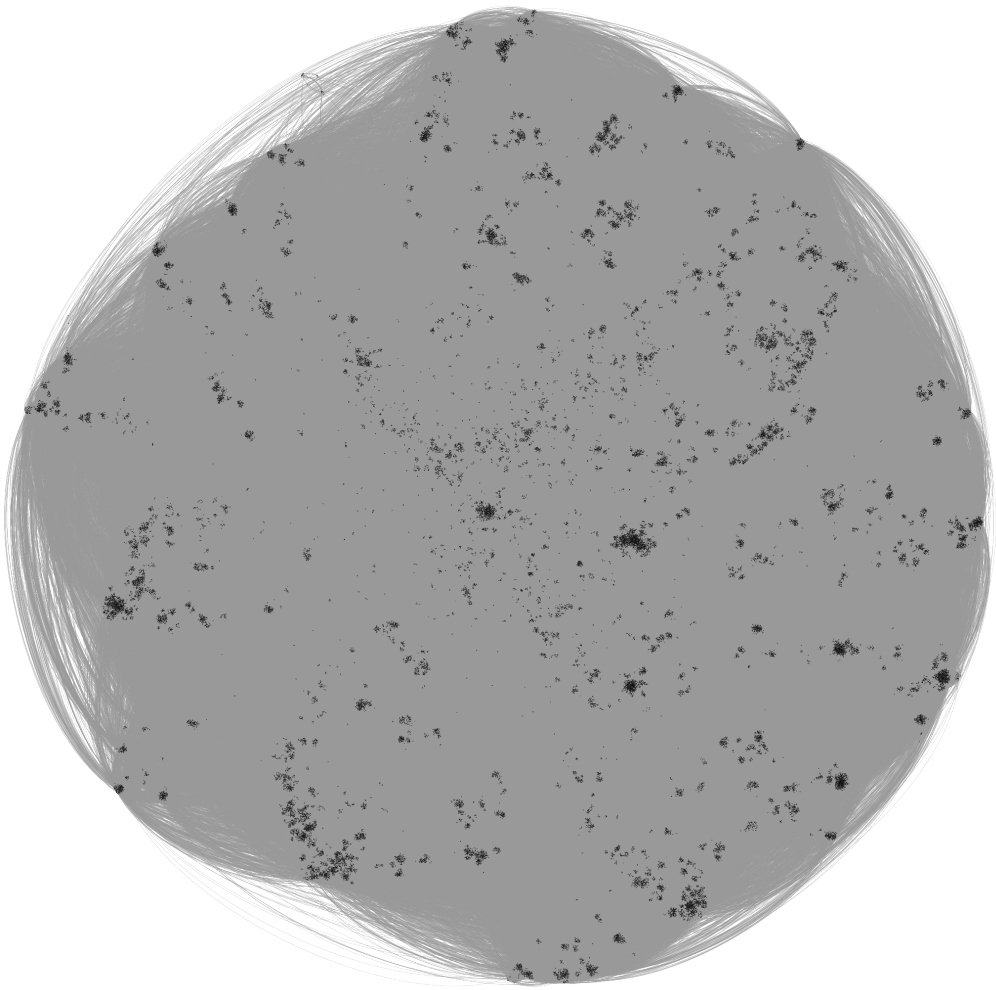
User-to-User Recommendations in a Fashion Portal, Utilizing a Social Network, Implicit Feedback and Clustering Approaches



Master Thesis, Aalborg University

June 3rd 2015

AALBORG UNIVERSITY
STUDENTS STUDYREPORT

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9220 Aalborg Øst
Phone no.: 99 40 99 40
Homepage: http://cs.aau.dk

Title: User-to-User Recommendations in a Fashion Portal, Utilizing a Social Network, Implicit Feedback and Clustering Approaches

Theme:

Recommendation Systems

Project period:
10th Semester (MI)
Spring semester 2015

Project group:
mi102f15

Group members:
Bjarne Kock

Dan Duus Thøisen

Davide Frazzetto

Supervisor:
Thomas D. Nielsen

Pages: 108

Finished on June 3rd 2015

Synopsis:

Abstract

In this report we address the social link prediction problem, applying our research to design a recommender system for the Sobazaar social network. This domain shows several interesting characteristics. Firstly, differently from most recommender system scenarios, solely Implicit Feedback is available. Secondly, the evolution of the fashion trends causes complex social dynamics to appear in the structures of the network, therefore finding communities of users with similar characteristics is crucial to understand the users' behavior. We conclude that by combining clustering with information that can be extracted from the users' activity it is possible to positively contribute to the social link problem.

We would like to express our gratitude to our supervisor Thomas D. Nielsen for his remarks, useful comments and engagement throughout master thesis. We would like to thank our families and friends for their kind support during this challenging period of our life.

Resume

In this report we have worked with data gathered from Sobazaar, an online fashion portal which caters to primarily women. Sobazaar gathers women wear from webshops such as H&M and Vero Moda and scrapes their products to preview them in a structured way on their own web portal. Sobazaar can be accessed from an iPhone via an application. The users of Sobazaar can browse through different products, and like the products, or create labeled boards and publish them for their follower to see. Users can follow each other which results in them receiving the followed users' updates, in the sense of likes or new published boards. We investigate the underlying cause of one user following another, and how we can combine implicit feedback and a social graph in order to make user-to-user recommendations. To do so, we build several crawlers to gather the data from the public Sobazaar API, in the period between the 27th of February 2015 to the 10th of April 2015.

We give a complete overview and an analysis of the data gathered. In our analysis we find popular products and popular boards. We analyze how users behave in this system and what they like to share and we build a social network using the follow connections between the users. The social graph consists of $45,650$ nodes with $860,883$ directed connections. We use 4 different clustering approaches on the social network, the clustering algorithms are: *Social Network Clusterer, Voltage Clusterer, Authority Clusterer, and Spectral Clusterer* which try to detect communities in the social network or densely connected sub graphs. Clustering will also reduce the number of users we will have to consider, in order to make recommendations as it splits the graph into smaller sub graphs which means that the running time of our recommender can be improved. The recommender we have developed uses the clustering approaches and it is evaluated using the K-Fold cross validation method. We investigate which recommender to use by running experiments on the matrix factorization and K-NN approach. In the second experiment we investigate which perception feature performs the best. The features for the perception are: *Pagerank, Item Similarity, Common Neighbors, Activity, PageRank,* and *Adjacency*. In the final experiment we investigate which of the four clustering algorithms clusters the graph best, based on the quality of the recommendations created using that clustering.

Our results show that the best recommender is K-NN with the common neighbors feature as perception, and not using any clustering. The best clustering approach was voltage using 10 clusters. Using clusters improves the running time which makes the choice of using clusters a performance versus accuracy decision. We have found evidence for users following one another in the social graph and that people try to conform to their friends following common users as the users around them. We propose that the implicit feedback and social network structure can be combined to make user-to-user recommendation using a clustering approach.

Contents

Chapter 1

Introduction

Recommendation systems are widely used today in many different applications. They are used to create a highly personalized user experience, in either recommending movies, songs or other products the user has not seen before. Recommendations can increase user satisfaction and potentially increase profit. The purpose of a recommendation system is to predict a user's preference of some given product, before the user has ever seen nor purchased this product. In domains where it is unfeasible to browse through a vast amount of products, a recommendation system can filter out irrelevant products that the system has predicted the user not to like. In larger e-commerce websites such as Amazon.com that has millions of both physical and virtual products, it is impossible for the users to browse through all products. Amazon.com is using a recommendation system to present the users with products they most likely want to buy. The E-commerce industry wants to increase sales, and one of the ways this can be done is using recommendation systems. According to [26] there are three different ways of doing it: Converting browsing visitors into buying customers **(Browsers into buyers)**, recommending products which are related to a product currently being bought **(Cross-buy)**, and improving the relationship between user and store, such that the user gains more loyalty to the store **(Loyalty)**.

In our previous work [14] we presented a recommendation system for the online fashion portal Sobazaar. In the Sobazaar mobile application, users can browse through women wears, purses, jewelry, and accessories. While browsing users can express their preference by pressing a heart button, symbolizing that the user likes this product, create a board consisting of products they seem fit or follow other users and brands. Following brands and users will give the user updates from all their followings on the frontpage of the application. In recommender systems we differentiate between implicit and explicit feedback. Explicit feedback is when a user has the option to express both a negative or a positive feedback, for example the 1-5 star system in Netflix or the thumbs-up or thumbs-down on Youtube. The feedback Sobazaar receives from users is solely implicit. Implicit feedback can be used to extract a preference from a

user that can only be positive or negative and not a mixture. In other words, the users of Sobazaar has only a choice of giving a preference, or removing a preference after it has been given. The user cannot show aversion for a given product or another user. Implicit feedback does therefore not directly show the preference of a user which makes the tasks of constructing a recommendation system using implicit feedback especially interesting.

Equally interesting was it to process the amount of data we gathered. The recommender system we constructed in our previous report, was able to recommend items to users. This report will focus on recommending users to other users.

We expect you to be familiar with notation used in our previous report [14, Section 3.1.2].

1.1 Problem Definition

This report will explore the possible solutions to create user to user recommendations, in the online fashion portal Sobazaar, by solving the following problems:

Problem 1 What is the underlying cause of one user following another?

Problem 2 How can we make user-to-user recommendations using a social network utilizing implicit feedback and clustering techniques?

1.2 Goals

In this project we want to achieve the following goals

- Solve the problems stated in the problem statement.
- Contribute to the current research done in the area of recommendation systems more specifically on the use of social graphs and implicit feedback in a fashion portal context.

1.3 Discussion

Sobazaar contains mainly women wear, bags and other accessories which the system allows their users to interact with. The users can also interact with each other, they can follow other users, browse products, give likes to products and construct labeled boards they can share among followers. Much of this is like a traditional social network such as facebook and twitter although the users are limited to interact only with products Sobazaar provides. This is uncommon for a social network to provide the content the users are allowed to share and like.

Finding the underlying cause of users following one another is therefore an interesting problem not only because it allows to recommend relevant users, but also in identifying how the behavior differs from conventional social networks and how users interact in a fashion domain.

The type of data used in this report is implicit feedback, and we cannot for this reason directly identify what a user prefers and as an effect we only have partial knowledge of which products or users she has visited. It is therefore interesting to see if this data can contribute to find users which are relevant to recommend and how it could be used in conjunction with a social network.

When considering a recommendation, an intuitive choice is to look at similarities between users depending on similar user preference of products. We assume that people trust their friends when it comes to choices, asking friends for recommendations. If this is true we can utilize the social structure of the application to find the connections between users. Finding their network of trusted users allows for additional information which can be incorporated in the recommender system.

In a social fashion application users' activity is a balance of both personal interest and conformity. The users' activity is publicly available, and the need of belonging to a certain group or to show only certain aspect of herself is central in the social activity. Relevant research on the topic has already been conducted in [33], where it has been shown that users change their preferences to conform to others. In our application it will be important to understand what the balance of these two components are, in the users' decisions. Our analysis of the difference between the *purchase* and *like* activity in [14] supports this hypothesis. We saw that there was a difference in the users activity of the two actions, the purchase was private and like was public, this resulted in users having different items as targets of their private activity. A consequence to this relevance of an item for a subject might depend more on her social connections than on the items she has purchased. Because of this, an interesting point would be to identify these communities and group users together based on the current trend in that community.

We conclude that a clustering approach might be able to find these communities which are reflected both in the social network and in the items which the users liked and/or put into boards. The result of recommending users correctly is twofold. Firstly, recommending new connections will supposedly lead to a higher user activity.

Secondly, we argue that an accurate user recommendation will have a positive side effect to the recommendation of relevant items to users. This assumption is made on the basis of how the Sobazaar application works. The public activity of a user is published on the front page of each of the followers of this user. If a recommended user shares preferences with her new followers, these public feeds will find their interests, hence a higher probability for them to interact with the items. One can argue that the user feeds become a recommendation, hence the followed user is unknowingly recommending items to

the target.

1.4 Structure of Report

Chapter 2 contains a description of the Sobazaar application, an overview of the data, how it was retrieved and an analysis of both the implicit feedback and the social graph.

Chapter 3 gives an overview of the current state of the art techniques in the area of recommendation systems and clustering techniques.

Chapter 4 outlines all our experiment pipelines and shows the approaches we have taken, which algorithms we have chosen and the evaluation of the recommender we have built.

In Chapter 5 we show the experiments we conducted and make comparisons of different algorithms we have chosen.

Chapter 6 makes a conclusion on the results and the discussion about the results and also discusses which additional research could be done.

Chapter 2

Data Analysis

This chapter gives an overview and analysis of the data collected from the Sobazaar API. Section 2.1 describes the Sobazaar application, Section 2.2 gives a brief overview and explains the structure of the data and defines the terminology used. Section 2.3 describes the metrics of the data and show statistics of the occurrences of certain events. Section 2.4 briefly explains how the social graph is generated and presents an analysis. Section 2.5 gives a detailed analysis of how clustering algorithms can be applied to the social graph.

2.1 Overview of Sobazaar Application

Sobazaar is a mobile application [app] for the iPhone and was previously also a web portal which is now closed. Sobazaar collects women wears from online retailers such as H&M or Vero Moda, and displays them as seen in Figure 2.1. When opening the app the user is presented with her feed, which contains updates from brands or users she follows as shown in Figure A.2b.

Users of the application can then select items and visit their product details as shown in Figure A.2a. The product detail shows the current price of the item, how many likes it has received, allows the user to purchase the item and like it herself as well as gain more information about the item.

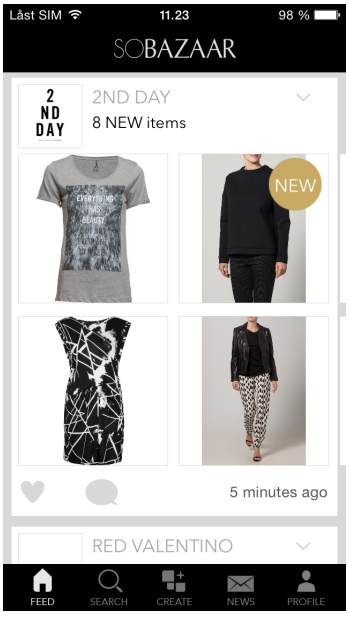


Figure 2.1: Sobazaar App for the iPhone. Taken from [14].

An additional function in the Sobazaar application is the creation of boards. Figure A.1a shows the interface where the user can choose the layout of the board shown at the bottom of the phone and the items she selects in the top part. Figure A.1b shows the board in its published state. The author is seen at the top of the board, while comments are made on the bottom.

Users in the Sobazaar system have profiles, Figure A.3 shows a profile, which displays the items they have liked and the boards they have created it also displays the number of users which she is following and being followed by. When following a user, the followed user's boards and likes will appear on the feed of the following user. The *follow* system of Sobazaar can be compared to the subscription system of YouTube. On YouTube you subscribe to a channel if you like what the channel provides and you want to be notified when this channel uploads new videos. The same happens on Sobazaar when a user follows another user. The activity of the followed user is visible to the user who follows.

2.2 Data Overview

The data used in this research has been gathered from the public online API of the fashion portal Sobazaar. We have developed four webcrawlers that have extracted the data from this API in the time period between 27th of February 2015 and 10th of April 2015. Each of the crawlers has been used to obtain a different category of data, e.g. users, items, boards, etc. A database design of the datasets we have available can be seen in Figure 2.2. The data has been saved into a MySQL database, with the appropriate keys and foreign keys.

Initially we have collected a dataset of users we have obtained by requesting the API for sequentially generated user ids, obtaining those users associated to a valid id. This process required a lot of time as the user ids are not contiguous. The extracted user ids has been used as initial seed for the crawlers. Like a web crawler moving through page links, our implementation is crawling through the user connections.

If we consider a user u, the users that are following u are called followers, while *followings* are users u is following. When crawling the user u we retrieve her followers and followings. For each of these users v we check if the pairs (u, v) or (v, u) are already present in the following table of the database. If for example (u, v) is not in the database, we add this as a new entry into the database, associated with a timestamp of when the crawler has encountered this new following/follower. This timestamp is not the time of when the user u followed the user v.

An *unfollow* is defined as a user u that has followed a user v at a timestamp t and removed the follow connection to the same user at a timestamp t', where $t' > t$. This *unfollow* is then added to the database. If u starts following v again, and the crawler encounters this event, the entry will be added again.

An unlike is added to the database in the same way as the unfollow. If a

user likes a product, and then the crawler detects the user does not like it any more, it will be added as an unlike. Notice that this is not a dislike, this is only indicating that a user has removed the preference of a like.

