f,z} = 0.2339$

- Interests and events:

- $\sigma(z, v) = 79.9202$

- $r_{z,v} = 0.4881$

- Friends and events:

- $\sigma(f, v) = 231.1366$

- $r_{f,v} = 0.1748$

As suggested previously, all three graphs and the calculated values of covariance and Pearson's correlation coefficient show that indeed the three metrics (number of interests, number of friends, number of events marked as 'attending' by the user) are rather weakly correlated. The strongest correlation occurs between interests and events.

Another interesting observation is the fact that a vast majority of users allows 3rd party Facebook applications to indirectly access information about their friends, events and interests. Indirectly because those users profiles are accessed by the application through a single friend that used the application, without ever explicitly asking for permission or informing those users that data from their profiles is being accessed.

Table 1: Categories of interests fetched from the 'Music' section

Category	COUNT(*)
Musician/band	6161
Music	89
Musical genre	54
Album	54
Record label	41
Song	38
Community	23
Movie	13
Musical instrument	12
Artist	11
Arts/entertainment/nightlife	7
Interest	6
Playlist	6
Public figure	6
Music chart	4
Actor/director	4
Concert tour	3
Non-profit organization	3
Landmark	2
Internet/software	2
Radio station	2
Unknown	2
Personal blog	2
Music video	2
Teens/kids	1
Website	1
Dancer	1
Sport	1
Community organization	1
Author	1
Tv show	1
News/media	1
Book	1
Fictional character	1
Society/culture	1
Athlete	1
Entertainment	1
Producer	1
Media/news/publishing	1