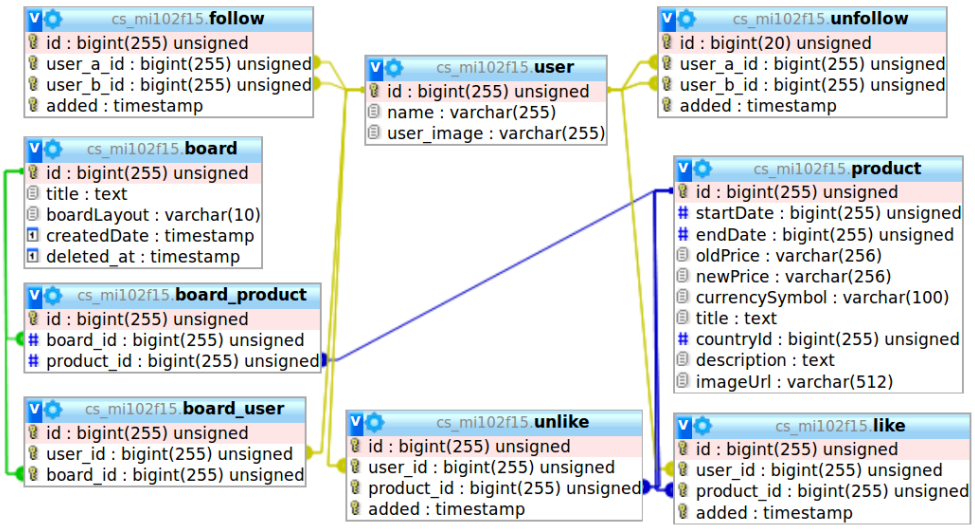


Figure 2.2: Database Design

In Figure 2.2 we show our database design, consisting of tables: follow, board, board_product, board_user, user, unlike, unfollow, product and like. A connection from a column inside a table symbolizes a foreign key. Follow and unfollow both have user_a_id and user_b_id which reference the user table. Follow, unfollow, unlike and like all have an *added* column which is a timestamp of when the crawler added this entry to the database. Board has a createdDate which is a timestamp gathered from the API that states when the board was created by the user, and a deleted_at which is a timestamp the crawler is adding, if it detects that the board does not exist anymore. The tables board_product and board_user are both pivot tables, and are used to store the foreign keys of boards, the users creating the boards, board and the products inside.

In Table 2.1 we list a count of all the tables in the database. Notice here that inside the follow and unfollow table there can exist duplicates with a different timestamp. If a user has followed, and unfollowed another user multiple times, these will end up being duplicates with different timestamps.

2.3 Preliminary Analysis

In this section we discuss the data we retrieved from the Sobazaar API, more specifically: boards, likes, and follows.

Table	Count(*)
board	46407
board_product	203498
board_user	46377
follow	860883
like	601721
product	35470
unfollow	4495
unlike	165
user	45650

Table 2.1: Database table overview with count for each table

2.3.1 Boards

In the Sobazaar system users can publish labeled displays of products called boards. To create a board the user chooses a board design, which decides how many products the board can contain. The user then adds her products to the board, assigns a name to the board, and publishes the board. When the board is published, all her followers can see this board on their feed. We have 46,407 boards in total, with one user having the maximum of 432 boards published. An average of boards published of 6.7822, only considering users with at least one published board. We have 35,470 products where 20,050 of them are present in at least one board.

Number of products	Count of boards
1	283
2	519
3	6022
4	26685
5	17
6	12881

Table 2.2: Number of boards containing 1-6 products

In the Table 2.2 we see a count of how many products are inside of the boards. The *Number of products* is in the interval from 1 to 6 products respectively. Where the *Count of boards* are how many boards consist of these 1 up to 6 products. The first line gives us the information that there exist 283 boards which only contains one product and 17 boards with 5 products.

The product with the id 95118006 has been used in 1992 boards and is the most used product.

There are products being used in boards multiple times by the same user. We will list the top 5 products, which can be seen in Table 2.3.

In Table 2.3 we can see that the users find particular products and com-

product_id	user_id	present in # boards	Product image
17388368	120948003	31	
383868025	175788022	27	
359488006	167908001	26	
115638018	186538038	26	
209118003	169328013	24	

Table 2.3: Products that are present in multiple boards by one user

bines them with different outfits. Because the same product has been used in multiple boards.

product_id	# boards	Product image
95118006	1992	
1918030	832	
106808093	790	
127388010	766	
45258004	548	

Table 2.4: Products and the number of boards they appear in

In Table 2.4 we see the products and the number of boards they appear in. These products are the most popular products used inside boards.

2.3.2 Highly Popular Boards

We are finding highly popular products by counting the number of times each product has been used in boards. Then by assigning this popularity value to each product, we sum the popularity value for each board's products, to find the most popular boards.

We believe there can be many reasons for why a user created a highly popular board. One can be that the user finds all the products inside this board appealing, or that the products inside this board seem fit with many outfits.

2.3.3 Follow and Unfollow

Every user inside the Sobazaar system can indicate a relationship with another user, by pressing a follow button. Users can follow multiple people, and be followed by multiple people. If a user follows another user, and later decides to stop following this user, we call this an *unfollow*.

Via the Sobazaar API we gathered the follow connections, by asking for a particular user's followings, and the API returned a JSON string with all her followings. Taking this list and saving each following user in our own local database, and adding a timestamp of when this entry was added to the database. This timestamp is being used to identify changes to the current state, for example if a user unfollows another user at a later point in time.

We have in total $860,883$ follow connections and in total $4,495$ unfollows.

2.3.4 Like and Unlike

When a user is browsing the Sobazaar IOS application or the website, and is finding products she prefers, she can press a little heart button to indicate a preference for this product. This heart button is called a *like* button. We have built a crawler that request the Sobazaar API for a user's like list, which retrieves a JSON string consisting of all the products this user likes. In total we have $601,721$ likes where only $11,413$ users have one or more likes. We have 165 unlikes, and an unlike is gathered when the crawler, by retrieving the like list, checks our database for an already existing like. If the like existed, and is not present in the retrieved list any more, it is being considered an unlike, and saved in the database as such.

We have an average count of likes per user of 13.18. The average like of the users who has at least one like is 52.7224 and a median of 18, where the most active users and their like count is shown in Table 2.5.

As expected we have users that are way more active than others, and digging deeper into these highly active users. To eliminate any bias from our data, such as a developer testing out features which could result in many likes, boards published or many followings. We looked up some of the highly active users, on other social media websites, and found that some of them do in fact work for Sobazaar as a fashion blogger, and since this is not unusual behavior, i.e. a fashion blogger working for a fashion portal is expected, and should be included in a recommendation system also. So since this is real events from a human being, and not a developer testing features, we chose to keep the data as is.

In Table 2.6 we see the top 5 highly popular products via a like count, listed in a descending order. Comparing the Table 2.4 and Table 2.6 we can see that the product_ids 95118006, 1918030 and 106808093 is occurring in both lists.

user_id	like count
177508024	5712
5002	3468
303818007	2817
7268020	2661
141518010	2090

Table 2.5: Top 5 user ids with their like count

product_id	like count	Product image
95118006	1457	
106808093	1018	
210328037	967	
1918030	931	
94808034	891	

Table 2.6: Top 5 products with like count

2.4 Social Graph Analysis

In Section 2.3.3 we mentioned that users and their connection with other users is called a follow connection. We also gathered if a connection between two users has been disconnected by one of the users, called an unfollow.

We have constructed a social graph utilizing the users following and follower connection with each other.

2.4.1 Graph Definition and Notation

A social network can be described as a graph $G = (V, E)$, defined by a set of vertices $V = \{v_1, v_2, ..., v_n\}$ and a set of directed edges $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V\}$. An edge in E represents an action that connected two users associated with the time in which this interaction took place $t(v_i, v_j)$.

The social graph can be constructed from the log of the users' activity. In the Sobazaar application a user v_i can follow another user v_j at time t. This creates the edge $(v_i, v_j)_t$ between the two vertices in the social graph. If at a time point t' the user v_i unfollows v_j then the edge $(v_i, v_j)_t$ is removed from the graph.

2.4.2 Generating the Social Graph

The SQL query to gather the data used to construct this social graph can be seen in the appendix Listing B.1.

The construction of the graph, runs through all users and adds them as a vertex in the graph, and connects all vertices, with a directed edge, where a follow connection is made.

In Table 2.7 we present some of the characteristics of the social network.

The density describes the percentage of edges in a graph $G = (V, E)$, defined as

$$\text{density} = \frac{|E|}{|V| \cdot (|V| - 1)}$$

The denominator is the number of possible edges in the graph, the numerator the number of edges present in the graph. With a density of 0.0004% we can see how the Sobazaar social network is extremely sparse: only $856,388$ edges are present out of the $2,083,876,850$ possible edges.

The mutuality describes the percentage of pairs of nodes (u, v) that are connected in both directions. Formally it is defined as

$$\text{mutuality} = \frac{\sum_{u,v \in G}((u, v)|(u, v) \in E \wedge (v, u) \in E)}{|E|}$$

where the denominator is all the possible pairs of nodes, and in the numerator we are counting those for which both edges exist. As we can see in Table 2.7 the mutuality coefficient is 0.44485%. This shows how the relationships in Sobazaar are most probably not related to actual friendship, as it

could be in other type of social networks, but more subject to trends or taste similarity.

The number of nodes for which we have a like lists is $11,413$, which is only 26.9% of all nodes in the graph. This means that we can only partially use the like lists in our models.

In the next sections we will investigate the nature of the social network, analyzing how the users are connected, what the relationship between their connections are, and how the general structure of the graph can help our objective of improving a social link recommender system.

number of nodes in the network	45,650
number of disconnected nodes	3,222
number of edges	856,388
average degree	20.1849
density	0.00041
mutuality	0.44485%

Table 2.7: Overview of the social network's characteristics

2.4.3 Structure of the Graph

We start the analysis of the social network by investigating the structure of the graph. The first aspect of the social network we want to describe is the general connectivity of the graph, considering the existence of different separated components in the graph. Given a graph $G = (V, E)$, a component is a subgraph, such that each pair of nodes in the subgraph is connected by a path, and these nodes in the subgraph, are not connected to other nodes of the graph. Since the social network is a directed graph, the definition of component that would naturally apply is the strongly connected component, defined as

$$G' = \{v \in G \quad | \quad \forall u \in \{G \setminus v\} : \exists v \rightsquigarrow u\}$$

In the case of a social graph we can not expect to find large strongly connected components. In our case we are more interested in understanding how the nodes are connected, independently from the direction of the connections. Therefore, if we consider that the existence of an edge between two nodes represents a connection, we can translate the directed graph into an undirected graph. A graph is weakly connected if by replacing all the edges with undirected edges, the graph is strongly connected.

We have translated our social network into an undirected graph and started a breadth first search from a random node, adding the nodes that are found by BFS in the same component. The result is that all the nodes of the graph are discovered by one single run of BFS, meaning that all nodes with at least one incident edge are all part of the same weakly connected component. At first this could seem like an interesting result, showing that all the users are

part of the same undirected graph, if we do not consider the direction of their connections. It is important though to recall the way we have constructed the dataset; since we do not have access to all the users, we have designed the crawlers to move from user to user following their connections. Even though we have initialized the crawlers with a set of seed users, as described in Section 2.2, it is highly probable that the original social network of the Sobazaar application contains more users. Therefore, we can not guarantee that the original social network consists of only one weakly connected component. If this is the case, then what we have retrieved is only part of the social network and we can not make any hypothesis on the size of the original platform. Nonetheless, we will now study the interactions between users and the particularities of the graph which can be extracted from our dataset, to confirm that it is still representative of the original network.

2.4.4 Degree of Nodes

From this point of view, an important aspect of the social network is the degree distribution, namely how many inner and outer edges each node has. In Figure 2.3 it is possible to see how most of the users are following less than 100 users, and the highest percentage of users have less than 100 followers. By extracting the users that have more than 200 edges, we have seen that they only account for 53, out of the 45,650 users of our data set.

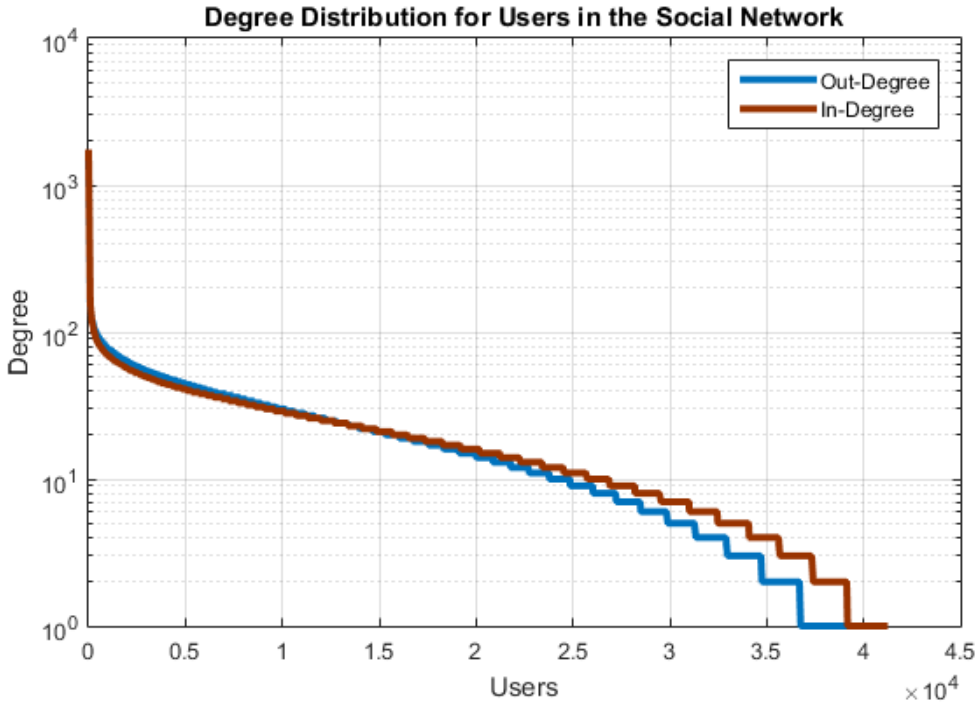


Figure 2.3: In and Out degree distributions of the network, in a logarithmic scale

2.4.5 Scale Free Network

An interesting type of network is the *scale-free* network. A common feature in real world networks is the presence of *hubs* and *authorities*. v_{Δ^+} describes the indegree of node v while v_{Δ^-} describes the outdegree of node v. While authorities are nodes with a higher indegree than most of the nodes of the network, hubs are nodes with an outdegree above the average. The property of this can be seen from the long tail of the distribution we have shown in Figure 2.3. The users in this tail, i.e. the right part of the distribution, have a lower degree compared to those in the head, the leftmost section. This long tail behavior confirms that our social network follows the definition of a scale free network.

Definition 1. *Given a directed graph $G = (V, E)$ a set of nodes $A \subset V$ of size k is defined as the set of authorities if*

$$\forall a \in A, \forall v \in V \setminus A,\ a_{\Delta^+} > v_{\Delta^+}$$

Definition 2. *Given a directed graph $G = (V, E)$ a set of nodes $A \subset V$ of size k is defined as the set of hubs if*

$$\forall a \in A, \forall v \in V \setminus A,\ a_{\Delta^-} > v_{\Delta^-}$$

The main consequence of this property is that these authorities function as centers of connections, on which the other nodes are aggregated. In our case this is an important feature, since the presence of these authorities indicates that the users tend to connect with the very popular users of the network, while the degree of connection among the other users is fairly sparse. This is important for two reasons. Firstly, the possibility to identify and utilize these authorities to find communities in the network. Secondly, it shows that it is possible to recommend new links to users inside these communities. Since most of the connections only address the authorities the density between the other nodes is low, theoretically leaving space to new recommendations among them.

We have extracted the names associated to the IDs of these authorities. Afterwards, we have tried to find their identities by looking at different public social networks, and we have discovered that some of them are Sobazaar employees, users we could describe as fashion blogger which role we assume is to publish content. These users are probably famous to the users in the application, and this ulteriorly confirms our hypothesis that users that are important for fashion purposes are also those that are highly connected.

2.4.6 Reasons for Following

We wish to investigate the reasons for why one user follows another since these reasons ultimately are required in order to make successful recommendations. We therefore investigate whether we can find a relation between the implicit

feedback and a users followers and followings. We take a baseline for the similarity based on the implicit feedback, namely the likes, by creating every possible edge between all users for whom we have like lists.

Metric	Outgoing Edges	All Pairs
Mean	0.1142	0.0631
Standard Deviation	0.1387	0.0884
Similarity above zero	51.48%	19.08%

Table 2.8: Statistics of the similarity based on the like lists.

Table 2.8 shows the statistics of all pairs of outgoing edges for which we have like lists. It can be seen that over 51% of the pair of nodes there exists similarity and this similarity is around 0.11. To compare we added the same metric for all possible pairs for which we have like lists, which is much lower, around 19% and the mean similarity is also lower 0.6. The distribution of these two subsets can be seen in Figure 2.5 and Figure 2.4.

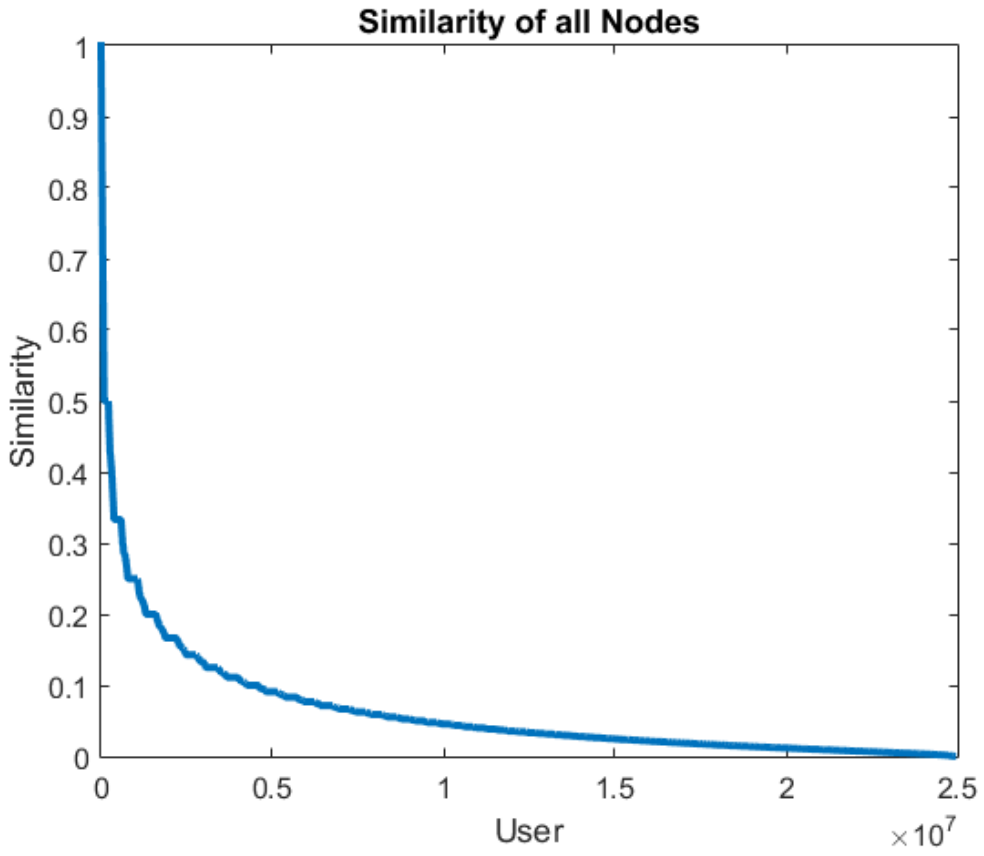


Figure 2.4: Similarity of all nodes with a like list, zero entries omitted

Figure 2.4 shows the distribution of similarity of the subset of node pairs which have like lists and which have similarity greater than zero.

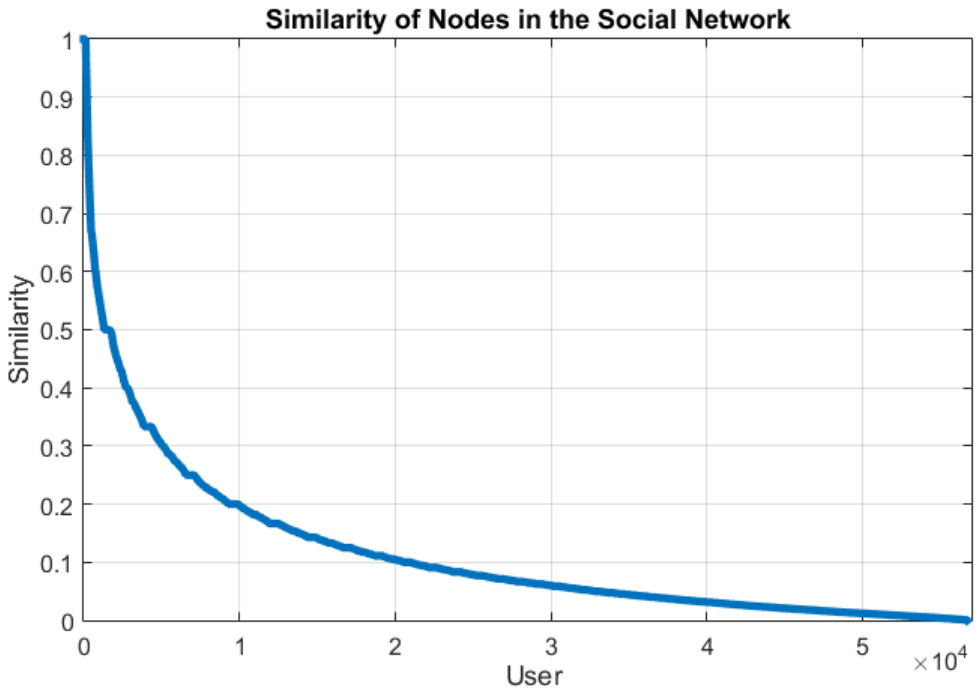


Figure 2.5: Similarity between nodes and their outgoing edge target nodes, zero entries omitted

The Figure 2.5 shows the similarity of nodes based on their like list for all node pairs that have an outgoing edge. The user pairs with zero similarity have been omitted for clarity.

The similarity is calculated as follows:

$$\frac{|L_u \cap L_v|}{|L_u|} \text{ for } |L_u| \wedge |L_v| > 0 \tag{2.1}$$

where u, v are both users and L_u, L_v are like lists of user u and v respectively.

The numbers in the Table 2.8 clearly indicate that there exists a relation between the like list and social connections. An interesting question is now whether the similarity causes the *following* or whether the *following* causes the similarity. Currently there only exists evidence for the latter; we know that as soon as a user follows another user, she will receive likes and boards from that user on her feed, which suggests that she will have opportunity to like the same items as the user she follows, thereby increasing similarity.

Activity

The activity of a user might give information about her popularity in the social graph or the popularity might indicate high activity. User's likes and boards get published to the feed of all their followers, this means that an active user would intuitively seem to be preferred as this would give additional content for the user following.

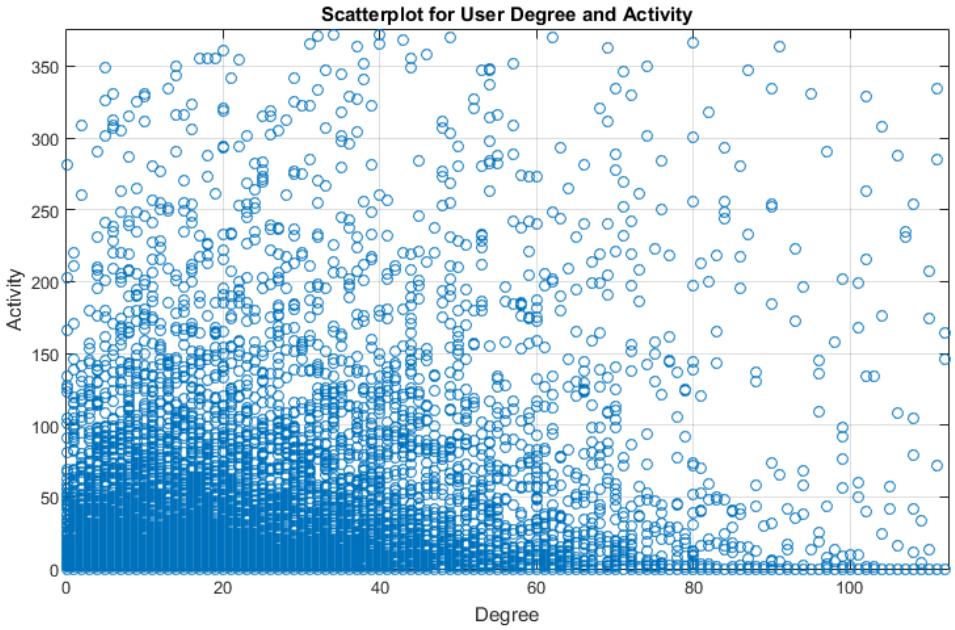


Figure 2.6: A scatter plot of the indegree on the x-axis and activity on the y-axis of users in the Sobazaar system.

We therefore plotted the degree and activity of each user for which we have activity, Figure 2.6. The activity is generated by taking the number of likes and the number of published boards and adding them. Figure 2.6 only refers to the indegree of the users, meaning their followers. The figure shows that there is no correlation between a user following another user and their activity.

We can therefore conclude that the activity by itself is not a reliable metric for predicting popularity.

2.5 Clustering the Social Network

We now focus our analysis on investigating if, and how, it is possible to find clusters of users in the social network. In Section 1.3 we have introduced how, by clustering the social graph, it could be possible to identify communities of users, fashion trends, understand why people connect to each others, and in

general, to improve the performance of the recommender system, as we will discuss in more detail in Section 4.2 and Section 5.1.3. We analyze different clustering techniques, which algorithms' details we describe in Section 3.5. A preliminary consideration before showing our analysis is to fully understand the mechanisms that lead to the creation and evolution of these clusters. We should have access to the time data for each of the events in the social network, therefore, our analysis is not fully representative of the real behavior of the users in the Sobazaar application, but we still believe we could obtain some useful insights for our research. In order to cope with this lack of information, we perform our clustering analysis only on a sub graph of the entire social network, and we will compare how the clusters obtained from this sub graph adapt when considering the whole graph, in exactly the same way shown in Section 4.2.5.

In Table 2.9 we present the clustering algorithms we have tested, with all the associated parameters. For each of them we have tried different number of clusters, $5, 20, 30$ and 50. The only difference is Voltage Clustering, for which we can only specify the maximum number of clusters that should be generated, then the algorithm finds the number of clusters it considers appropriate accordingly to the set threshold and likewise with the Social Network Clusterer that takes as input an α and a β value that the clusters obey.

With the previous number of clusters set as limits, the number of clusters that Voltage Clustering obtained are $4, 13, 16$ and 22. For Spectral Clustering we use three different affinity matrices, depending on three different types of distance functions: Adjacency, Item similarity and Common Neighbors similarity, that we define in Section 4.2.2.

The Social Network Clustering algorithm's α and β parameters have been set to 0.01 and 0.009 respectively, after several attempts to find the right parameters that could fit our graph.

Algorithm	Parameters	Number of Clusters
Authority Clustering	/	5,20,30,50
Spectral Clustering	Adjacency, Item Similarity, Common Neighbor	5,20,30,50
Social Network Clustering	$\alpha = 0.01, \beta = 0.009$	/
Voltage Clustering	/	4,13,16,22

Table 2.9: List of the clustering algorithms with the associated parameters

2.5.1 Size of the Clusters

We look at the size of the clusters generated by each clustering algorithm.

We do not show the size of the clusters when the number of clusters are 50 for lack of space, the behavior is similar to the one shown in Figure 2.9.

Figure 2.7: Size of the clusters and outliers when number of cluster = 5.

We can see that when increasing the number of requested clusters the clusters found by the Spectral Clustering change. Both Voltage Clustering and Authority Clustering find instead the same clusters with almost the same number of nodes, while generating other very small clusters with less than 100 nodes. Spectral Clustering instead finds completely different clusters, equally distributing the nodes over all the clusters. For example, in Figure 2.7 Authority Clustering finds that most of the nodes are in cluster 1 and 2, containing together approximately 75% of the nodes in the graph, and continues to keep these two clusters in Figure 2.8, now named as clusters 1 and 12. Spectral Clustering instead, independently from the parameters used, changes the size of all the clusters, decreasing from an average of 7500 nodes in Figure 2.7 to 1200 in Figure 2.9.

Finally, for each clustering algorithm the graphs show the number of outliers, i.e. those nodes that the algorithm can not associate to any cluster.

Because of the workings of Authority Clustering, defined in Section 3.5.4, all the nodes are associated to the closest *authority*, hence no outliers are found by this algorithm. Spectral Clustering finds instead the same number of outliers for all the number of clusters, average 8800 when using the adjacency matrix and 6285 with the Common Neighbors similarity matrix. Voltage clustering considers 12956 nodes as outliers when only 4 clusters are retrieved, 2895 for 13 clusters and 4229 for 16 clusters.

We can conclude that, since these three completely different behaviors both

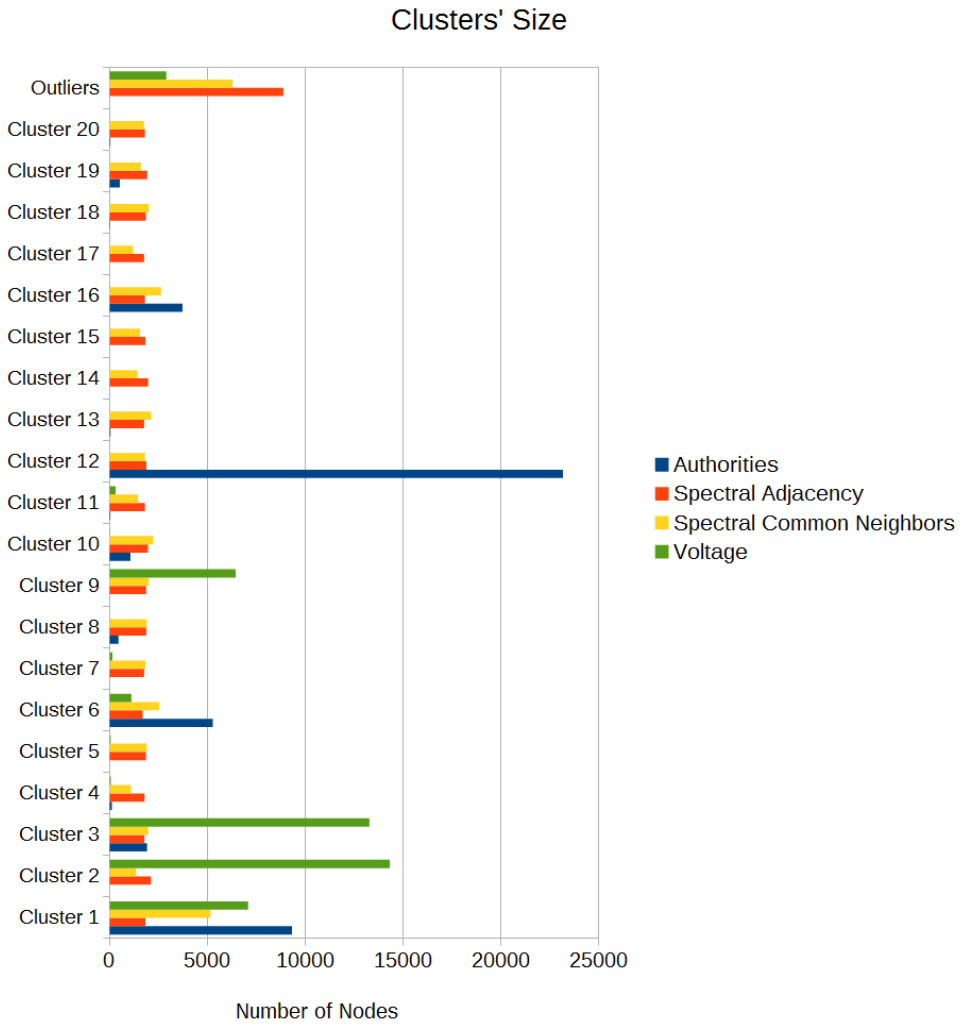


Figure 2.8: Size of the clusters and outliers when number of cluster = 20.