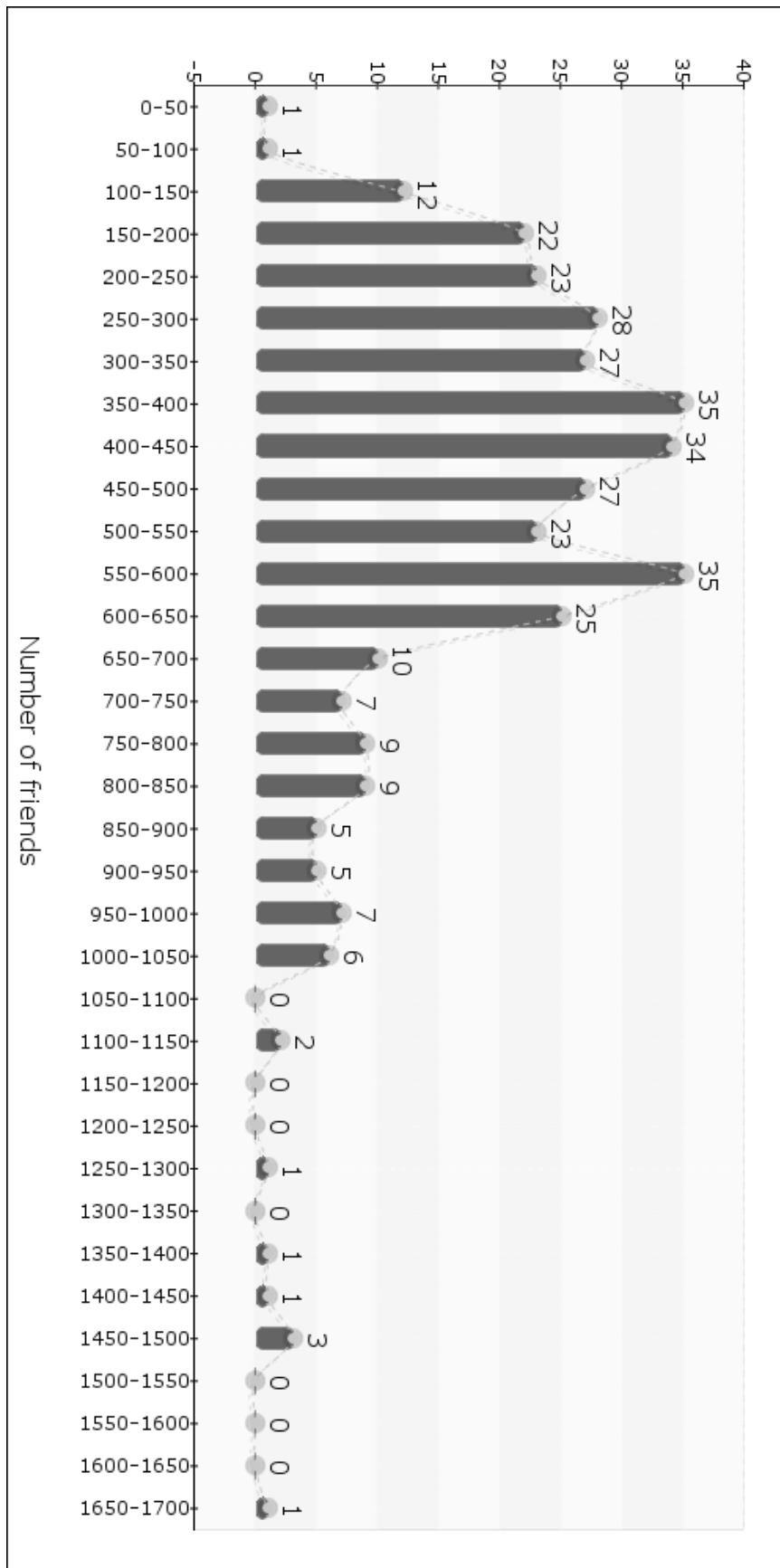


Figure 8: Histogram of the number of friends

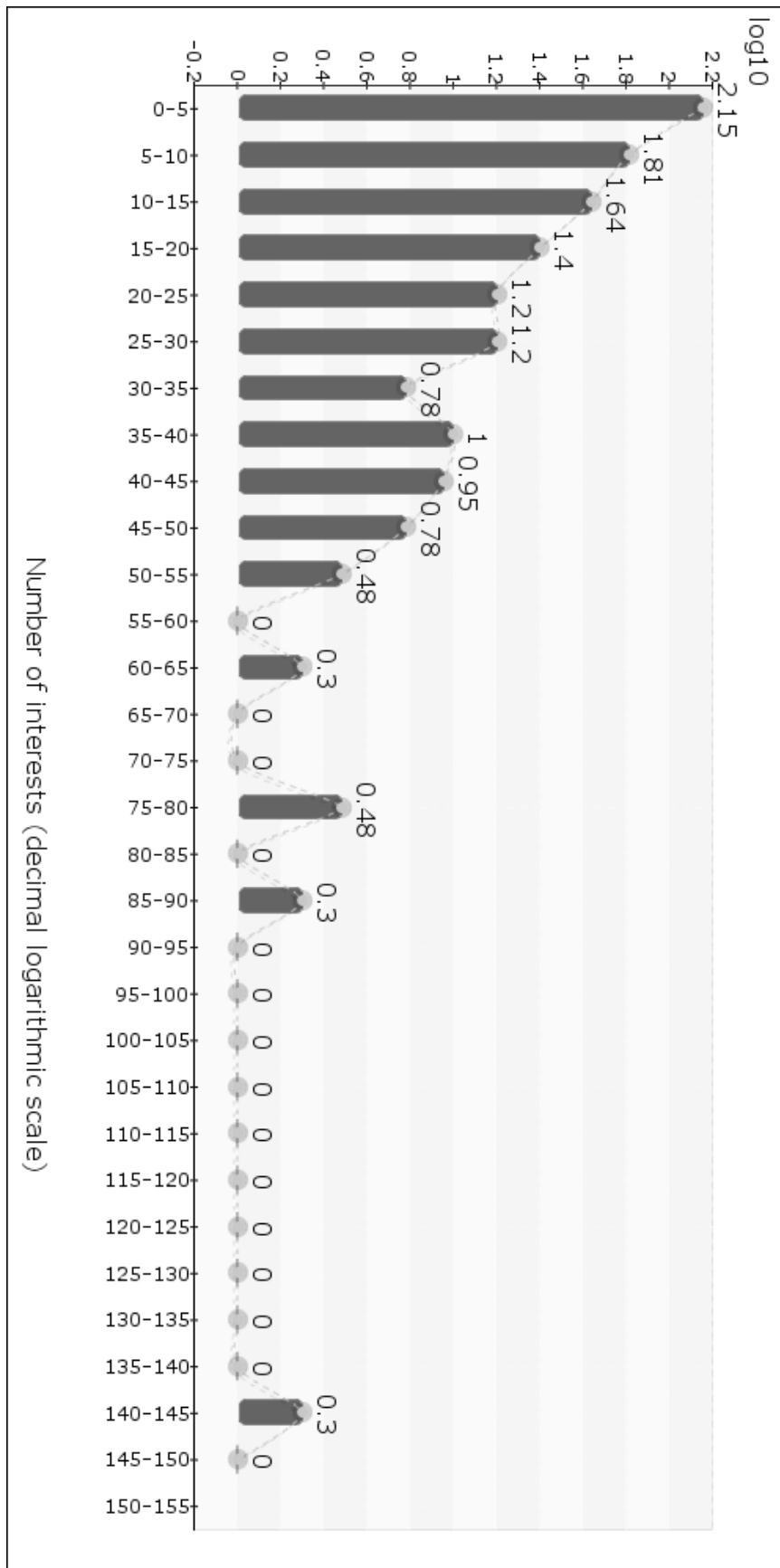


Figure 9: Histogram of the number of interests (decimal logarithmic scale)

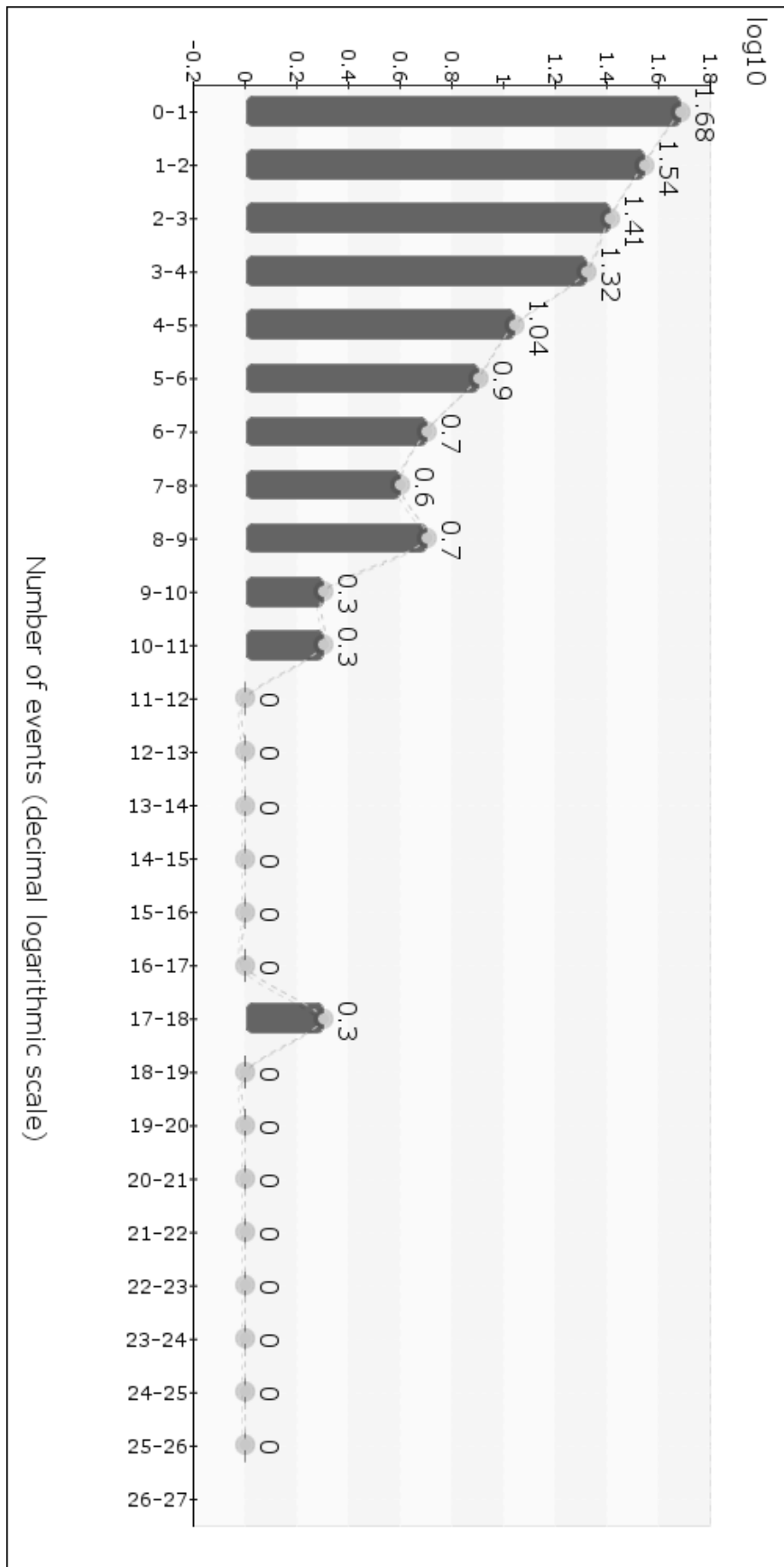


Figure 10: Histogram of the number of events RSVP'd 'attending' (decimal logarithmic scale)