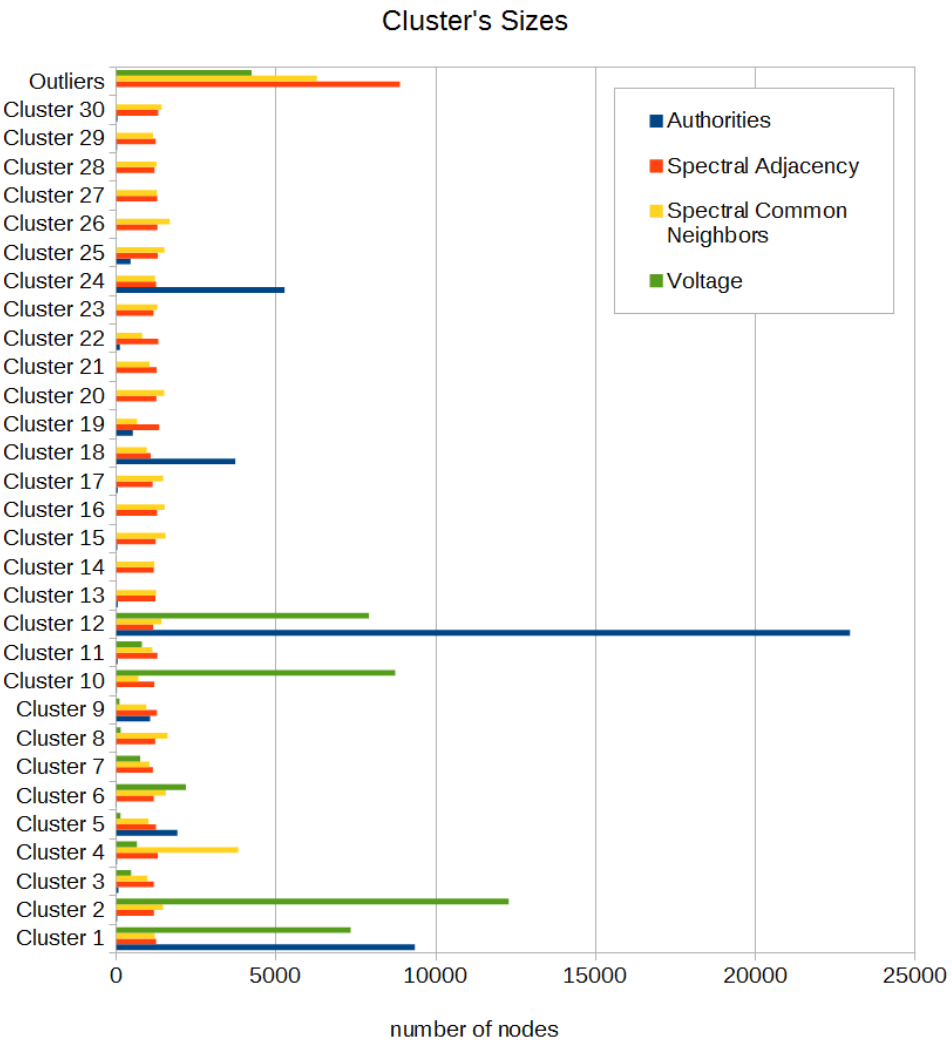


Figure 2.9: Size of the clusters and outliers when number of cluster = 30.

in terms of size of the clusters and outliers, it is not clear which algorithm is more appropriate in our data set, in order to correctly describe the structure of the communities or groups we are willing to find. We continue investigating the characteristics of these clusters.

2.5.2 Trends and Communities

Considering the fashion nature of the Sobazaar application, it is natural to wonder if users follow different trends and fashion groups, if users with similar taste connect to each others and if they create communities inside the social network where they share these interests. Assuming that such behavior is present in the social network, the question is if it is possible to detect these communities, and finding the underlying reasons which ties these users together. If the reasons are their interest in a particular group of items, we should see how the users inside these clusters like similar items and how these items differ between clusters and compared to the popular items in the entire social network.

In Figure 2.10 we analyze how the clusters find communities of users that more tightly share interests based on items. To do so, we calculate the average similarity of the user in a graph $G = (V, E)$ as

$$\frac{\sum_{(u,v)\in E} sim(u, v)}{|E|} \tag{2.2}$$

where $sim(u, v)$ is the similarity between user u and v for which there is a directed edge (u, v) in the graph. We calculate the similarity for the graph and find it to be 0.0045129. For each clustering algorithm, we then calculate the similarity inside the subgraphs obtained from the clusters. We can compare these values by dividing the similarity inside the cluster with the similarity in the graph. If the value of this ratio is greater than 1 we can conclude that we have restricted the cluster to those users that share similar interests in terms of the items they have interacted with and thereby found fashion trends inside these communities.

In Figure 2.10 we show the results where we have requested 20 clusters from each of the algorithms mentioned, and only analyze those clusters that contain at least 50 nodes.

All the clusters have a strictly positive ratio. Some of the clusters have users for which the average similarity is 500 times higher than in the original graph, and in particular Voltage Clustering obtains the overall higher results, meaning that it finds users that are connected and that also share interests. On the other hand, Spectral clustering does not perform well.

In Figure 2.11 we show how the most popular items in the communities differ from the items that are popular in the social network.

We consider the 10 items that have received the largest number of likes and that have been published in the highest number of boards as the most

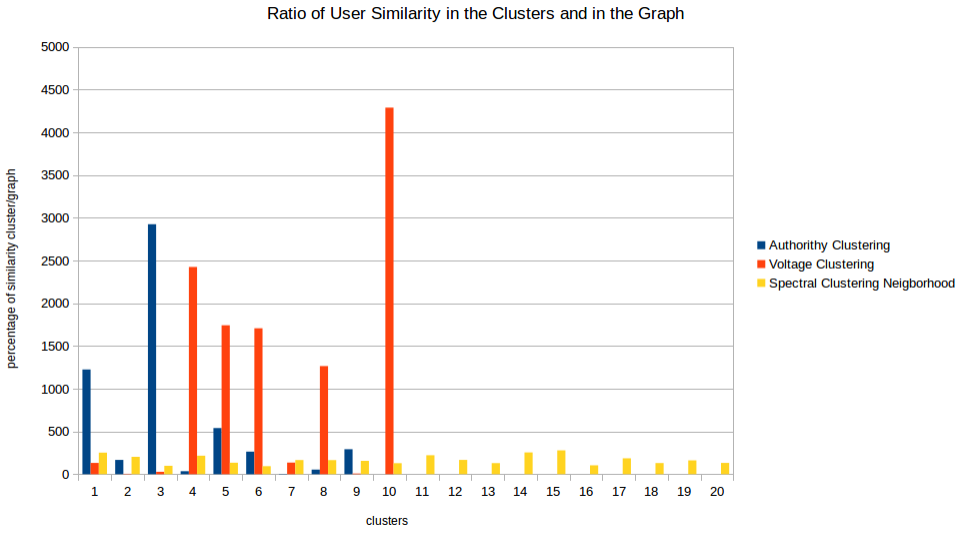


Figure 2.10: Ratio between the user similarity based on the Like and Boards similarity in clusters against the user similarity in the entire graph. The X axis shows the clusters, the Y axis the ratio between the two user similarities.

popular. We retrieve the items, for both the entire graph and for each cluster. In Figure 2.11 we show, for each cluster, the percentage of most popular items in that cluster which are also popular in the graph. The figure shows only those clusters that contain at least 50 users.

For most of the clusters, the percentage of popular items which are popular in the graph as well, are above 60%, and for some of the clusters found by authority clustering and spectral clustering the value reaches 90% of similarity. The only difference is voltage clustering, for which all the clusters have below 70% of common popular items. This particular behavior of Voltage clustering, together with the high similarity inside the cluster found by this algorithm previously shown in Figure 2.10, is an indication that voltage clustering is able to identify communities in which users share interest about particular fashion trends, that can be considered different from the general trend of the network. Knowing this, we could enforce our hypothesis that by recommending users that lie in the same community we can also recommend items based on these connections, since these users also share particular fashion interests.

2.5.3 Density of the Clusters

Considering our objective of finding communities of users that are connected to each other, a good indication of the quality of the clusters is to compare the density of the cluster with the density of the whole graph.

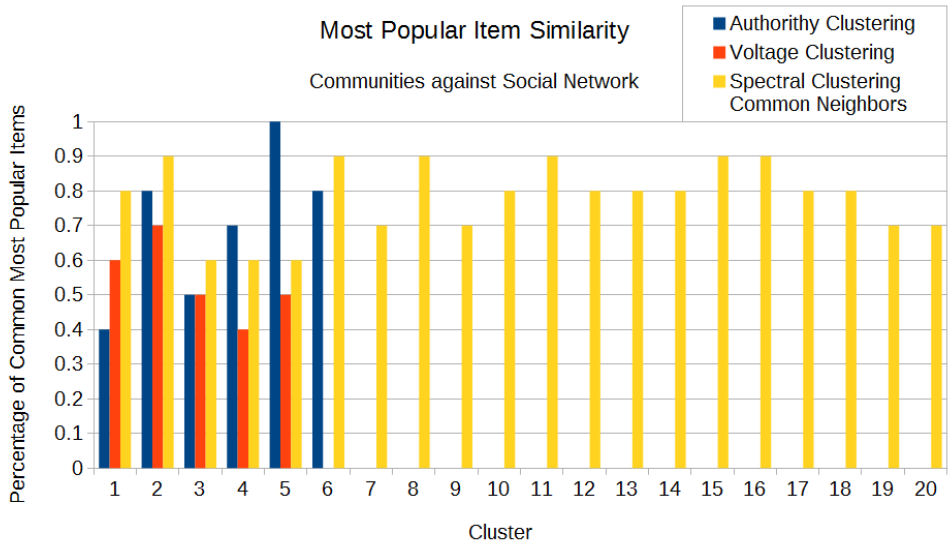


Figure 2.11: The graph shows the percentage of most popular items that are in common between the clusters found by Authority, Voltage and Spectral Clustering and the items that are generally popular in the social network. The x-axis shows the values for the different clusters, the y-axis the percentage of items that are popular in both that cluster and the graph.

We remind the definition of density $D(G)$ of a graph $G = (V, E)$ as

$$D(G) = \frac{|E|}{|V| \cdot (|V| - 1)}$$

When the clustering algorithm finds the set of nodes C_i inside the i-th cluster, we can then extract the sub graph $G_i = (C_i, E_i)$ of G with only the nodes in C_i and the edges that are incident on these nodes $E_i = \{(u, v) \in E \mid u \in C_i \wedge v \in C_i\}$.

We can now compare the density of a cluster and of the original graph as $\frac{D(G_i)}{D(G)}$, where a value greater than 1 shows that the density in the cluster is higher than the density in the graph.

In Figure 2.12 we show the ratio of the average density of the clusters, for each clustering algorithm, against the density of the graph. The plot shows how this ratio changes for the different numbers of clusters, on the x-axis.

We can see how all the ratios are greater than 1, meaning that on average the clusters are more dense than the graph. As shown in the previous section, some clusters contain only one node, or very few nodes, hence the density of their sub graphs are 0 or close to 0. Nevertheless, it is possible to see that the higher the number of clusters, the larger the ratio is, meaning that the clusters are more densely connected structures. This is especially true for the authority clustering algorithm, the average density, when 30 clusters are extracted is 90 times larger than the density of the graph. This is expected

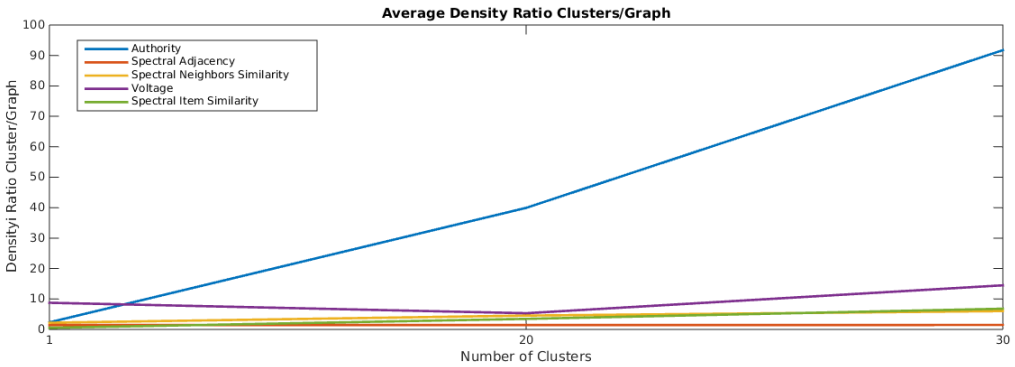


Figure 2.12: Ratio between the average density of the clusters compared to the density of the graph. The x-axis shows the number of clusters and the y-axis the density ratio.

because of the nature of the algorithm; since we know that most of the users in the social network are only connected to the authorities, when authority clustering groups the nodes around these users we obtain sub graphs that are more dense than the original graph.

In conclusion, we can observe that all the clustering algorithm are able to find more dense sub graphs than the original graph. The cause of these results varies dependent on the algorithm and to understand if our techniques are able to correctly identify the reason why users connect to each others, we investigate if the evolution of the network follows the same patterns.

2.5.4 Evolution of the Clusters

The objective of clustering is to find communities in which the users connect to other users that lie in the same cluster. Our assumption is that considering the evolution of these communities, they will continue connecting to the same group of users.

To understand if this behavior applies to the Sobazaar social network we extract a sub graph of the original social graph by removing a subset of the edges, while keeping all the nodes. We call the sub graph *training set*, while we keep the set of removed edges as *test set*. We describe the details of this process in Section 4.2.5. We apply our clustering algorithms to the training set, obtaining the different clusters in which the users are grouped. For each edge (u, v) in the test set we check if v lies in the same cluster of u. If it does, we have correctly predicted that the users will connect to users in their same cluster.

In Figure 2.13 we show the results for all our clustering algorithms. The first result shown by this analysis is that each clustering algorithm is able to find groups in which at least one new edge is in the same cluster, since the percentage is positive for all the algorithms. While the different types of Spectral Clustering and the Social Network Clustering perform poorly in this

regard, both Authority Clustering and Voltage Clustering show how more than 40% of the new connections reside in the same cluster. This is particularly interesting because it shows how, even by focusing on small subgraphs of the social network, we could still perform accurate recommendations.

Finally, since we have observed that we can rely on these algorithms for finding communities that will evolve in time, especially for the authority clustering and for the voltage clustering, we can now move the analysis to the evaluation of the recommender system based on these algorithms, as we will show later in Section 5.1.3.

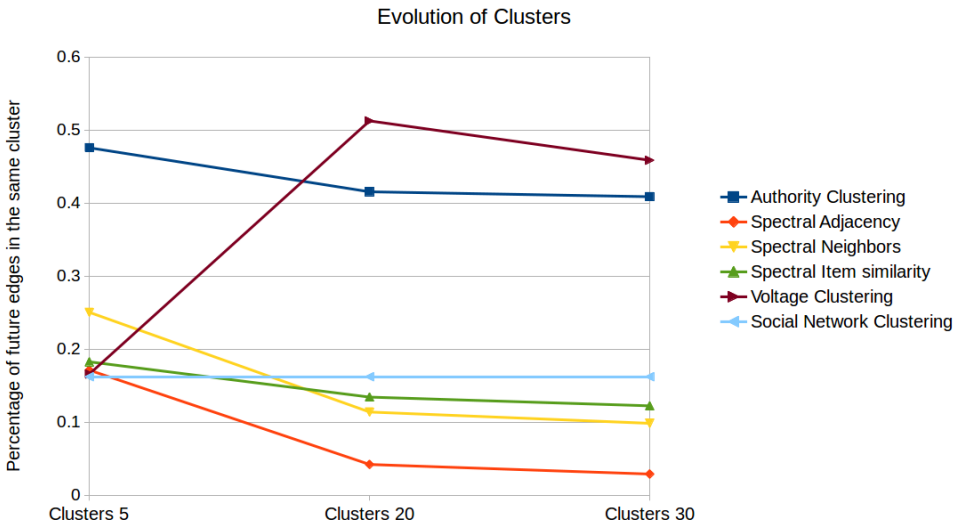


Figure 2.13: The graph shows the number of edges in the test set for which both ends lie in the same cluster. The different lines show the results for the clustering algorithms. In the x-axis the number of clusters extracted from the graph, the y-axis the percentage of edges for which this criterion holds.

Results of the Clustering Analysis

We have seen different points of view that can give a first overview of how clustering the social network could contribute to our research. First of all we have seen that the clustering algorithm find different clusters, each with different characteristics. While it is hard to identify fashion trends in the different communities, we can still observe how the algorithm group users that are connected and that share similar interests together. Algorithms as Voltage Clustering and Authority Clustering are able to identify clusters that describe an evolution that perpetuates in time.

Considering all these results, we will continue our analysis of how clustering can contribute in the social link problem in Section 5.1.3, where we will apply these algorithms to evaluate the recommender system.

2.6 Conclusion

In this chapter we described the data collected from crawling the Sobazaar API, we described the data scheme used in order to store the data collected and we analyzed aspects of the data.

Section 2.1 described the Sobazaar application for the iPhone. A user of the application can through the feed see what the users or brands she follows have liked or which boards they have published.

Section 2.2 presented how the data was collected using several crawlers on the Sobazaar API and how we structured the database. The terms follower, following, unfollow and unlike was defined.

Section 2.3 contains a detailed description of the data crawled, and analysis of each of the tables in the database, to get a better insight in the data.

Section 2.4 gives an introduction to the graph notation used in this report, furthermore it gives an analysis of the social network created by using the follow as a directed edge, connecting the users.

Section 2.4.6 starts combining the analysis of the social network and the likes and boards previously described. It is shown that the similarity inside the social graph, on average, is higher than nodes, which are not directly connected in the network.

Section 2.5 described the cluster generated by the different clustering algorithms which will be described in Section 3.5, and how each of them generated clusters with different characteristics which could have useful applications.

After completing the data analysis we will now investigate which techniques we can use in order to address the problems stated in the Section 1.1 taking the observations made in this analysis as the starting point.

Chapter 3

State of the Art

This chapter describes the state of the art approaches in the field of recommendation systems and social graphs. Section 3.1 describes the current state of the art techniques in recommendation systems. Section 3.2 describes which information can be retrieved by analyzing and using a social network. Section 3.3 describes a common approach used in recommendation systems, namely matrix factorization and describes how social information can be used directly to improve the recommendations.

Section 3.4 elaborates the problems which exists when predicting missing links in a social network, which is the equivalent to having to recommend users to one another. Section 3.5 presents common clustering algorithms which have an application in the context of this report.

3.1 Recommender Systems

This section introduces the objective, the theory and the notation of recommender systems.

We have stated in Section 1.1 the main problem we wish to address is how to recommend users to one another. We propose that techniques which are used in order to recommend items in most cases can have application for recommending users as well. We therefore investigate techniques for item recommendations as well as user recommendation in this section.

In [14] we have already described the most important goals of a recommender system, where we have focused on the task of recommending items to users. In Section 3.1 we will instead describe different objectives that the modern recommender systems have addressed. Moreover, we will expand our research in a social network context, where the recommender systems incorporate information about users' relationships or when the objective is shifted to increase the connectivity of users. Later in the same section we will remind the reader of the theory and notation behind recommender systems.

In [14, Section 4.3.1] we have addressed the difficulty of evaluating our recommender system for the Sobazaar application. Several problem arose:

defining how implicit feedback affects the recommendation, offline evaluation processes, absence of negative feedback and extremely sparse data. In Section 4.1 we will explore different ways of dealing with this problems, looking at what researchers have already studied in similar cases.

Introduction to Recommender Systems

In modern web based applications, recommender systems are an important tool to narrow the users' search for the provided services, from the hundreds of thousands to those few the user is really interested in. In an application like Sobazaar, where users are browsing items, a recommender system's main goal is to find the items a particular user prefers.

In the research, the two most popular approaches for designing recommender systems are content-based filtering and collaborative filtering.

Content-Based Filtering Content-based filtering (**CBF**) is a technique used for recommending items to users based on the content describing the items, such as a text-description. By using this information, users' preferences can be described in terms of the content they have showed a preference for, which then again can be used for recommending new items.

Collaborative filtering (**CF**) is a technique used to filtrate information using the effort of multiple collaborators. In recommender systems, where we are interested in finding new items preferred by users, other users can be used collaboratively to find the preferences of items for each others.

Collaborative filtering can be grouped into two different types: Memory-based and Model-based.

In the Memory-based approach, the ratings given by users are used directly in predicting new items, either by using the ratings of similar users, an approach called user-based, or by using the ratings of similar items, called item-based.

In the Model-based approach, the preferences of similar users or items are not used directly, but instead utilized to make a predictive model. This makes enables the system to give new recommendations without having to access the whole database of users, items and their ratings, but rather use a model which abstracts on a higher level how preferences of items relates to users.

In a traditional application these two methods have been the most commonly used. In [14, Chapter 3] we described the current state of the art and applied some selected methods to our Sobazaar application seen in [14, Chapter 4]. Nevertheless, the social network nature of Sobazaar opens two other relevant points:

- Exploiting the social network to boost items to users recommendations.
- Recommending users to other users.

For the first point, traditional recommender systems assume that all users are identical. As shown before, CF techniques make use of other users to predict the preference of a new user. Similar users are considered to have equal influence, but this assumption disregards the different relationships people establish in a social context. The social network contains all the information on how the users interact with each others, what is their type of common activity, and to whom they are closer. These are important features that can be used while designing a recommender system, and we will deepen our research for these possibilities in Section 3.3.2.

The second point opens the path to one of the main topic of our research: finding to whom users would prefer to connect. So far the objective has been to recommend items to users. Nevertheless the task of recommending users to other users can be as interesting as the previous one. In the literature, this topic is defined as *Social Link Prediction*. In 3.4.2 we will define the particularity of this problem, why it is important to research and we will present different techniques to address this task.

Recommender System's Definition

Now that we have described the main ideas of recommender system we can introduce the notation we will use throughout our report for defining the recommender systems concepts.

Let U denote the set of users for a given recommender system and I denote the items of the system. We use the set R to denote the recorded ratings in the system and the set S, such as $S = \{1, 2, 3, 4, 5\}$ or $S = \{Dislike, Like\}$, to denote the possible values of ratings. To get the rating given to an item $i \in I$ by a user $u \in U$, we use the notation r_{ui} and to get the set of users of U,who has rated on a specific item $i \in I$ we use the notation U_i. Similarly let I_u denote the set of items rated by a user u.

The goal of recommender systems is to learn a function $f : U \times I \rightarrow S$, which can predict the rating a user $u \in U$ would give a not yet rated item $i \in I$, this rating is also preferred to in this report as $\hat{r}_{ui}$. Using this function we can find an item i^*, which will be deemed having the highest preference for a user u.

$$i_u^* = \arg\max_{i \in I \setminus I_u}, f(u, i) \tag{3.1}$$

3.2 Analyzing Social Networks

Social Network Analysis is the study of the social relationships between entities [20]. This discipline views these relationships in terms of nodes and edges, representing the actors of the interactions, describing their relationships. Therefore a suitable way to represent this model is a graph, that intrinsically describes the concept of network.

Due to the proliferation of social network websites the interest in social network analysis has grown among technical researchers, after being a topic of research in the areas of psychology, sociology and behavior science [27]. By the mean of computer science, data mining and graph theory, social network analysis has been seen as a way to utilize the large amount of data that users reverse in websites such as *Facebook*, *Twitter* and others.

Social network analysis attempts to address an important problem: determining the function and role of the individuals in the network. Humans interact differently, but their behavior is reflective of their persons, and from their activity in the social networks information about them can be extracted, and used for different purposes: finding leaders of a community, the *early adopters* of a new product, terrorist groups and, last but not least, used for recommender systems.

In Section 2.4.1 we define a social network and the standard methodology for mining information. In Section 3.2.1 we propose solutions to one of the relevant problems for our research: how to find influential users in a fashion social network.

3.2.1 Social Influence

Some persons are more influential than others, meaning that what they do in terms of activity, ideas, publications and so on is considered, by other people, more important than what other would do. It is enough to consider how people rely on important scientists, prime ministers or celebrities for their everyday life to understand how relevant is the influence of a person in understanding the behavior of individuals.

A social network models these interactions, and heuristic have been studied to recognize influential nodes. Some of these are the *betweenness*, *closeness* or *degree* centralities.

The most intuitive of these measures is the degree centrality, describing the number of edges incident in a node v. This criterion simply describes the size of the neighbourhood of a node, therefore its possibility of sharing or receiving information. For a node $v \in G$ the measure is defined as:

$$D(G, v) = deg(v)$$

where $deg(v)$ is the number of edges incident in v, both inner edges and outer edges.

Betweenness centrality describes the probability that a node acts as a bridge between other nodes in the graph [20]. We define the shortest paths, *or geodesics*, between two nodes v_i and v_j as the notation Ω_{v_i,v_j} and the subset of these geodesics that also include v as intermediate node, $\Omega_{v_i,v_j}(v)$. Betweenness centrality is then defined as:

Definition 3. *The betweenness centrality for v given graph G, $BC(G, v)$, is the number of times it appears in the shortest path between all the pairs of*

nodes in graph G:

$$BC(G,v) = \sum_{v_i,v_j \in V} \frac{\Omega_{v_i,v_j}(v)}{\Omega_{v_i,v_j}} \text{ where } v_i, v_j \neq v$$

The following pseudocode shows how to calculate the betweenness centrality value for each node in the graph.

```
1 T[][] <- Floyd_Warshall(G) // T[i,j] = shortest path between <i,j>
2 for v in V
3    for   u,w in V: u,w != v
4       if v in T[u][w]
5          B[v]<- B[v] + 1
```

To compute the matrix of the shortest paths between all the pairs of nodes in the graph the Floyd-Warshall algorithm can be used, associating a weight of 1 to each edge $e \in E$. The running time of this algorithm is $O(V^3)$. The two cycles to calculate the betweenness have a cost of $O(V \cdot (V-1) \cdot (V-2)) = O(V^3)$. Therefore the cost is $O(V^3)$, that in a large social network could become a problem.

The intuition behind the last measure, closeness centrality, is the time it takes to spread information from a node v to the rest of the graph. This can be seen as a measure of how close a node is to the other nodes. With V_v describing the vertices reachable from a node v, the *farness* of the node v is defined as the sum of the shortest distances to all the nodes in V_v

$$F(v) = \sum\nolimits_{v_i \in V_v} d(v, v_i)$$

where $d(v, v_i)$ is the shortest distance between v and v_i, defined as V if there is no path in $\langle v, v_i \rangle$. The closeness is computed as:

$$C(G,v) = \frac{|V_v|^2}{|V|F(v)}$$

In this case the fewer the nodes that are reachable from a node v the smaller the numerator, hence a lower value. Similarly, more nodes that are not reachable from v make the denominator bigger, again decreasing the centrality measure.

3.3 Combining Social Network and Recommendations

To answer the problems we have stated in Problem 1 and Problem 2 we wish to investigate techniques for understanding how to extract information from a social network that can be used for our recommender system.

Traditional recommender systems often use a collaborative approach when recommending items to users. This section investigates collaborative based

techniques for recommending users to users. Although techniques presented here are mainly focused on recommending items, these approaches are applicable for recommending users as well, as the same metrics used for recommending items also exists in the context of users, i.e. there exist underlying reasons why one user follow another.

In order to explore collaborative filtering we refer to a common approach, namely latent factor models which make use of underlying features or factors, which are not explicitly stated but inferred from users preferences for items. In the context of Sobazaar such features might for example have factors which are not apparent from the description of the product, i.e. for a summer dress and sandals an underlying factor which might not be directly stated could be that these items are usually popular during the summer season, the same might hold true for the underlying design and the current popularity of the brand which has produced them. Latent factor model often represent the data as a matrix consisting of ratings, indicating how much a user preferred an item. A common type of latent factor model approach is matrix factorization [15].

The goal of matrix factorization is to predict a rating a user would give an item which he has not yet interacted with, based on her previous interactions and other users previous interactions. Recommendations can therefore be made by selecting the items with highest predicted preference.

More formally given a rating matrix R, where $r_{ui} \in R$ and r_{ui} represents a user u's preference of item i, we want to find the predicted values $\hat{r}_{ui}$ for all users and items by decomposing the matrix R into two latent factor matrices P and V such that $R = P \cdot V$. We then find the predicted rating for a given user and item by taking the dot product of the column vectors of P and V, p and v respectively [15].

$$\hat{r}_{ui} = p_i^T \cdot v_u$$

Where p_i^T is transposed. A way to learn the latent factor matrices is to minimize the squared error given some training data T:

$$\min_{p_\cdot, v_\cdot} \sum_{(u,i) \in T} (r_{ui} - p_i^T \cdot v_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{3.2}$$

here λ is a regularization constant, and the use of $||q_i||$ and $||p_u||$ is L2-Norm, L2 is used because L1 norm is non-differentiable, and will not work with Stochastic Gradient Descent. The difference between L1 and L2 is that L2 is the sum of the square of the weights, and L1 is the sum of the weights. There exist several techniques to minimize the squared error, one such technique is stochastic gradient descent [15].

In stochastic gradient descent we first find the error of the current $r_u i$ of the training set by using the column vectors:

$$e_{ui} = r_{ui} - p_i^T v_u$$

We then follow the update rules below which modify the column vectors, P and V, given a learning rate γ, following the opposing direction of the gradient λ until convergence:

$$v_i = v_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot v_i)$$

$$p_u = p_u + \gamma \cdot (e_{ui} \cdot v_i - \lambda \cdot p_u)$$

In conclusion, this approach allows us to approximate a predicted rating r_{ui} for a given user u on an item i.

3.3.1 Probabilistic Matrix Factorization

In the previous section we briefly described the general matrix factorization approach, the paper [22] introduces a new concept, namely probabilistic matrix factorization which is based on a probabilistic model.

Probabilistic matrix factorization more commonly used in order to incorporate social networks directly in the decomposition of the rating matrix [18] [12], which makes it relevant in the context of Sobazaar.

The problem addressed by this approach is that there is a need for algorithms which scale linearly with the number of observations and have acceptable precision even when applied to sparse data or data that is imbalanced. To illustrate the intuition behind this approach we take an example based on Sobazaar.

We define m to be the pieces of clothing and n the users of Sobazaar and let r be a value with the domain $[0, 1]$, where 1 indicates that a user has liked an item and 0 if a user did not like an item.

First we create a rating matrix $R : M \times N$, where $M = |m|$ and $N = |n|$ which contains the rating of user n_i of the products m_j at position $R_{i,j}$. By choosing an arbitrary number of factors D we can find two latent factor matrices U and V as $U \in \mathbb{R}^{D \times M}$ and $V \in \mathbb{R}^{D \times N}$ such that $R = U^T \times V$.

We define U_i and V_j to be column vectors where U_i contains user specific latent values, while V_j contains clothing specific latent values. The conditional distribution of R is then defined as

$$p(R|U, V, \sigma_R^2) = \prod_{i=1}^{M} \prod_{j=1}^{N} \mathcal{N}\left[(r_{i,j}|U_i^T V_j, \sigma_R^2)\right]^{I_{i,j}^R} \tag{3.3}$$

where $\mathcal{N}$ is a probability density function with mean $\mu = U_i^T V_j$ and variance σ^2, while $I_{i,j}^R$ is an indicator function returning 1 if the position at (i, j) is 1, 0 otherwise.

The prior probability of the latent feature vectors are defined with a zero mean density function:

$$p(U|\sigma_U^2) = \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2 \boldsymbol{I}) \quad (3.4) \qquad p(V|\sigma_V^2) = \prod_{j=1}^{M} \mathcal{N}(V_j|0, \sigma_V^2 \boldsymbol{I}) \quad (3.5)$$

Prior probability of U and V, defined by a zero mean density function.

The objective function is then found by minimizing the sum-squared-error of the log posterior distribution

$$E = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{N} I_{i,j}(R_{ij} - U_i^T V_j)^2 - \frac{\lambda_U}{2} \sum_{i=1}^{M} ||U_i||_F^2 + \frac{\lambda_V}{2} \sum_{j=1}^{N} ||V_j||_F^2 \quad (3.6)$$

λ is the regularization constant given for each matrix respectively and usually defined by the variance of the matrices further described in Section 3.3.3, and $|| * ||_F^2$ describes the frobenius normal form of a matrix. To optimize this objective function E we can apply algorithms such as stochastic gradient descent as presented in the previous Section 3.3.

In the following section we will describe how this probabilistic matrix factorization is used in order to create recommendations which incorporate a social network.

3.3.2 Social Recommendations using Probabilistic Matrix Factorization

The following sections contain a description of state of the art recommendation systems which use social networks in order to improve their accuracy. As we have already discussed in Section 3.1, traditional recommendation systems assume that users are independent and identically distributed. This assumption ignores the social connections among users of a system. In reality, we always turn to friends for asking suggestions and advice, and this turns into a natural *offline recommendation*, where we simply receive recommendations from our friends. In [29] it has been studied how friends' recommendations are preferred over those given by a recommendation system. This is because of the concept of *trust*. We trust our friends, we trust people we believe being expert in a field but we do not, usually, trust an automatic system, even if its recommendations are tailored on our needs more than those from a friend would be. Because of this reason, recommendation systems which do not consider social relationships fail modelling the users' preferences properly. Recently, researchers have started including the social properties of a user while constructing a recommendation system. Before introducing algorithms which make use of both social networks and user-item data, we introduce the concept of *social trust*.

Social trust Social trust is the concept of describing the strength of a relation between two users [18]. In the context of recommendation systems, the more a user trust another user the more gladly she will accept recommendations coming from her.

Given a social graph, trust could be considered to be a weight on the edges, representing the trust between two users in the direction of that edge. However, in a normal scenario, we do not have these weights, hence trust must usually be inferred from the users' activity. How the social trust is computed is highly dependent on the domain. Intuitively, the social trust could be inferred by investigating the number or the type of interactions between a user and her friends. For example, a high amount of similar interaction in common would give a high trust value, while a low value would be given if there is little or no interaction.

In our case, the option which would be available is to investigate whether a user has some interactions with the systems in common with her friends, as for example liking the items a friend also liked or that she just put into a board. This source of information is implicit and one can not assume that similar interactions between two users is a result of trust. To address this we would introduce the concept of discounting the trust calculated through this method, by assigning confidence to each of the trust values generated. This confidence is based on the similarity of users and the similarity of interactions in the data.