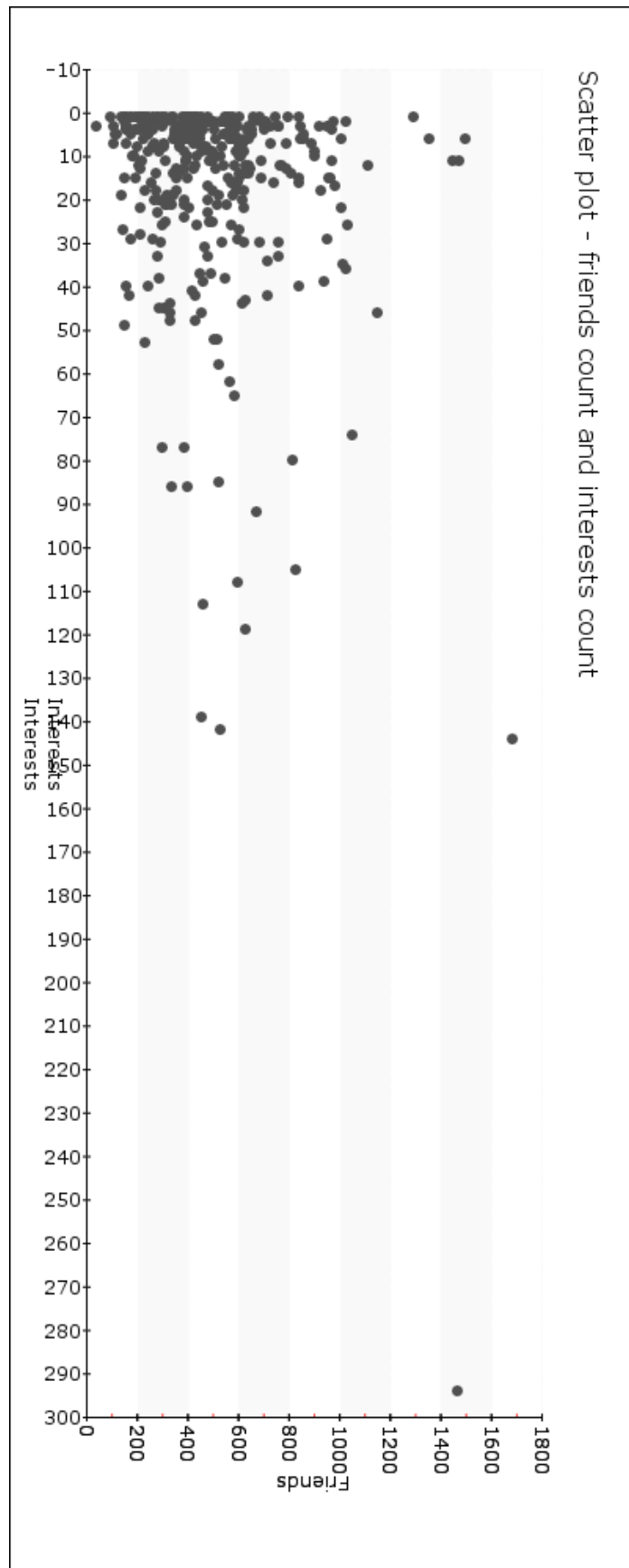


Figure 11: Scatter plot - friends count and interests count

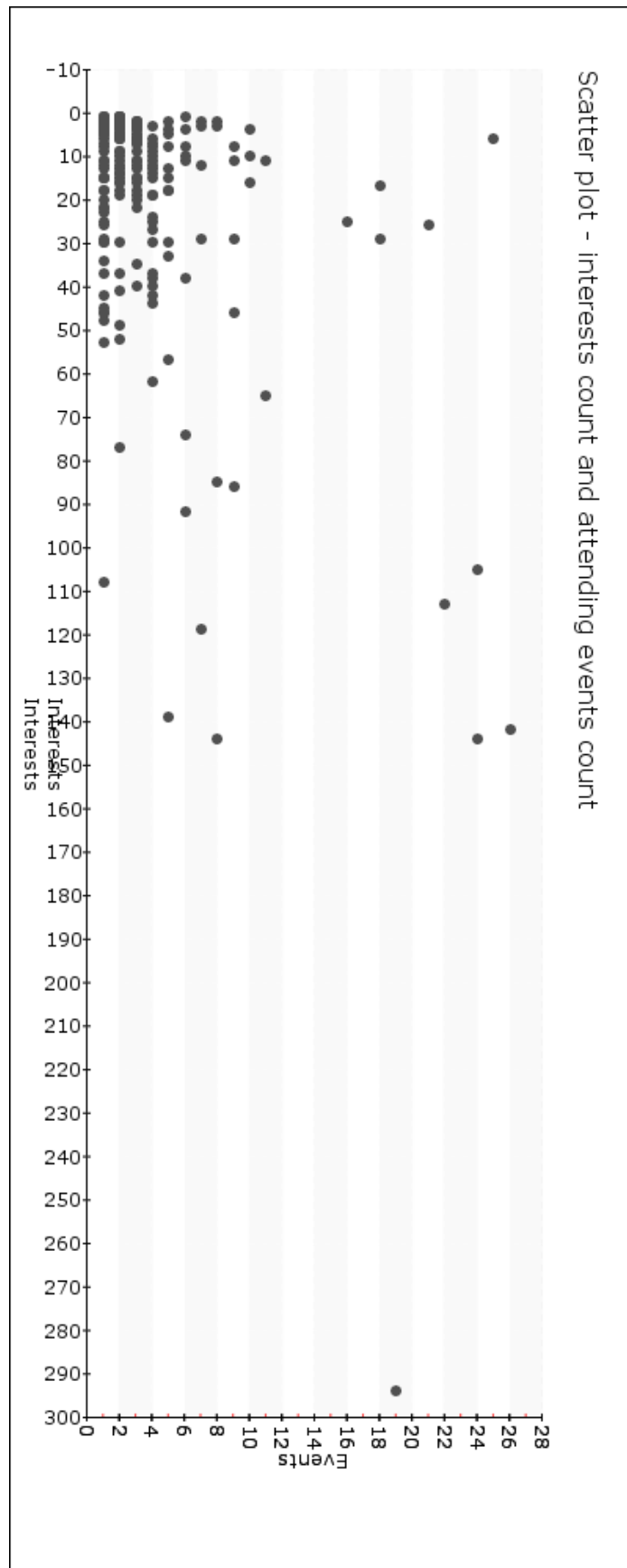


Figure 12: Scatter plot - interests count and attending events count

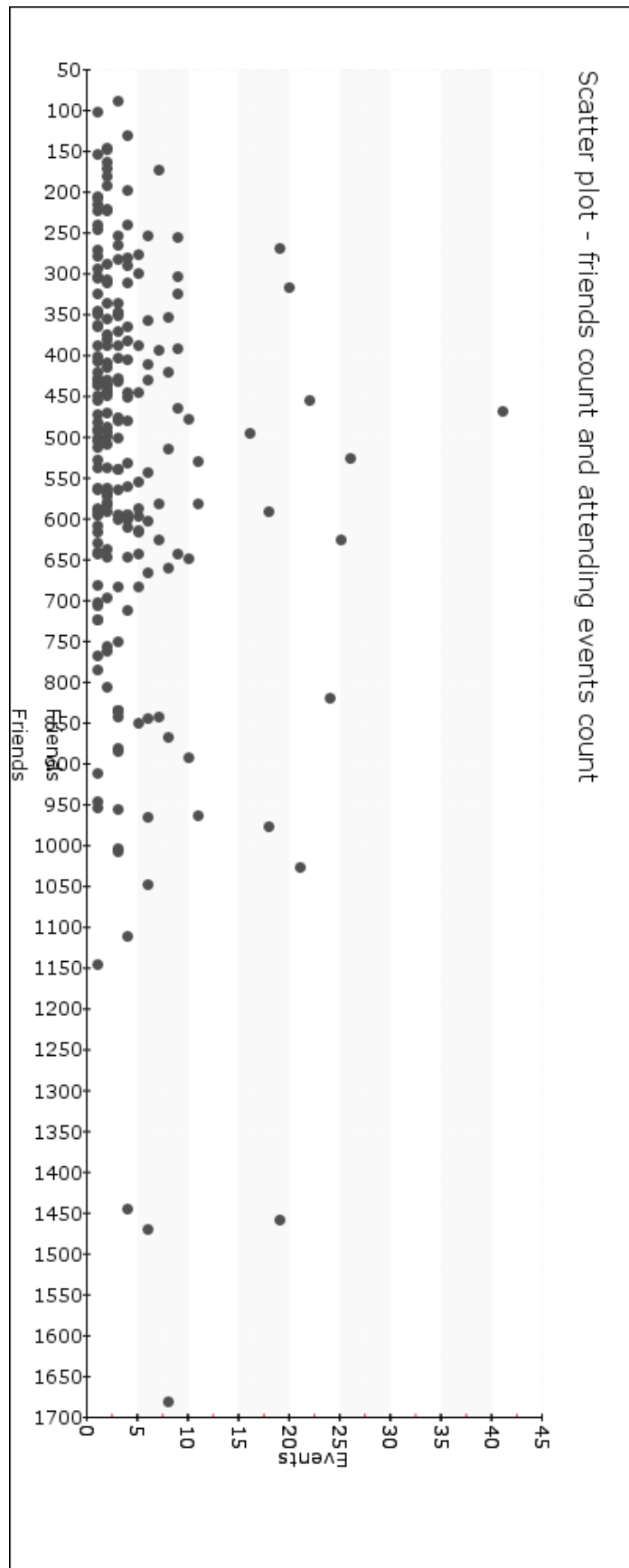


Figure 13: Scatter plot - friends count and attending events count

4.3 Building the graph

Having fetched all the necessary information from the FB graph, one can analyze and modify that data in an offline environment, without the need to query Facebook. To create a recommendation algorithm a graph was built, where the vertices were artists and the weight of the edges defined the similarity of the artists. Several assumptions had to be made in order to compute the artist similarity graph. Finally two approaches were selected - a basic implementation and a refined implementation. Only users that had more than a single interest and at least one friend were taken into account while analyzing the data.

Both implementations of the algorithm needed to answer two important questions:

- how to determine the weight of the edges (artist similarity)
- how to determine the activity of users ('quality' or 'strength' of the datasource)

Three basic parameters were selected to measure users activity:

- Number of interests
- Number of friends
- Number of events the user is attending in the near future

4.3.1 Basic implementation

Assumptions were made for both of the above questions.

- $G(U, A, E)$ - initial bipartite graph of user interests, $u_i \in U, a_i \in U$ where u_i represents a user and a_i an artist
- $u_i = \{f_i, v_i, z_i\}$ where f_i - friends count, v_i - events count, z_i - interests count
- $B_{\bar{U} \times \bar{A}} : b_{ij} = \begin{cases} 1 & \text{if } (u_i, a_j) \in E \\ 0 & \text{otherwise} \end{cases}$

B is an adjacency matrix of user interests.

Artist similarity can be measured by the amount of users that at the same time like both artists. This count needs to be normalized so that the amount of all users liking any of the two bands is taken into account - otherwise niche artists with few fans could never achieve an edge weight similar to the most popular artists. The final equations are presented below.

- $L(A, Q)$ - final graph of artists and the strength of the similarity between them.

$$- q_{ij} = \frac{2 \sum_k s^2(a_i, a_j, u_k)}{\sum_k b_{ki} + \sum_k b_{kj}}$$

For user 'strength' or activity, an assumption was made that users who attend many events and have a lot of friends, while having few interests, are the best source of information - one could assume all that social activity is focused on just those few interests. Thanks to the findings described earlier in 4.2 - the exponential nature of those three values - they were normalized. The final equation is presented below:

$$s(a_i, a_j, u_k) = b_{ki} b_{kj} \frac{\log f_k + \log v_k}{\log z_k}$$

4.3.2 Refined implementation

A slightly more complex algorithm was also proposed. Artist similarity was again measured by the amount of users that at the same time like both artists, but this time it was normalized to the count of the less known of the two artists. What is more important, the 'strength' value was now calculated differently as well - average values for each of the three components (number of friends, number of interests, number of events being attended) were taken into consideration so that users that in general were above average would have more 'strength'. The assumptions made were as presented below:

$$\tilde{q}_{ij} = \frac{\sum_k b_{ki} b_{kj} \cdot \log \sum_k \tilde{s}(a_i, a_j, u_k)}{\min(\sum_k b_{ki}, \sum_k b_{kj})}$$

$$\tilde{s}(a_i, a_j, u_k) = \frac{\log(f_k)}{\log(\sum_m f_m / \bar{U})} + \frac{\log(v_k)}{\log(\sum_m v_m / \bar{U})} + \frac{\log(z_k)}{\log(\sum_m z_m / \bar{U})}$$

4.4 Initial results

The first results could be quickly verified by simply subjectively browsing through them and reviewing them manually.

Table 2: Top 10 results of the basic implementation

Artist 1	Artist 2	Similarity
Kings Of Leon	Coldplay	6.42623
The Doors	Pink Floyd	5.12121
Kanye West	MGMT	4.96552
Red Hot Chili Peppers	Coldplay	4.84
Radiohead	Massive Attack	4.74074
Paktofonika	Pezet	4.57143
The Beatles	The Doors	4.56604
Róisín Murphy	MIA	4.43077
Lady Gaga	Rihanna	4.37838
O.S.T.R.	Pezet	4.35946
...

Table 3: Top 10 results of the refined implementation

Artist 1	Artist 2	Similarity
Mozart	Haendel	1.1644
Children Of Bodom	Megadeth	0.959369
Godsmack	Disturbed	0.959369
Iron Maiden	Judas Priest	0.959369
Kings Of Leon	Coldplay	0.919527
The Doors	Pink Floyd	0.919527
Muse	Radiohead	0.894538
Czerwone Gitary	Budka Suflera	0.894538
Michael Jackson	The Beatles	0.874318
Bob Marley	Pink Floyd	0.874318
...

A basic subjective review of the results of both algorithms, although each provides different top results, suggests that each of them shows promise and is worth further exploration. For example, the first results for the basic implementation showcase, among other results, a diverse but accurate list:

- Kings Of Leon and Coldplay - two rock bands representing a similar genre of alternative rock. Both bands have performed in Poland, which explains a lot of interest expressed by the analyzed Facebook users,
- The Doors and Pink Floyd - two rock classics, representing similar genres of psychedelic rock and blues rock. Both began their careers in mid 60's and quickly became definitive icons of their era,
- Paktofonika and Pezet - two Polish hip-hop and rap artists,
- Lady Gaga and Rihanna - two very popular pop singers with a similar music style, both known for their trend-setting passion for excentric fashion.

The refined algorithm boosts scores gathered from heavy users which can be seen in the first results list, yet the results are still accurate:

- Mozart and Haendel - two famous composers of classical music,
- Children of Bodom and Megadeath, Godsmack and Disturbed, Iron Maiden and Judas Priest - all being iconic heavy metal bands. Those three results are also very accurately paired with heavy metal subgenres represented by each pair and even with years of musical presence,
- The Doors and Pink Floyd - a pair highly ranked by both algorithms,
- Czerwone Gitary and Budka Suflera - two classic Polish bands.

5 Cloudera Oryx

Cloudera, Inc. was founded in October 2008 in Palo Alto, California, with the goal of creating an enterprise implementation of Apache Hadoop. Before analyzing the Oryx project, it is worth to briefly discuss Apache Hadoop.

Apache Hadoop is an open-source project that develops software for reliable, scalable, distributed computing [8]. Currently Apache Hadoop consists of four subprojects:

- Hadoop Common - a set of libraries and utilities that are used in other subprojects,
- Hadoop Distributed File System - a distributed file system. In 2003 Google published documentation describing the Google File System (GFS)[9], but without any implementation details. Hadoop Distributed File System (HDFS) is a Java-based implementation of GFS.
- Hadoop YARN (Yet Another Resource Negotiator) - resource management technology for virtual environments.
- Hadoop MapReduce - a Java-based implementation of a distributed computing paradigm, based on the model described by Google in 2004 [10].

Cloudera Oryx was released in November 2013 as continuation of the Myrrix project - Myrrix was a complete, real-time, scalable recommender system, evolved from Apache Mahout. In July 2013 it became a part of Cloudera [11].

5.1 Oryx description

Cloudera Oryx is an open source project that aims to provide a real-time, large-scale infrastructure for machine learning and predictive analytics. The project comes with out of the box implementation of several popular algorithms - for the purpose of this paper we will be focusing on the collaborative filtering and recommendation capabilities of Oryx.

5.2 Oryx architecture

Oryx is divided into two layers - the Computation Layer and the Serving Layer. The job of the first layer is to build machine learning models and the job of the second layer is to serve models.

- Computation Layer - Java-based, long-running, offline, batch process. The Computation Layer continuously builds out machine learning models based on a snapshot of input at a point in time. Both input and output happen in HDFS with the use of PMML files, described in 5.4. The Serving Layer monitors the appropriate directory and automatically loads new models as they become available. The Computation Layer can run in two modes:
 - Distributed - the typical mode, based on Hadoop MapReduce.
 - Local - simplified mode, running locally in-memory with input and output being read and written to a local file system.
- Serving Layer - Java-based, long-running server process that exposes a REST API.