3.3.3 SoRec

Figure 3.2 shows a social graph of trust. The weights on the edges describe the degree of trust from one user to another. The trust value is bounded in the interval $[0, 1]$, where a higher value represents higher trust.

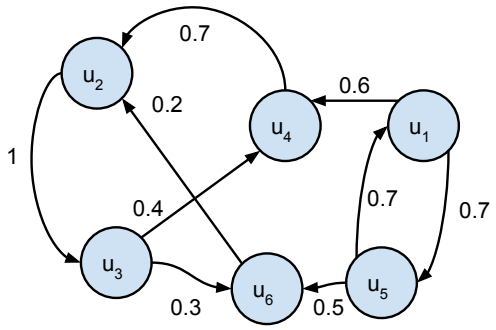


Figure 3.2: Social Graph $G = (V, E)$

A way to use this social graph in order to create recommendations is presented in the paper [18], which introduces SoRec. In order to create recommendations SoRec converts the graph 3.2 into a matrix, referred to as the

social network matrix $\mathbf{C}$: $N \times N$, where N is the number of users, as shown in Table 3.1. Each value in the matrix corresponds to the trust value in the graph where non existing edges are set to zero. The intuition of SoRec is to apply matrix factorization on both the social network matrix and user-item matrix in order to generate recommendations, even if the user-item matrix is sparse.

	u_1	u_2	u_3	u_4	u_5	u_6
u_1	0	0	0	0	0.7	0
u_2	0	0	0	0.7	0	0.2
u_3	0	1	0	0	0	0
u_4	0.6	0	0.4	0	0	0
u_5	0.7	0	0	0	0	0
u_6	0	0	0.3	0	0.5	0

Table 3.1: Social Network Matrix $\mathbf{C}$

The Table 3.2 shows a user-item matrix $\mathbf{R}$ where the value of a cell indicates whether the user has liked the item. For example, the cell $[1, 1]rangle$ with a value of 1 indicates that user u_1 liked the item i_1. Otherwise, a cell with a value of 0 shows that the user has never liked or interacted with the item.

	u_1	u_2	u_3	u_4	u_5	u_6
i_1	1	1	0	1	0	1
i_2	1	0	1	1	0	0
i_3	1	1	0	0	0	0
i_4	0	0	1	0	0	1
i_5	0	1	1	0	0	1
i_6	0	1	1	1	0	0

Table 3.2: User-Item $\mathbf{R}$. The rows represents the items, while the columns the users.

The social network matrix $\mathbf{C}$ and user item matrix $\mathbf{R}$ are then decomposed into three factor matrices such that $\mathbf{C} = U^T Z$ and $\mathbf{R} = U^T V$.

Some users in the social network might have a lot of outgoing edges, here the value c_{ik} of the social trust should be diminished as the confidence in that the user trusts all users also decreased. If a user on the other hand has a lot of incoming edges the confidence in the trust value c_{ik} should be increased as she is trusted by many users. To incorporate the decay in trust the term $c^{\star}_{ik}$ is introduced and defined as:

$$c^{\star}_{ik} = \sqrt{\frac{indegree(v_k)}{outdegree(v_k) + indegree(v_k)}} \times c_{ik} \tag{3.7}$$

where v_k is the node corresponding to c_k in the graph G and the functions *indegree* and *outdegree* return the number of directed edges ending and originating in v_k respectively. In order to compute the factor matrices the indicator function $I_{ik}^{\mathbf{C}}$ is introduced. $I_{ik}^{\mathbf{C}}$ returns a value of one if there is a non-zero value at position (i, k) in matrix $\mathbf{C}$ and is used in order to refer if there is any data available for the current position in a matrix.

The probability for the $N \times N$ size matrix $\mathbf{C}$ is then given by the latent matrices U and Z, where U_i and Z_k are column vectors at the i-th and k-th position, as:

$$p(C|U, Z, \sigma_c^2) = \sum_{i=1}^{N} \sum_{k=1}^{N} I_{ik}^{C}(c_{ik}^{*} - g(U_i^T Z_k))^2 \tag{3.8}$$

where $g(U_i^T Z_k)$ is a logistic function used for bounding a value in the range $[0, 1]$, defined as:

$$g(x) = \frac{1}{1 + exp(-x)} \tag{3.9}$$

A similar approach is taken for the user-item matrix $\mathbf{R}$, where m is the number of users and n is the number of items, the probability is defined as:

$$p(R|U, V, \sigma_R^2) = \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}^{R}(r_{ij} - g(U_i^T V_j))^2 \tag{3.10}$$

Finally the objective function over all matrices $\mathbf{R}$, $\mathbf{C}$, U, V, Z is defined as the following objective function:

$$\begin{aligned} \mathcal{L}(R, C, U, V, Z) = & \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}^{R}(r_{ij} - g(U_i^T V_j))^2 \\ & + \frac{\lambda_C}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} I_{ik}^{C}(c_{ik}^{*} - g(U_i^T Z_k))^2 \\ & + \frac{\lambda_U}{2} ||U||_F^2 + \frac{\lambda_V}{2} ||V||_F^2 + \frac{\lambda_Z}{2} ||Z||_F^2 \end{aligned} \tag{3.11}$$

λ is the regularization constant and defined as $\lambda_C = \frac{\sigma_R^2}{\sigma_C^2}$, $\lambda_U = \frac{\sigma_R^2}{\sigma_U^2}$, $\lambda_V = \frac{\sigma_R^2}{\sigma_V^2}$, $\lambda_Z = \frac{\sigma_R^2}{\sigma_Z^2}$ where σ^2 is defined as the variance of a matrix.

The first term describes the user-item matrix while the second term describes the social network objective. Here λ_C describes the magnitude the social network has on the final predicted rating. The last term is the regularization where different weights can be assigned to each of the three matrices. Giving more weight to one of them would result in a higher bias towards that

model. We can apply stochastic gradient descent in the same way it was described in Section 3.3.

The problem with applying SoRec directly on the Sobazaar data is that we do not have any information regarding the amount of trust a user has to the other. As previously discussed in Section 3.3.2 it could be possible to create a model which allows, with some confidence, to generate trust values by for example investigating if users liked the same items one after another in a short period of time, which would indicate that a user might have influenced another. Although our data does not allow that, since we do not have real timestamps of the like event.

3.4 Social Link Problem

In 3.4.1 we will describe the problem and why it is relevant in our research, while in 3.4.2 we present several state of the art techniques to address this task.

3.4.1 Problem Definition

We have already described how social network analysis has been widely researched to extract information about the actors and their interactions. Recently, the task of understanding the mechanisms modeling the evolution through time of the relationships among entities in such a complex structure has received interest [16] [17]. A social network is a dynamic object where connections can be tied and untied, or new entities can enter in the process changing the structure of the network. An important problem in this scenario is to understand if the evolution of the network can be explained by information held by the network itself, without the need of reasons exogenous to the network. This lead to the task of social link prediction:

Social Link Prediction Given a snapshot of a social network at time t, the task of link prediction aims to accurately predict the edges that will be added to the network during the interval from time t to a given future time t' [16]

In our application, the social link prediction problem can be formally defined as follows. Let $G = (V, E)$ be a social network in which each edge $e = (u, v) \in E$ represents an interaction between u and v that took place at a particular time $t(e)$, this time is when a follow connection was made. We select four different time points $t_0 < t_1 < t_2 < t_3$. $G[t_0, t_1]$ is the graph containing only the edges in $E_{[t_o,t_1]} = \{e \in E \mid t_0 \leq t[e] < t_1\}$, where $t[e]$ is the time label associated to the edge e. Using $G[t_0, t_1]$ as training set for a link recommender system, the objective is to propose a list of edges that do not appear in $G[t_0, t_1]$ and that are expected to appear in $G[t_2, t_3]$. Intuitively, a relevant recommendation will be an edge that is not present in the training

set and that is present in the test set, meaning that we have recommended to a user a new followings that she will eventually connect with.

3.4.2 State of the Art

In this section we present some of the possible methods for solving the link prediction problem. Each of the following approaches assign a $score(u, v)$ to each pair of nodes $\langle u, v \rangle$ of the graph, defining the *connection strength* between these two nodes. Intuitively, the higher the score the better the recommendation will be. This score is computed using the graph based on the training set and, for a given user, the recommendation will be a set of other users she is not connected with that have obtained the highest scores, accordingly to a specific method.

3.4.3 Neighborhood Based Methodologies

These methods define the $score(\cdot, \cdot)$ dependently on the structure of the neighborhood of a node. Let $\tau(u)$ be the set of neighbors of a node u. For two nodes u, v in the graph the idea is that if the intersection of $\tau(u)$ and $\tau(v)$ is large, then it is more probable for these two nodes to be connected in the future since they are connected with the same other users. Following this intuition, a simple implementation of the score can be defined as

$$score(u, v) = \mid \tau(u) \cap \tau(v) \mid .$$

We discuss the following methods, as reviewed in [16], which tries to refine this definition of similarity.

Jaccard's coefficient Given a random feature f representing a characteristic of one node, either u or v, the Jaccard coefficient describes the probability that both nodes possess this feature. The feature can be specified for example having a user x in the neighborhood, which means that the score can be measured as probability of having users in common:

$$score(u, v) = \frac{\mid \tau(u) \cap \tau(v) \mid}{\mid \tau(u) \cup \tau(v) \mid}.$$

Adamic/Accard The Jaccard's coefficient simply counts the number of features in common between two users u and v. Another option is to weight those features that are more rare to appear with higher weights, because they will define the similarity between the two users more than features that more frequently appear to be in common in pairs of users. If the feature that we are considering is a user in common in the neighborhood of two users u and v, the measure can be defined as

$$\sum_{z \in \tau(u) \cap \tau(v)} \frac{1}{log \mid \tau(z) \mid}$$

With this definition, having in common a user z who has a small neighbor, hence with a small probability of appearing in some other user's neighborhood, gives an higher score.

Preferential Attachment This measure defines the probability of two nodes u and v to connect in the future to be directly proportional to the size of their neighborhoods. The intuition is that the more connections they have, the easier is to get in contact with other users. Then the score is calculated as follows

$$score(x, y) = \mid \tau(u) \mid \cdot \mid \tau(v) \mid$$

PageRank

PageRank is a ranking algorithm proposed by Sergey Brin and Lawrence Page in 1998 as a feature for their search engine Google [5]. The algorithm was initially designed to infer the importance of a page by exploiting the hyperlink structure of the web graph. This idea can also be applied in a social network, where the objective is to identify the important users. The main criterion of PageRank is to give high scores to pages that are highly cited, i.e. referenced or hyper-linked by other pages.

The intuition behind the ranking principle is to try to model the user's behavior in her web navigation. The assumption is that a *random surfer* performs a random navigation starting from a given web page. This user will randomly visit other pages through the hyperlinks it finds, until it decides to jump to another random page using the address bar. With this kind of behavior the probability that the random surfer will visit one of the pages of the graph is exactly what the PageRank is describing. PageRank is then a measure of how many pages are referencing a given page: the higher this number, the higher the probability that the random surfer will end its navigation in that page. Furthermore, PageRank also weights the links accordingly to the importance, expressed in term of PageRank score, of their source nodes. The algorithm will then consider both the number of edges pointing to a particular node and their quality. For example if a page is linked by *bbc.co.uk* it is highly probable that the reference is not broken and that its quality is good, therefore these links will have a higher weight for the algorithm.

PageRank can be applied in any network context, hence also in our social network. Here the web pages are the users and the hyperlinks the relationships among them. The objective in our case is to find those users who are attracting more followers, and again that are connected with other popular users. Using

the same criterion explained for a search engine, these important users will have higher PageRank scores.

The formula to calculate the PageRank for a node u of the network is

$$PR(u) = \alpha \sum_{v \in T_u} \frac{PR(v)}{C(v)} \tag{3.12}$$

where α is the probability for one user will visit one of the users she is connected to, while with probability $1 - \alpha$ to jump to a random destination. T_u is the set of users that are following u. $PR(v)$ and $C(v)$ are respectively the PageRank and the outdegree of the node v, this last one used to normalize the weights of the edges by the number of users v is following. To consider the two aspects of the measure, both the number of links and the quality of these, the equation makes a recursive use of the PageRank value. Because of this, multiple iterations of the algorithm are necessary in order to approximate the PageRank to reflect the theoretical true values.

A PageRank based recommend system will then calculate the score for all the nodes of the social network. Sorting the users by this value a recommendation is then a list of k users with the highest PageRank value. A drawback of using only the PageRank as the recommendation criterion is that this algorithm does not take into account the preference of each user, but solely use the general link structure of the graph.

3.4.4 Conclusion

A social network is a constantly changing graph, where users add new connections with each other or users remove their already existing connections. A way to look at the social graph in a specific timeframe, means that all the connections inside the graph must have a timestamp, to see when their connections was made. The social graph we have generated does contain timestamps, although they are mostly from the same timeframe because of the way the crawler was made, meaning we cannot use the social link prediction. In Section 3.4.2 we present the score function which computes a score between two users, that can be used to strengthen a recommendation. Then in Section 3.4.3 we describe Pagerank, that is a way to give each node a score of popularity, i.e. a node that is being visited many times by the pagerank, gets a high score.

3.5 Clustering

Clustering of social graphs is a viable method to find patterns which can be used to predict future trends or recommendations. Clusters in social recommendations are often referred to as communities [8]. In the following sections we describe the theory and intuition behind state of the art clustering algorithms and if they are applicable in the context of Sobazaar.

3.5.1 Social Network Clustering

The paper [21] presents a novel way of finding communities in a social network graph. The clustering approach presented in this paper is referred to as (α, β)-clustering. The motivation of this approach is to find communities which are highly connected, secondly we do not want to assume that users in these communities only belong to one community. The α value is therefore a threshold defining how well users in a community are connected to other communities, and the β value is a threshold defining how well users in a community are connected with one another. The function $E(u, C)$ returns the set of all nodes $v \in C$ with an edge to u, while $|C|$ refers to the number of nodes in the cluster C. The definition is found in [21]:

Definition 4. *Given an undirected graph $G = (V, E)$ where V is the set of vertices and E the set of edges, and each vertex has an edge to itself, we define an (α, β)-cluster C such that $C \subset V$ given that*

- *All vertices in C are densely connected:*

$$\forall v \in C, |E(v, C)| \geq \beta \cdot |C| \tag{3.13}$$

- *All vertices in $V \setminus C$ are sparsely connected:*

$$\forall u \in V \setminus C, |E(u, C)| \leq \alpha \cdot |C| \tag{3.14}$$

In addition to this definition they also propose a deterministic algorithm to find all (α, β)-clusters in a given undirected graph, which is based on the notion of $\mathcal{P}$-champions. The intuition behind these champions is that a node, which champions a cluster, has less connections outside the cluster than a $\mathcal{P}$ percentile of the cluster size. $\Gamma(c)$ is defined to be the neighbors of c and $\tau(c)$ is defined as the neighbors of neighbors $\tau(c) = \Gamma(c) \cup \Gamma(\Gamma(v))$.

Definition 5. *A node $v \in C$ champions a cluster C if the following holds:*

$$|\Gamma(c) \cap V \setminus C| \leq \mathcal{P} \cdot |C|$$

where $\mathcal{P}$ is a value bound in the range of $[0, 1]$.

The deterministic algorithm of finding the clusters using this approach is a direct consequence of definition 4. Note that the term $|(v, cluster)|$ is defined as being the number of edges going from node v into the cluster *cluster*.

Listing 3.1: Determistic Algorithm to find the clusters.

```
1  SocialNetworkClusterer(α, β, size, V, E)
2  clusters ← ∅
3  foreach node c ∈ V
4     C ← ∅
5     foreach node v ∈ τ(c)
6        if |Γ(v) ∩ Γ(c)| ≥ (2β − 1)size
7           C.put(v)
8     if ∀v ∈ (V \ C), |E(v, C)| ≤ α · |C| ∧ ∀v ∈ C, |E(v, C)| ≥ β · |C|
9           clusters.put(C)
10 output clusters
```

Listing 3.1 shows the pseudo code for the deterministic algorithm proposed in the paper [21].

In relation to the Sobazaar domain we assume that people are interested in several different styles of clothing meaning they could be interested in a wide variety of users which have different items in their boards and like lists. This approach takes this assumption into consideration and allows the user to be in several clusters.

3.5.2 Betweenness Clustering

The paper [8] proposes a technique to identify communities in a graph by using the betweenness metric as described in Section 3.2.1.

The intuition of this approach is that most social-graphs contain tightly knit communities which are sparsely connected to other communities. By removing the edges which have the highest amount of betweenness the links between clusters get removed which makes it possible to detect these clusters.