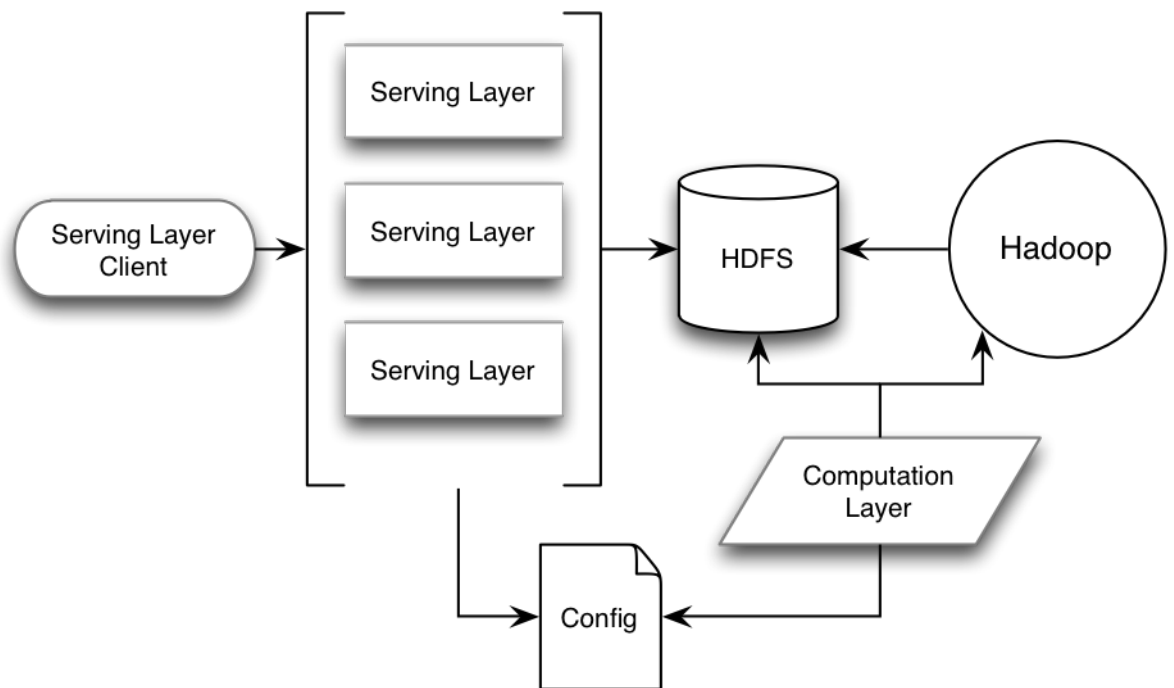


Figure 14: Oryx architecture overview [<https://github.com/cloudera/oryx>]

5.3 Oryx Collaborative Filtering algorithm

Oryx recommender engine implementation uses a variant of the 'Alternating Least Squares' (ALS) algorithm - Alternating-Least-Squares with Weighted-Regularization (ALS-WR)[12]. This very variant of ALS won the The Netflix Prize Contest in 2009 - a open contest hosted by Netflix in which teams were competing to create the best collaborative filtering algorithm that could predict user ratings for films, based on their previous ratings. In 2009 a team consisting of engineers from Yahoo and ATT Labs managed to create an algorithm that beat the original Netflix engine, and that algorithm is now used in Oryx.

The basic problem ALS-WR is trying to solve is to estimate the missing values in a user-movie matrix where each value represents a rating given by a user to a movie. Each user and each movie is represented by a vector of features and each rating should be based on those features. The problem is described as minimalization of the loss between the rating and the scalar product of the feature vectors - in case of this algorithm, a mean-square loss function is used. Because the initial dataset is sparse - most of the users ranked very few movies - there is a risk of overfitting. That is why additional regularization is introduced. In order to solve this problem, the optimal user and movie features vectors are needed. To find the features vectors alternating least squares is used as follows:

- Step 1: Initialize matrix M (movie features vector) by assigning the average rating for that movie as the first row, and small random numbers for the remaining entries.
- Step 2: Fix M , Solve U (user features vector) by minimizing the objective function (the sum of squared errors);
- Step 3: Fix U , solve M by minimizing the objective function similarly;
- Step 4: Repeat Steps 2 and 3 until a stopping criterion is satisfied.

To solve U , each column of U is determined by solving a regularized linear least squares problem involving the known ratings of the user, and the feature vectors of the movies that user has rated. Solving M is done in the same manner.

The mentioned stopping criterion in this case is a satisfyingly low root-mean-square deviation between iterations. Once the optimal user and movie features vectors are found, they can be used to calculate the final ratings by solving the original problem.

5.4 PMML documents

The Predictive Model Markup Language (PMML) is an XML-based language that allows to define and describe data mining models and statistical models. PMML is vendor-independent which allows users to develop models in one application and then use other applications to analyze, visualize and work with the models in a straightforward and standardized matter.

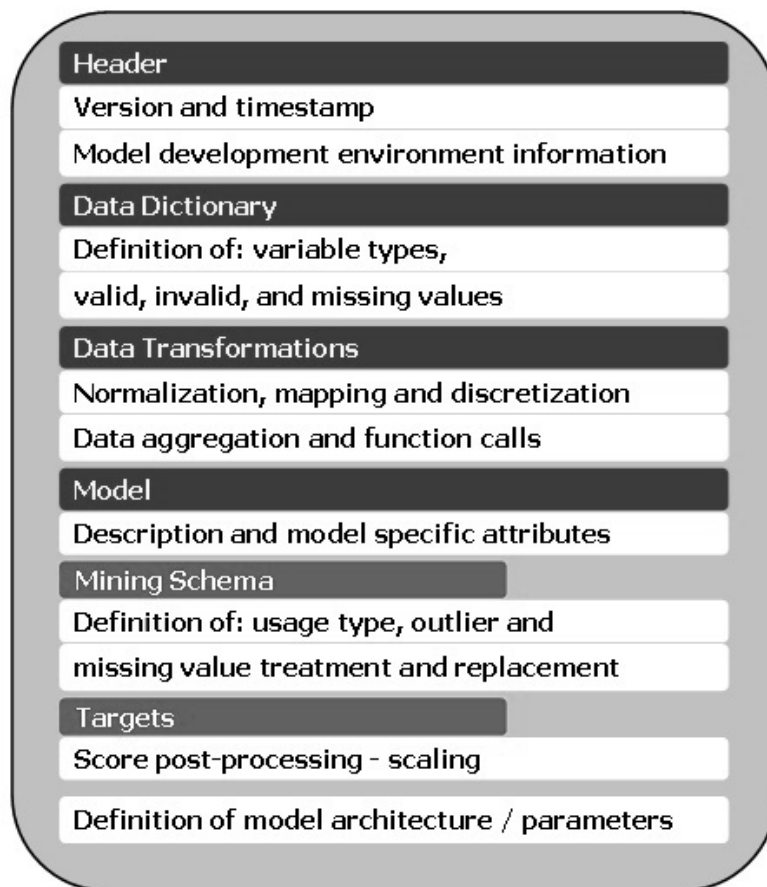


Figure 15: PMML Components

5.5 Apache Mahout

Apache Oryx (originally Myrrix) was built out heavily based on Apache Mahout[16]. The goal of the Apache Mahout project is to build a scalable machine learning library. It is used for recommendations, ad targeting and predictions by some of the current biggest names in technology - Foursquare,

AOL, LinkedIn, Twitter and many others [17].

Apache Mahout implements several Collaborative Filtering algorithms:

- User-Based Collaborative Filtering
- Item-Based Collaborative Filtering
- Matrix Factorization with Alternating Least Squares

5.6 Oryx initial test

Running a simple, local test with Oryx is rather straightforward. Latest releases of both the computation and serving layer can be downloaded as a .jar archive from GitHub¹. Both layers share a single configuration file.

The simplest version of the config that will allow to perform a local test is listed below:

```
model=${als-model}
model.instance-dir=/Users/oryx/local
serving-layer.api.port=8091
computation-layer.api.port=8092
model.local-computation=true
model.local-data=true
```

Listing 2: oryx.conf

Oryx will automatically load any .csv input dataset files that are placed in the model instance directory (set with the model.instance-dir parameter in the configuration file), within the *generation*/inbound directory, where *generation* is an index - generations start from 00000, with each directory containing the input (inbound) dataset snapshot and, once the computing layer is finished, output files containing the model that can be then consumed by the serving layer. Both serving and computing layers detect changes within those directories automatically and directories for new generations are also created automatically.

The inbound data must be formatted in a structured matter: user,item,weight (where weight is optional, by default it is 1.0). User and item are strings. For the purpose of this test a dump from Audioscrobbler from 2005 (before it was acquired by Last.fm) was used as input data. Audioscrobbler was a set of plugins for popular music players that aggregated information about

¹<https://github.com/cloudera/oryx/releases>

what music was listened to and in return offered top charts and recommendations. The .csv file is structured as follows: user identifier, artist, count (how many times any track of the artist was listened to by the user). The csv file contains 656620 records.

A sample listing for one of the users is presented below:

```
...
1000067," Alice_in_Chains" ,107
1000067," The_Beach_Boys" ,129
1000067," Aretha_Franklin" ,148
1000067," Jamie_Cullum" ,105
1000067," Katie_Melua" ,197
1000067," Red_Hot_Chili_Peppers" ,184
1000067," R.E.M. " ,149
1000067," Manic_Street_Preachers" ,252
1000067," Pixies" ,218
1000067," Weezer" ,330
1000067," Pearl_Jam" ,159
1000067," Nina_Simone" ,180
1000067," Green_Day" ,110
1000067," Norah_Jones" ,389
1000067," Otis_Redding" ,131
...
```

Listing 3: audioscrobbler.csv

The test was performed on a 2.26GHz Intel Core 2 Duo MacBook Pro With 8GB DDR3 RAM memory, running on OS X Mavericks (10.9.2) with Oryx release version 0.5.1.

Starting the long-running computation layer process is as easy as performing a single command:

```
java -Dconfig.file=oryx.conf -jar oryx-computation
-0.5.1.jar
```

The computation process started the (included within the jar) servlet container (Tomcat), started the main servlet and noticed no available generations, which immediately forced to start generating the models from the *00000/inbound/audioscrobbler.csv* input file. The computation layer ran 30 iterations of the ALS algorithm and wrote all the necessary output files. The whole process took approximately 5 minutes with about 10 seconds needed

for each iteration of ALS.

While the computation layer was working the serving layer was started, also with a single command:

```
java -Dconfig.file=oryx.conf -jar oryx-serving-0.5.1.jar
```

Once the web servlet was running, a web interface of the serving layer was exposed:

```
INFO: Serving Layer console available at http://192.168.0.105:8091
```

It took the serving layer a couple of seconds to notice the files generated by the computation layer. The latest model was loaded instantly:

```
INFO: All model elements loaded , 73458 users and 47065 items
```

Using the web interface it was now possible to fetch suggestions. For example, running a GET request for *similarity/Pink Floyd* returned:

```
Led Zeppelin ,0.9319506
The Beatles ,0.7827824
The Doors ,0.74322164
Soulcracker ,0.69468755
Captain Bogg & Salty ,0.6938288
Poul Dissing ,0.6567173
Domenico Scarlatti ,0.6377974
The Rolling Stones ,0.6323819
```

This basic test shows how simple it is to work with Oryx, even without understanding the underlying algorithms and technology Oryx can be a great addition to existing platforms where a recommendation solution is required. The split into two layers is also worth highlighting - thanks to the serving layer the end user has constant access to the latest available information with a asynchronous, pleasant user experience, while the computation layer can be constantly updating the models and serving them to the serving layer only when they are ready.

5.7 FB Graph data with Oryx

Preparing an input CSV file from the data gathered as described in subsection 4.1 was a rather straightforward task. The file is very similar to the

Audioscrobbler example, but this time the last parameter, weight, is omitted so that it defaults to 1.0.

A sample listing for one of the users is presented below:

```
...
1138818755,UKF Dubstep
1138818755,Gooral
1138818755,Projekt WARSZAWIAK
1138818755,Sparks And Fuel
1138818755,Sovinsky
1138818755,koVValsky
1138818755,donGURALesko
1138818755,Snoop Dogg
1138818755,Arctic Monkeys
1138818755,MIKA
1138818755,Lenny Kravitz
1138818755,Johnny Cash
1138818755,David Guetta
1138818755,Kanye West
1138818755,Amy Winehouse
1138818755,Queen
1138818755,Pink Floyd
1138818755,Red Hot Chili Peppers
1138818755,Coldplay
...
```

Listing 4: graph.csv

The test was performed on the same hardware as for the example Audioscrobbler data-set and with the same version and configuration of Oryx. Using the similarity endpoint of the Oryx serving layer, a similarity graph was built, with a structure very similar to the original graph described in subsection 4.3. An example of the results is presented in the table below.

Table 4: Oryx graph for FB Social Graph interests

Artist 1	Artist 2	Strength
...
Handel	Bach	1.0
Handel	Igor Stravinsky	0.97780216
Handel	Prokofiev Sergei	0.97455966
Handel	Włosi & Lasoniowie	0.97455966
Handel	Henryk Wieniawski	0.97455966
Handel	Daniil Trifonov	0.97455966
Handel	Ingolf Wunder	0.91570157
Handel	AGA ZARYAN	0.85863763
Handel	Czesław Mozil	0.83407235
...
Coldplay	Kings Of Leon	0.60366696
Coldplay	Red Hot Chili Peppers	0.50101006
Coldplay	NERO	0.42829087
Coldplay	Kelis	0.42829087
Coldplay	U2	0.42186016
Coldplay	Kanye West	0.41922408
Coldplay	Hans Zimmer	0.39399052
Coldplay	Adele	0.38684645
Coldplay	Lenny Kravitz	0.3814772
...

6 Results verification

It was of essential importance to find a proper way of verifying the results of the proposed algorithms. Recommender systems, though extremely popular in recent years, operate in a subjective realm, making it difficult to indisputably rank their effectiveness. Two data sources have been explored to rank the results:

- Amazon.com *Customers Who Bought This Product Also Bought...* - a popular Amazon.com feature that displays information about products often bought together, which implies their similarity.
- Last.fm *Similar Artists* - one of the basic features of Last.fm, with data gathered from its 50 million users.

6.1 Verification with Amazon recommendations

Amazon.com *Customers Who Bought This Product Also Bought...* feature was selected as a good verification mechanism because purchases imply a strong connection to a given item - if albums of two artists are often bought together, they should be returned as a recommendation. Also, the Amazon product collection is huge which gives promise of results for most of the entities in the artists graph.

From a technical point of view accessing this information proved to be challenging. The only API that exposes that data - indirectly - can be accessed through the Amazon Associates Program for Amazon.com, which is typically used when one would like to offer Amazon products on their website. For example, an owner of an online blog featuring movie reviews could link to Amazon product pages for movie DVDs. For each movie sold on Amazon that was referenced through his website, the owner of the blog would earn a commission. In such a scenario the owner of the blog can also include a *Customers Who Bought This Product Also Bought...* section on his website - the API can be consumed with the use of a mostly undocumented PHP SDK. The API returns the results in subsets of 5 results with random order, which meant that for every artist the request had to be performed several times. The amount of results was different for each artist so every time a new artist was detected in the results, the requests counter for each artist was reset - this meant that at least several API responses were received with no new artists. Additionally the API returned results after several seconds and errored-out when too many requests were made. A simple watchdog script

was introduced that would verify the results are gathered correctly. When any error was detected a rollback was performed up to the point of the last fully analyzed artist, and the script would be paused (sleep) for a while. Although the whole process took a lot of time, all artists were automatically sent to the Amazon API and the returned values were stored. The end result was a graph of similar artists, with an example listing for the first two artists provided below in Table 5.