The Figure 3.3 shows an example of two communities $\{A, B, C, D\}$ and $\{E, F, G, H\}$. The edge (D, E) has the highest betweenness as it has the highest number of shortest paths passing through it.

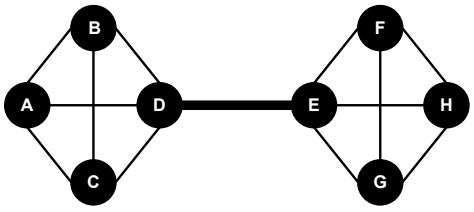


Figure 3.3: The betweenness of an edge between two communities, here (D, E) has the highest betweenness.

The proposed algorithm for detecting these edges is presented as having four steps:

1. For all edges, calculate the betweenness.
2. For all edges, remove the edges with the highest betweenness.
3. For all edges affected by the removal, recompute the betweenness.
4. If less than x edges remain terminate else go to step 2.

x is a user defined number, if too few edges are removed not fully connected communities might be split up, while removing too few edges results in a graph

still containing edges which connect communities. Step 3 is necessary as there might exists more than one connection between communities and if all nodes in the community only used the first connection in their shortest path the second connection is not be detected and will not be removed in the next iteration.

The Listing 3.2 shows the aforementioned steps which are taken in order to compute the cluster using the betweenness. The betweenness is computed for the whole graph as presented in Section 3.2.1 Definition 3.

Listing 3.2: Betweenness Clustering

```
1  BetweennessClusterer(G, numberOfEdgesToBeRemoved)
2  BetweennessCentrality ← computeBetweenness(G)
3  while(k < numberOfEdgesToBeRemoved)
4     score ← 0
5     to_remove ← null
6     for edge e ∈ G
7        if BetweennessCentrality.getScore(e) > score
8           to_remove ← e
9           score ← BetweennessCentrality.getScore(e)
10    G.removeEdge(to_remove)
11    BetweennessCentrality ← recomputeCentralityForAffectedLinks(
          to_remove)
12    k++
13 weakcomponentClusterer(G)
```

The function `recomputeCentralityForAffectedLinks` performance the third step mentioned above namely to recompute the shortest path for the nodes which are affected by the removal of an edge.

The `weakcomponentClusterer` creates the clusters by sequentially going through the graph's nodes clustering nodes which still have edges between one another into the same cluster and removing them from consideration.

The running time of the proposed algorithm, assuming the use of the Newman betweenness algorithm proposed in [23], is $O(n \times m^2)$ where m are the number of edges and n are the number of nodes in the graph.

This clustering approach allows cluster the social network only based on the structure of the network. Further it requires no specifications of how many clusters are wanted but a number of edges which should be removed which could allow for interesting results given that we do not know the number of communities.

In relation to Sobazaar this clustering algorithm allows for an alternative to the social network algorithm clustering algorithm presented in Section 3.5.1. It also only uses the social network structure in order to create clusters, but rather than allowing for nodes to be part of several clusters this algorithm restricts the nodes to be part of only one cluster. A difficulty using this clustering algorithm is to find the number of edges to be removed to the point where the clusters created are useful for creating recommendations.

3.5.3 Voltage Clusterer

Voltage clustering proposed by [32], presents a way to find clusters in linear time. The intuition behind the voltage clusterer is that the graph is seen as a circuit board where edges between nodes function as resistors with equal resistance, and by applying a voltage to a node at one end of the graph, and a ground applied to another node at the other end of the graph, we can distinguish between sections of the graph by looking at the voltage of each node. If the voltage of nodes is close together we expect that they are located in a densely connected part of the network while if there are large drops in the level of voltage we can expect that the nodes lie in a sparse region.

We define $V_1 = 1$ to be the vertex which receives the initial voltage, the source, and we define $V_2 = 0$ to be the ground or sink. These nodes are selected randomly from the graph, this means that results of this algorithm might be impaired as nodes in the same cluster might be picked thereby obfuscating the fact they belong to the same cluster.

The paper defines a function based on the Kirchhoff equations [10] in order to find the voltage of the remaining vertices in the graph:

$$V_i = \frac{1}{k_i} \sum_{j=3}^{n} V_j a_{ij} + \frac{1}{k_i} a_{i1} \text{ for } i = 3, ..., n \tag{3.15}$$

where k_i is the degree of node i, and a_{ij} is the adjacency matrix.

For each node in the graph, besides the source and sink we therefor update the voltage of node c, defined as V_c using the following update step:

$$V_c = \frac{1}{k_c} \sum_{i=1}^{n} V_{D_i}, \tag{3.16}$$

updating the voltage of a node by the average of it's neighbors. This update step is repeated a finite number of time until convergence.

Here the implementation used for this report deviates from the paper [32], as we are using an external library, more in Section 4.3.

The paper proposes a way to find communities which are equal in since, as we do not assume that clusters have to be of equal size we use a simpler approach; We generate a laplacian matrix from the subgraph, using the voltage values as the degree matrix, not containing the sink and source and apply the k-means clusterer using the eigenvalues. The k-means clusterer is further explained in Section 3.5.5.

Finally the output is a set of clusters which is either equal to or less than the number of clusters requested. Nodes which could not be clustered are stored in the k^{th} cluster and can be discarded.

3.5.4 Authority Clustering

We developed an approach for clustering the users in the social network based on authorities as previously defined in Section 2.4.5 Definition 1, authorities in a graph are the top k nodes with the highest indegree. The approach is based on the assumption that popular users, meaning users with a high degree of followers, are able to set trends. Thereby attracting like minded users, which we are then able to detect and recommend to one another. A good value for the number of authorities k, can be found by investigating the distribution of the indegree for nodes as shown in Section 2.4.4. The number of clusters generated is equal to the number of authorities set.

Listing 3.3: Authority Clustering Algorithm.

```
1  AuthorityClusterer(G, k)
2  clusters ← ∅
3  authorities ← getTopKInDegreeUsers(G, k)
4  distanceFromAuthority ← ∅
5  for authority in authorities
6     clusters.put(authority, new cluster)
7  for node in graph
8     distanceFromAuthority ← calculateDistances(G,authorities)
9     closestAuthority ← findCluster(distanceFromAuthority.get(node),
          authorities)
10    cluster.get(closestAuthority).add(node)
11 return clusters
```

The pseudo code in Listing 3.3 shows which steps the clustering algorithm takes. *getTopKInDegreeUsers* retrieves the users with the highest indegree from the graph G, the clusters are then created from those authorities. *calculateDistances* returns the distances from the clusters to all nodes. We then for each node find the cluster for which the authority the cluster is based on is closest. Ties caused by a node having equal distance to two authorities are solved by picking the cluster for which the authority has the most indegrees.

Clustering the social graph by authorities could be a useful approach in the Sobazaar application if the premise for this approach holds namely that users who have similar interests want to follow one another and authorities in Sobazaar are setting trends, by trends is meant a meaningful assortment of clothing which in some form fits together, eg. summer shoes and a dress.

3.5.5 Spectral clustering

This section describes the spectral clustering algorithm and k-means and how to convert a social graph to a similarity matrix and a laplacian matrix.

As mentioned in Section 2.4 we have constructed a social graph using connections gathered from the Sobazaar API. For each edge in the social graph we have created a similarity between each node in the graph as given by their similarity based on their likes list, defined in Equation 2.1. Given a set of data points $\{x_1, ...x_n\}$ and the nodes similarity $s_{ij} \geq 0$ between all pairs of data

points x_i and x_j, a clustering of the points is to divide them into groups that are similar to each other, and dissimilar to other points in other groups.

A similarity matrix shows the similarity between all nodes in a graph. A graph $G = (V, E)$ will generate a $|V|^2$ similarity matrix S, and with a graph consisting of 45650 nodes as the Sobazaar data set does, it will generate a similarity matrix which is too large to work with efficiently. The solution to this is to compressed the matrix by exploiting the fact that the matrix is symmetric on the diagonal, meaning without loss of data, information can be stored by only retaining the information on either diagonal side of the matrix. We generate our similarity matrix by comparing all pairs of nodes in the social graph, and retrieving both nodes' like list, i.e. the list of the products they like. Comparing these list of users, using Equation 2.1.

For the purposes of this report we define the laplacian matrix as the normalized Laplacian as described in [31]. Where we define D as the degree matrix A as the adjacency matrix and L is the laplacian matrix before normalization which is defined as $L = D - W$ where W is the weight matrix:

$$\mathcal{L} := D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2} = (\tilde{\ell}_{ij})$$

where:

$$\tilde{\ell}_{i,j} := \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

The laplacian matrix for a directed graph can be constructed by either considering only the in- and out-degree or by considering the edges as a unidirectional edge.

The spectral clusterer uses the k-means algorithm in order to create the final clusters, the main idea behind the k-mean algorithm is to define k centroids, one for each cluster [19]. The placement of these centroids are what initially defines the end clusters. Given a set of observations $(x_1, x_2, \ldots, x_n)$ where each observation is a d-dimensional real vector. K-means partitions the n observations into $k(\leq n)$ sets $S = \{S_1, S_2, \ldots, S_k\}$ while minimizing the sum of squares from within clusters. The objective function is defined as follows:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of points in S_i and the term $\|\mathbf{x} - \boldsymbol{\mu}_i\|$ provides the distance between a point and the cluster's centroid.

The k-means algorithm goes through the following steps:

1. Define the initial groups' centroids. (Picked at random)

2. Assign each entity to the cluster that has the closest centroid. (Using euclidean distance)

3. Recalculate the values of the centroids. (Taking the average of the cluster, and finding new centroids)

4. Repeat steps 2 and 3 until entities can no longer change groups.

Listing 3.4 shows the pseudocode for constructing clusters using the spectral clustering approach using k-means, it is worth noting that the spectral clusterer is used to find the seeds of the clusters and then clusters based on the similarity matrix S.

Listing 3.4: Spectral clustering taken from [31]

```
1  SpectralClusterer(G, k)
2  S ← computeSimilarityMatrix(G)
3  D ← computeDegreeMatrix(G)
4  W ← computeWeightMatrix(G)
5  L ← D − W
6  ℒ ← D^{-1/2} L D^{-1/2}
7  U ← computeKeigenvectors(ℒ, k)
8  C ← {C_1, ...C_k}
9  for i ∈ K
10    y ∈ ℝ^K is the set of column vector of U.
11 for j ∈ y
12    cluster y_j into clusters C, by K-means
13 return C
```

The spectral clustering implementation we use, is from [2] Apache Mahout. We describe in more detail how we create different similarity matrices for spectral clustering in Section 4.2.3. We use spectral clustering instead of solely k-means because of the clusters k-means produce show a compactness of points, where spectral clustering shows the connectedness in the graph, spectral clustering also outperforms k-means on for instance concentric circles.

3.5.6 Conclusion

In this section we described five different clustering algorithms, Social Network Cluster, Betweenness Clustering, Voltage Clustering, Authority Clustering and Spectral Clustering. The Section 3.5.1 describes the social network clustering which allows nodes to be clustered in to more than a single cluster and which besides internal density also requires nodes inside the cluster to have sparse connections to the outside. The parameters which dictate how sparse or dense the connection have to be can be set using the α and β parameters. We concluded that the Social Network Cluster could be a valid approach as we assume that users might follow different styles and are influenced by several users which might be better reflected if they are a part of more than one community which in this case would be equivalent to be part in more than one cluster.

We described in Section 3.5.2 the edge betweenness clustering algorithm which removes a parameterized amount of edges from the graph based on their betweenness, meaning how often the edge is traversed in the set of shortest paths between all nodes. We concluded that the difficulty would be to find the correct amount of edges to be removed especially since the data set has over $800,000$ edges, but it could be a valuable alternative to the social network clustering as it also only uses the network structure and requires no additional information, although edge betweenness has a running time of $O(n \times m^2)$, where m is number of edges and n is the number of nodes, this is not a suitable running time on larger graphs, which resulted in not using it.

A faster clustering algorithm was described in Section 3.5.3 voltage cluster, which runs in linear time. This algorithm clusters a graph as a circuit board one put an electrical current to, and finds closely connected components in the graph.

In Section 3.5.4 we described the Authority Clustering which was an approach developed during the project which tries to model the assumption that users follow authorities and by grouping users into clusters based on their closest authority, we can recommend users which follow the same trend.

Chapter 4

Recommender System Setup

This chapter describes our recommender system model. In Section 4.1 we start by investigating how to properly approach and evaluate a recommender system. In Section 4.2 we describe our model and our experiment methodology. Finally, in Section 4.3 we present a short overview of our implementation.

4.1 Evaluation of Recommender Systems

In our previous research [14] we have described the different ways a recommender system can be evaluated. The main objective we have so far presented is to predict the rating a user would give to an item she has never interacted with before. The dominant methodology to evaluate this task has been the prediction error, described by measures as RMSE or MAE, representing how close the recommender's predictions can get to the original ratings.

Recent interest has been shown in understanding what should be the real objective of a recommender system, such that in 2012 ACM RecSys [1] dedicated a workshop to this problem [4].

In this section we open our research to this direction, especially trying to understand how to deal with the particularities of our system: implicit feedback, sparse dataset, social network and offline evaluation.

In Section 4.1.1 we describe the evaluation problems and the current status of the research. Later in Section 4.1.2 we present alternative approaches that can be used to evaluate these methods.

4.1.1 Evaluation methods

Most of the research in recommender systems has relied on the idea that the prediction of new ratings is the most informative criterion for evaluating the performance. The Netflix prize was also based on this opinion, where the goal was to decrease the RMSE by the 10% [25]. In such scenario the recommender system is tailored to be able to perform the best in the task of predicting

[1] The ACM Conference Series on Recommender Systems

	Actual Positive	Actual Negative
Test Positive	True Positive (TP)	False Positive (FP)
Test Negative	False Negative (FN)	True Negative (TN)

Table 4.1: Truth table describing the evaluation of results.

ratings. For example, popular algorithms based on Matrix Factorization, as those that we have described in Section 3.3, make use of objective functions based on the prediction error to learn a model, as the following

$$\min_{q,p} \sum_{(u,i)\in R} (r_{ui} - \hat{r}_{ui})^2 \tag{4.1}$$

where the objective is to find the parameters that minimize the error between the prediction $\hat{r}_{ui}$ and the original rating r_{ui}.

On the other hand, modern research has started questioning the role of a system based on rating predictions [4] [7]. The motivation is that the prediction error alone is now regarded as a misleading criterion for the evaluation of a recommender system, especially in an offline scenario where no real user feedback is available in response to the recommender's proposals. Nevertheless, the mathematical convenience of using prediction error as a basis for the optimization methods, as the one in Equation 4.1, has made such metrics popular in the literature.

Other approaches for evaluating recommendation systems are instead inspired by Statistics and Information Retrieval evaluation measures such as precision, recall and the F_1-score [9]. To define these measures we start by showing in Table 4.1 a truth table where each cell describes the outcome of a binary classification.

The objective of the classification is to find a relevant recommendation of a new user v to a target user u. If the recommender proposes a user that is relevant for u it is considered a *true positive*, otherwise a *false positive*. If a recommendation is not relevant for u and it is evaluated as not relevant by the recommender system, it is considered a *true negative*, in the opposite case a *false negative*. The relevance of a recommendation must be defined dependently on the application.

We can now define the metrics used in these evaluations, namely:

$$precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{4.4}$$

Because of the divergences in the adoption of a standard evaluation methodology for recommender systems, the interpretation and comparability of the research of different authors is still a difficult task. One of the advantages of using the measures we have just shown is the higher readability of the results, given a definition of what a relevant recommendation is. Precision already shows a percentage of how many relevant items are recommended, rather than a simple numerical value showing the distance of the recommendation from a precise value. For this reason, we have also chosen Information Retrieval measures in the evaluation of our experiments, that we will show in Section 5.1. A wrong definition of relevance would lead to unclear results, a problem we have already faced in our previous research [14].

Recommender systems are usually modeled and evaluated on all the available items. Most of the users are interested in a small subset of recommendations, and recommender systems only present a very small number of the available items. In this kind of scenario it has been shown how IR based algorithms can outperform RMSE based ones [7]. The prediction error looks not being the best estimation of the real results. Other points supporting this argument are:

Rating is not shown In most applications, the recommended items are not shown with the predicted rating, but they are simply proposed to the users.

More difficult problem When the objective is to propose a set of items to the user, the rating is only a way to obtain the user's preference, not the goal.

Small recommendations User analysis has shown how the attention of an average user focuses only on a very few number of proposal, i.e. the first page of results in a search engine.

A different trend is connected to the first two points. A question researchers have started addressing is why to perform the rating prediction in the first place when the final objective is to present a set of items, sorted accordingly to the preference of the user. A more natural solution would be to eliminate the middle step of prediction and simply being able to understand if a list of items is relevant for a user or not.