Table 5: Amazon *Customers Who Bought This Product Also Bought...* graph

ID	Artist 1	Artist 2
1	Kings Of Leon	Mumford & Sons
2	Kings Of Leon	Florence + the Machine
3	Kings Of Leon	Foster the People
4	Kings Of Leon	The Black Keys
5	Coldplay	The Fray
6	Coldplay	John Mayer
7	Coldplay	Radiohead
8	Coldplay	The Killers
9	Coldplay	Keane
10	Coldplay	U2
11	Coldplay	Death Cab for Cutie
12	Coldplay	Adele
13	Coldplay	Onerepublic
14	Coldplay	Snow Patrol
...

6.2 Verification with Last.fm recommendations

To further enhance the verification mechanism, a second datasource was introduced. Last.fm directly exposes an API that returns recommendations for a given artist and even offers a auto-correct feature, which performs a fuzzy search on the artists name. The recommendations are built based on 'audio scrobbles' - as mentioned in section 3.2.2, Last.fm offers a set of plugins for popular music players that automatically gather information about what music was listened to. In return Last.fm users can expect a more personalized experience.

From a technical point of view working with the Last.fm API proved to be much easier than the Amazon Associates Program for Amazon.com API.

A well-documented PHP SDK is available and fetching the results is straightforward. Similarly to Amazon, the Last.fm API does not allow for requests that happen too often so a similar watchdog was introduced. Because the results are returned quickly, the automated mechanism was also randomly paused (sleep) for several seconds. The end result was a graph very similar to the one built with the Amazon API but with significantly more results, partially listed below in Table 6.

Table 6: Last.fm recommendations graph

ID	Artist 1	Artist 2
1	Kings Of Leon	The Black Keys
2	Kings Of Leon	Coldplay
3	Kings Of Leon	Foo Fighters
4	Kings Of Leon	Kasabian
5	Kings Of Leon	The Killers
6	Kings Of Leon	White Lies
7	Kings Of Leon	The Kooks
8	Kings Of Leon	Arctic Monkeys
9	Kings Of Leon	Band of Horses
10	Kings Of Leon	Mumford & Sons
11	Kings Of Leon	Cage the Elephant
12	Kings Of Leon	Editors
...
123	Kings Of Leon	Empire of the Sun
124	Coldplay	OneRepublic
...

6.3 Amazon and Last.fm results comparison

The final numbers prove the two selected sources for verification were a good choice:

- The original artists database consisted of 1119 entries,
- Amazon API returned results (recognized the artist) for 365 artists,
- Amazon API returned on average 5.4771 results per artist
- Last.fm API returned results (recognized the artist) for 871 artists,
- Last.fm API returned on average 95.0526 results per artist

6.4 Verification results

The final, verified results are promising:

- Basic implementation - the final graph contains 157922 edges. On the other hand Last.fm found matches just for 1250 edges. This suggests that most of the edges are just noise that should not be taken into account. The similarity metric described in subsection 4.3.1 is of exponential nature which further proves this observation. After computing the natural logarithm of that metric, a edge case value of 1.0 was selected. For edges with the metric greater than or equal to 1.0 Last.fm matched positively 1050 pairs, whereas only 200 pairs were matched with a metric of less than 1.0. Amazon matched only 300 edges, but 240 of those had the metric greater than or equal to 1.0. Both verification sources prove that the basic implementation performs well, with a false acceptance rate at roughly 15%.
- Refined implementation - this implementation is based on the same graph so again it contains almost 160 thousand edges. False acceptance rate and false rejection rate based on Last.fm verification in dependence of the cut-off value of the refined metric are presented in Figure 16. Using the FRR and FAR values, for this metric an edge case cut-off value of 0.25 was selected. For edges with the metric greater than or equal to 0.25 Last.fm matched positively 1160 pairs, whereas only 90 pairs were matched with a metric of less than 0.25. Out of the 300 edges verified by Amazon, 250 had the metric greater than or equal to 0.25. Both verification sources prove that the refined implementation performs slightly better than the basic implementation, with a false acceptance rate at roughly 10%.
- Oryx - the final graph built by Oryx contains much less noise with just 8450 edges, which is understandable given that it was built by requesting similar artists. Last.fm found 1510 matches for those edges. Oryx returns similarity in the range of 0.0 - 1.0. FAR and FRR based on Last.fm verification in dependence of the cut-off value of the Oryx result metric are presented in Figure 17. For this metric an edge case value of 0.5 was selected. For edges with the metric greater than or equal to 0.5 Last.fm matched positively 1420 pairs, whereas only 90 pairs were matched with a metric of less than 0.5. Amazon verified 210 edges, out of which 180 had the metric greater than or equal to 0.5. This means that for Oryx the false acceptance rate is proportionally roughly the same as for the proposed refined algorithm.

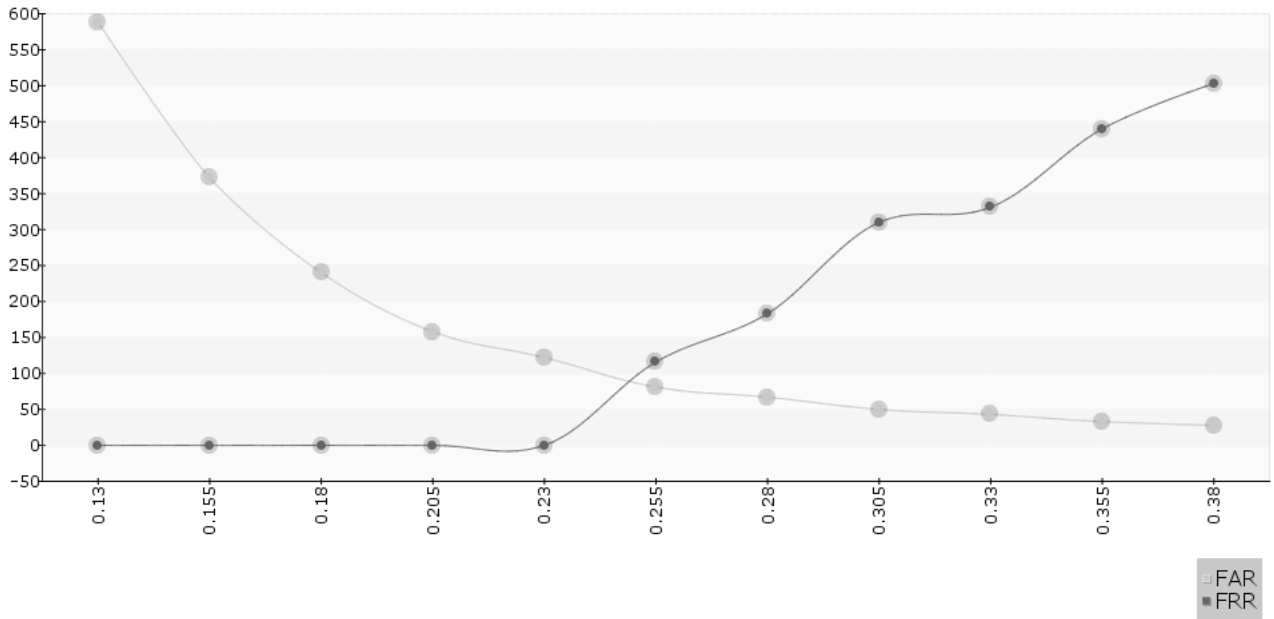


Figure 16: Refined implementation - FAR and FRR (based on Last.fm)

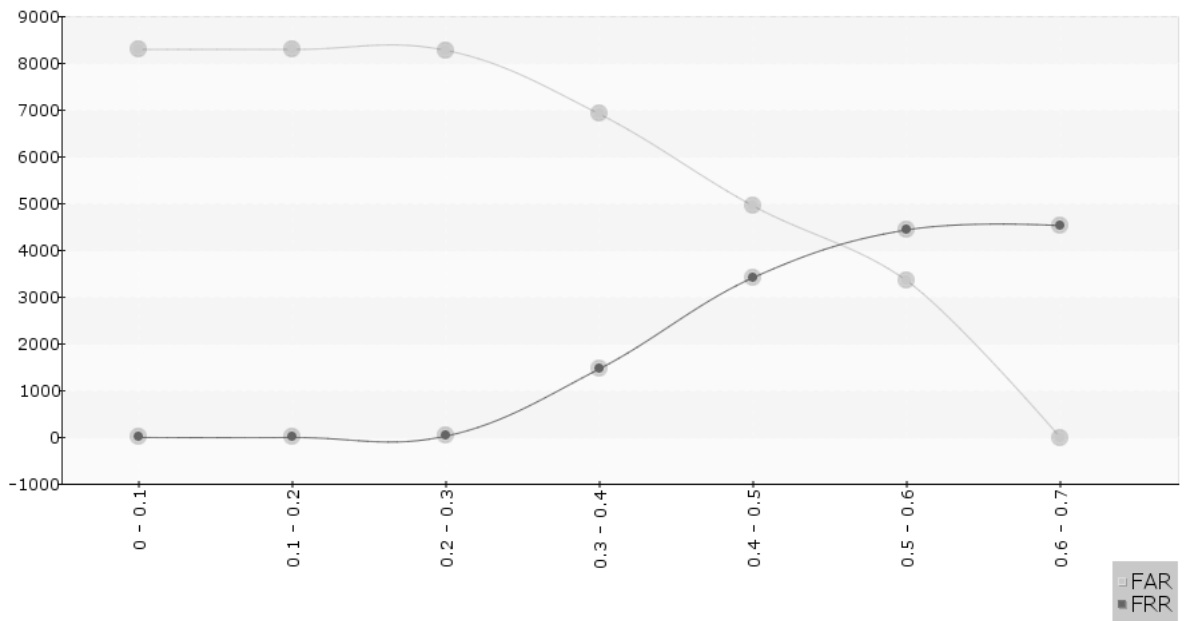


Figure 17: Oryx - FAR and FRR (based on Last.fm)

6.5 Conclusion

The tests and comparisons performed on the developed algorithms and Oryx prove that in the world of Web 2.0 social networks are an interesting and innovative data source for Recommendation Systems. Both algorithms created, although simple in their nature, provide good results, comparable with a much more sophisticated Oryx. The verification mechanisms created, although not perfect, provide valuable insight about the developed algorithms.

Recommendation Systems are already being successfully utilized in a variety of use cases. Using data from social networks allows for new and additional usage, for example suggestions for a group of users, be that a casual group of friends or a group of colleagues at work. Traditional Collaborative Filtering Recommendation Systems try to look for patterns and similarities between users - however, in a social network users are already connected together in a variety of ways and this information can be used to enhance recommendation results. Another big issue with current Recommendation Systems is that users interests and taste changes with time - and social networks are the first and most active location for that to be noticed. For example, the proposed algorithm took into account the number of events a user was attending. Furthermore current Recommendation Systems are limited to the realm they operate in - for example, an e-commerce platform can base the recommendations only on purchases made by users or products browsed by users. On the other hand currently developers have at their disposal data from a huge palette of social networks and the data acquired from those networks can complement each other, creating a very precise and very up to date profile of the user that can be used to suggest various content - the same versatile Recommendation System could be used for ads, fast moving consumer goods, culture, travel, suggesting other users and many other use cases. Finally, current Recommendation Systems very often face the problem of very sparse data sets - for example, an e-commerce platform has a big catalog of products but few purchases. Especially recommending anything to a new user is virtually impossible. The use of data gathered through social networks solves this problem as the user profile is already detailed.

7 Summary

The aim of this paper was to analyze the currently known and used content suggestion mechanisms and then create a new one based on data gathered from Facebook, in particular suggesting music bands and musicians based on user interests. Additionally, because of the suggestive nature of such suggestions, a verification method had to be introduced. All of these were accomplished.

Facebook user data was gathered through the official API, based on Facebook Graph and FQL (Facebook Query Language). This data was then used to create an interest graph that became the foundation of a suggestion mechanism. To verify the results, two highly popular data sources were used for comparison - Amazon.com *Customers Who Bought This Product Also Bought...* and Last.fm *Similar Artists*. Additionally, the developed system was tested against a popular open source solution, Oryx. The results are promising and suggest that social data can be a valuable source of information when suggesting content.

The unstoppable propagation and growth of the Internet and new social networks emerging constantly indicate that this area of data exploration and analysis is worth further research.

8 Bibliography

- [1] Rory Cellan-Jones, Technology correspondent, BBC News *Hackers and hippies: The origins of social networking* (<http://www.bbc.co.uk/news/technology-12224588>), 2011 [downloaded: 30.09.2014]
- [2] Ben Mezrich *The Accidental Billionaires: The Founding of Facebook, A Tale of Sex, Money, Genius, and Betrayal*, 2009
- [3] Daniel Lemire, Anna Maclachlan *Slope One Predictors for Online Rating-Based Collaborative Filtering* (http://lemire.me/fr/documents/publications/lemiremaclachlan_sdm05.pdf), 2005 [downloaded: 30.09.2014]
- [4] *Facebook Newsroom - Key Facts* (<http://newsroom.fb.com/Key-Facts>) [downloaded: 30.09.2014]
- [5] Francesco Ricci, Lior Rokach and Bracha Shapira *Introduction to Recommender Systems Handbook* (<http://www.inf.unibz.it/ricci/papers/intro-rec-sys-handbook.pdf>), 2011 [downloaded: 30.09.2014]
- [6] Greg Linden, Brent Smith, and Jeremy York, Amazon.com *Amazon.com Recommendations: Item-to-Item Collaborative Filtering* (<http://dl.acm.org/citation.cfm?id=642471>), 2003 [downloaded: 30.09.2014]
- [7] *Facebook Query Language (FQL) Reference* (<https://developers.facebook.com/docs/reference/fql/>) [downloaded: 30.09.2014]
- [8] *What is Apache Hadoop?* (<http://hadoop.apache.org>) [downloaded: 30.09.2014]
- [9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung *The Google File System* (<http://research.google.com/archive/gfs.html>), 2003 [downloaded: 30.09.2014]
- [10] Jeffrey Dean and Sanjay Ghemawat *MapReduce: Simplified Data Processing on Large Clusters* (<http://research.google.com/archive/mapreduce.html>), 2004 [downloaded: 30.09.2014]

- [11] Sean Owen *Myrrix Joins Cloudera to Bring "Big Learning" to Hadoop* (<http://blog.cloudera.com/blog/2013/07/myrrix-joins-cloudera-to-bring-big-learning-to-hadoop/>), 2013 [downloaded: 30.09.2014]
- [12] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan *Large-scale Parallel Collaborative Filtering for the Netflix Prize* ([http://www.hpl.hp.com/personal/Robert_Schreiber/papers/2008AAIMNetflix/netflix_aaim08\(submitted\).pdf](http://www.hpl.hp.com/personal/Robert_Schreiber/papers/2008AAIMNetflix/netflix_aaim08(submitted).pdf)), 2008 [downloaded: 30.09.2014]
- [13] Yifan Hu, Yehuda Koren and Chris Volinsky *Collaborative Filtering for Implicit Feedback Datasets* (<http://labs.yahoo.com/files/HuKorenVolinsky-ICDM08.pdf>), 2008 [downloaded: 30.09.2014]
- [14] Jelena Grujic *Movies Recommendation Networks as Bipartite Graphs* (<http://www.scl.rs/papers/2008-LNCS5102-576.pdf>), 2008 [downloaded: 30.09.2014]
- [15] Douglas Hebenthal, Cesare Saretto, Kathleen Mulcahy, James Allard *Following online social behavior to enhance search experience* (<http://appft.uspto.gov/>), 2012 [downloaded: 30.09.2014]
- [16] *What is Apache Mahout?* (<http://mahout.apache.org>) [downloaded: 30.09.2014]
- [17] *Powered by Mahout* (<http://mahout.apache.org/general/powered-by-mahout.html>) [downloaded: 30.09.2014]