Regarding the third point, neither RMSE nor IR measures can capture the particular behavior of a user visiting a web page, when her attention is focused on small portions of the data. In this case, a good recommendation at the very top of a list of proposals should be recognized as more relevant than as if the same recommendation was instead put at the end of such list. This idea stems from the field of *Learning to Rank*, and we will describe it in detail in the next section.

4.1.2 Learning to Rank Evaluation

Considering the previous ideas, several researchers [24] [28] [11] have started approaching the evaluation of a recommender system in a different way from what we have previously described. Considering the difficulty of finding relevant recommendations when the data set is sparse and no online feedback is directly available, the focus has moved on evaluating if a recommender system is able to discern between relevant and irrelevant recommendations. To do this the basic approach is, instead of requesting the recommender to find a set of relevant users to propose to a target user u, to directly propose a list of users where each of them has already been identified as relevant or non relevant.

The objective of the recommender is then to sort the list such that all the relevant users appear on the top of the list, while the negative recommendations are moved to the bottom. The closer the list produced by the recommender is to this kind of configuration, the more accurate the recommender system is. This method can be described as in Figure 4.1: starting from a perfect list (left side of the figure) as input for the recommender algorithm, a sorted version of the same list is produced, accordingly to values based on the model on which the recommender has been trained (right side). These two lists are then compared, comparing the occurrences at each position of the lists.

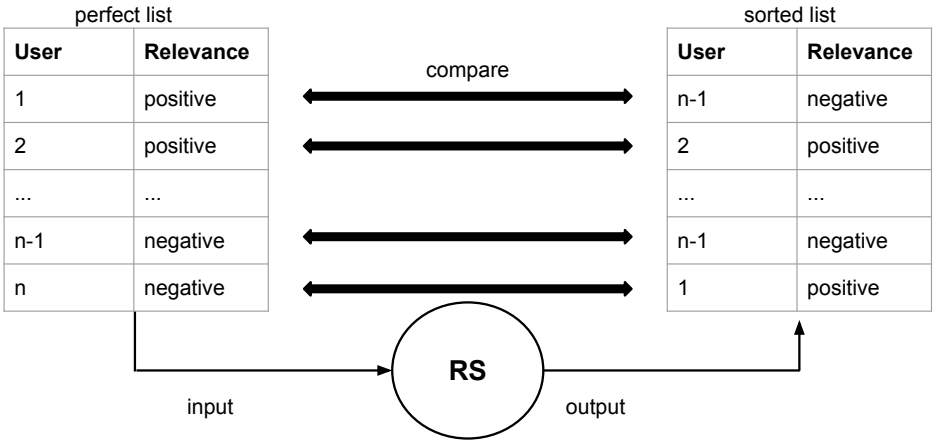


Figure 4.1: Learning to Rank listwise comparison method.

There are two reasons for applying this method. First, we avoid the case in which the recommendations are considered negative only because the target user has never interacted with them, since we already focus on the users with which the target has already connected. Secondly, in a real scenario the user will mostly be interested in a small subset of proposal, hence an evaluation that considers also the position of the relevant items in a list to propose to

the user, is more representative of the real usage of a recommender system.

In the following sections we will formally define the concept of relevance in our application, and we will propose measures that can be used to evaluate this new methodology.

Binary Relevance

Relevance in information retrieval is the concept that some information is more relevant to a user. This concept is useful when evaluating ranked lists as we do in this report. The most intuitive way of defining relevance is by assigning a binary value to the samples, that can then be called Binary Relevance.

In the context of evaluating a social link recommender, an example of a binary relevance score for a user v, considering a target user u for which we want to make a recommendation, can be defined as follows

- 1 - The user u follows the user v
- 0 - The user u does not follow user v

With a definition of relevance, we can then apply the measures we will describe in the following sections.

Cumulative Gain

Cumulative Gain is a technique in information retrieval to evaluate ranked lists with binary relevance associated to them. The main point is to use the information relevance as described in 4.1.2 to compute the cumulative gain at position i the following way: assume that G' is a vector, where each entry is the relevance relating to a specific user with the domain $G'_D = \{0, 1\}$

$$G' = \langle 0, 1, 1, 0, 1, 1, ... \rangle$$

The Cumulative Gain (CG) is then defined as

$$CG_p(G') = \sum_{i=1}^{p} rel_i$$

where p is the length of the ranked list, while rel_i is the relevance score at position i.

Even though this definition can be useful in order to evaluate the gain individually, in order to assign meaning to the position of the information it is necessary to discount the relevance in relation to the ranking.

Discounted Cumulative Gain

Discounted Cumulative Gain (DCG) is based on the following premise:

Premise 1 Information at a lower position is less valuable than at a higher position, as the likelihood of a user finding it will be lower [13].

$$DCG_p(G') = \sum_{i=1}^{p} \frac{2^{r_i} - 1}{log_2(i+1)}$$

where r_i is the relevance for the entry at i. This definition gives a special emphasis on higher relevance at higher ranks, therefore satisfying the premise of more valuable items having a higher value [6].

Normalized Discounted Cumulative Gain

The only difference compared to the previous criterion is that the function is normalized in such a way that a perfect ranking has a value of 1:

$$nDCG = \frac{DCG(G')}{DCG(G'_i)}$$

where G'_i refers to the ideal ranked list.

4.1.3 Conclusion

In Section 4.1.1 we have presented the common problems in evaluating a recommender system, that we also face in our application. Together with these, we propose to apply Information Retrieval metrics, as already adopted in other similar research.

In Section 4.1.2 we describe an alternative evaluation approach, that tries to compensate the flaws of the most commonly used methods.

In the next section we will continue defining the specifications and the design of our recommender system, taking into consideration all the ideas proposed so far in our Data Analysis of Chapter 2, the State of the Art presented in Chapter 3 an the concepts presented in this section.

4.2 Recommender System Model

In order to find the answers for the questions posed in Section 1.1 we have designed an experimental methodology that allows us to evaluate the different techniques we have described so far, and to analyze to what extend they can be successfully applied to the Sobazaar application.

In particular, we remind again our main goals:

Problem 1 What is the underlying cause of one user following another?

Problem 2 How can we make user-to-user recommendations using a social network utilizing implicit feedback and clustering techniques?

Furthermore, our approach is based on the following two assumptions:

Assumption 1 If users, represented as nodes in a social graph, are clustered based on density of edges, the resulting cluster's nodes will have a higher similarity with nodes inside the cluster, than nodes outside the cluster.

Assumption 2 If Assumption 1 holds we assume that users in the future continue with this pattern of following users which are similar to themselves.

In Chapter 2 we have investigated if Assumption 1 holds by applying different clustering algorithms on the social graph, and they do.

Our experiment model combines different features to produce accurate recommendations, and can be outlined as in Figure 4.2. Considering our main goal to propose a solution for the social link problem, the model has been designed to combine both the information from the structure of the social network and the user's activity in the application. We will extract features that will help finding relevant recommendations from these two sources, that we will show in Section 4.2.2. We will also apply different clustering algorithms to the social network, using methods that utilize both the activity of the users and the social graph, and we will describe the techniques we have applied in Section 4.2.3. These three components are used in the recommender system model, and we show the algorithms we have tested in Section 4.2.4. Finally, in Section 4.2.5 we define how we evaluate the recommender system.

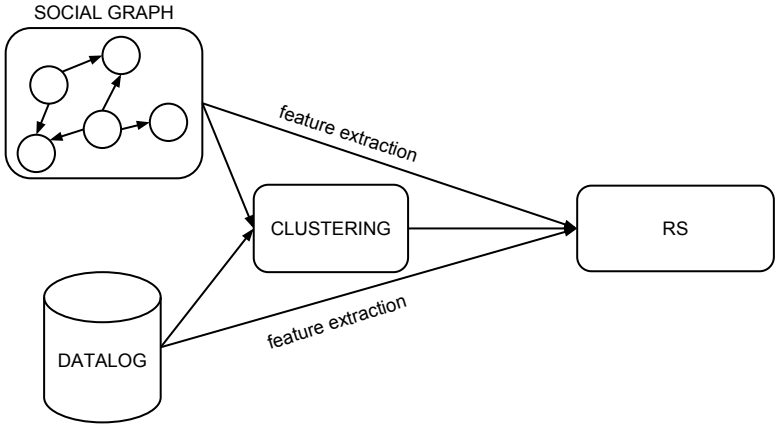


Figure 4.2: General overview of the model: the social network and the data log of the users and the clustering algorithm used to model the recommender system

4.2.1 Social Network Model

The Sobazaar's social network has been modeled as a directed graph, using the classes shown in Figure C.2 in the Appendix. The package allows the construction of different type of graphs. We use the user IDs to construct the nodes of the graph and the follow actions to connect the edges.

A social network is usually defined as a labeled graph, where the labels on the edges are the timestamps on which the connections between two users were created. As we have shown in Section 2.2, the follow actions in our database are associated to the timestamp on which the crawler found the connection. Reconstructing the graph using this data, applying the timestamps to edges, we have discovered that most of the timestamps lie in the same 24 hours time span. This is due to the way the crawler retrieves the data from the API, and because the Sobazaar application closed before the end of our data collection phase.

Because of this, we have simply decided to remove the timestamps from the edges. Nevertheless, this has lead to the impossibility to make a proper evaluation of the social link prediction problem, since we do not have any possibility to analyze the time evolution of the network, crucial for understanding the activity of the users. As described in Section 3.4.1, when evaluating link recommendations a common approach is to split the data set accordingly to a definite moment of time, then learning the recommender system with the data that appears in the first time window and test the proposed recommendation against the data in the second time window. To overcome the absence of the time labels in our graph, we have designed a different way of evaluation, that we will describe in detail in Section 5.1.

4.2.2 Features of the Model

The implicit feedback that represents the activity of the users in our data, does not allow a direct description of their preferences. In [14] we have already approached the difficulty of extracting valuable information from implicit feedback. The approach we follow in this section is similar to a common data mining scenario, we extract domain relevant features that can be used to translate the raw data to a profile of the interests of the users. Since the data we work on in this project is different from the one we had available in [14], we focus now on different ways of mining the data.

From the data log of the users' activities that we have described in Section 2.2 and from the structure of the graph we have selected different features that we have used for both the recommender system algorithms and for the clustering algorithms. These features define either the importance of a user in the network or the similarity between two users.

In Table 4.2 we show a review of the features we have extracted and their focus.

Feature	Description
Item Similarity	Similarity of two users in terms of the number of items they both like or they both use in boards
Adjacency	Similarity that indicates if two users are connected (considering the direction of the edge)
Common Neighbors	Similarity of two users in terms of the number of user that they both have as neighbors
Pagerank	Degree of authority of a user in the network
Activity Rate	Degree of visibility of a user in the network

Table 4.2: The perception features

Item Similarity This feature describes if two users have interacted with the same items, either with a like action or by putting them in their boards. The Item Similarity $IS(u, v)$ of a user u following user v is calculated as:

$$IS(u,v) = \frac{|L_u \cap L_v|}{|L_u|} + \frac{|B_u \cap B_v|}{|B_u|} \tag{4.5}$$

where L_x is the set of items liked by a user x and B_x is the set of items the user x has put in at least one board. The combination of these two values describes how many common items they have expressed a preference for. Although it should be noted that, since we are considering a directed graph, the similarity between users is also calculated considering the direction.

Adjacency This simple feature assigns a value of 1 to a pair of users if they are connected in the graph, again considering the direction of the edges. The Adjacency $AD(u, v)$ is then defined as:

$$AD(u,v) = \begin{cases} 1, & if(u,v) \in E \\ 0, & otherwise \end{cases} \tag{4.6}$$

where E is the set of edges of the graph

Common Neighbors This feature implements the Jaccard's coefficient that we have described in Section 3.4.3, that considers, for a pair of users (u, v) where u follows v, the number of users they have in common in their respective neighborhoods:

$$score(u,v) = |\Gamma(u) \cap \Gamma(v)|$$

where $\Gamma(u)$ has been defined as the set of users in the neighborhood of u.

This criterion describes the tendency of users to group together, if they have common friends. We will use this feature to show if the Sobazaar's users also behave according to this assumption, that will then positively contribute to find relevant recommendations. We have defined our network as directed,

hence we adapt the Jaccard's coefficient to consider the direction of the relationship between two users. Therefore, the coefficient $JC(u, v)$ is calculated as

$$JC(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u)|} \tag{4.7}$$

Since our graph is directed, we consider neighbors to be both the followers and followings of u, because we believe that both connections can contribute in describing the user.

Pagerank As we have already described in Section 3.4.3, this feature indicates the relevance of a user in the network, purely using the structure of the graph. Since Pagerank has already been proven a successful indicator of the quality of a recommendation, for example in the case of web pages in search engines, we assume it can be used to find those users that the network regards as more important.

Activity Rate This feature describes the amount of public interactions the user has in the system, namely the number of product liked and the number of boards published. Similarly to the concept of most popular user, we assume that a user with a high level of activity, compared to the average user activity of the network, will be considered more interesting and *reliable* in their taste by another user, hence it will also be followed with a higher confidence. The activity rate $AR(u)$ of a user u is calculated as

$$AR(u) = \frac{|L_u|}{|L|} + \frac{|B_u|}{|B|} \tag{4.8}$$

where L and B are the set of liked items and published boards in our data set respectively, while L_u and B_u are instead the items liked and the boards published by the user u.

It is worth noting that Pagerank and Activity Rate are *static* values, independent on which pair of users we are considering to evaluate. Therefore, when making a recommendation to a user, Pagerank and Activity Rate will give a value that is not tailored on the user we want to make a recommendation to. We will then use these features alone and in combination with the other ones to understand if they can be used to improve a personalized recommendation.

Focus of the research at this point will be to define what a combination of features is, and how it can be used to extract the preference of the user. Since some of these features go beyond the classic interpretation of similarity between two users, as Pagerank and Activity, we define the concept of *Perception* as a linear combination of these different features, where each of them has equal weight. To allow the combination of the features we normalize each of them in $[0, 1]$ using min-max normalization.

After the initialization of the structures and the computation of these different features, the output of this phase is then the Perception function, that can be used in the following steps by the clustering and the recommender system algorithms.

4.2.3 Clustering Algorithms

To group the users of the social network in clusters we will test the algorithms described in Section 3.5, using the different configuration parameters.

Algorithm	Number of Clusters	Input
Authorities Clustering	5,20,50	G=(V,E)
Social Network Clustering	/	G=(V,E), $\alpha = 0.01$, $\beta = 0.009$
Spectral Clustering	5,20,50	Similarity Matrix:{ Perception }
Voltage Clustering	5,20,50	G=(V,E)

Table 4.3: The different clustering algorithms use in the evaluation.

After having analyzed how these clustering algorithms perform on our data set in Section 2.5 we have decided to apply the algorithms shown in Table 4.3, with the parameters described in the table.

In particular we have tested the algorithms to find $5, 20$ and 50 number of clusters for each algorithm, in order to find which one more accurately represent the structures of the Sobazaar social network. The Social Network Clustering algorithm does not look for a fixed number of clusters but instead uses α and β to find the clusters that fulfil the constraints given by these parameters, as explained in Section 3.5.1.

All the clustering algorithms, beside Spectral Clustering, require as input the graph representing the social network and the number of clusters, since they only consider the structure of the network for finding the clusters. The only difference is Spectral Clustering, for which we have already discussed the algorithm in Section 3.5.5. The algorithm uses the similarity matrix representing the graph as input. To generate this matrix we have considered three possible similarity functions: Adjacency, Common Neighbor Similarity, Item Similarity, that we have defined in Section 4.2.2. Since Spectral Clustering is commonly used on undirected graphs, we define the undirected similarity between two users $S(u, v)$ as the average of the similarity of the two edges (u, v) and (v, u)

$$S(u, v) = \frac{P(u, v) + P(v, u)}{2} \tag{4.9}$$

where P is the Perception function.

Finally, we can generate a *Social Similarity Matrix* $M : |U| \times |U|$, where U is the set of users, describing the similarity between two nodes of the graph only if they are connected by an edge, calculated as follows:

$$M(u,v) = \begin{cases} S(u,v), & if(u,v) \in E \\ 0, & otherwise \end{cases} \qquad \forall u,v \in U \tag{4.10}$$

Using this matrix the Spectral Clustering algorithm will consider both the structure of the network and the similarity between users. In our methods we will evaluate how the Spectral Clustering algorithm performs when using different Perception functions to construct the similarity matrix.

Independently of the clustering algorithm used, the result of this phase is a set of clusters, each containing the set of nodes in that cluster. Each cluster can then be transformed into a directed graph, that can be extracted from the main graph with the following function:

Listing 4.1: Function to extract the subgraph

```
function Graph extractSubGraph(Set nodes) {
    Graph subGraph = Graph;
    foreach (node : nodes) {
        subGraph.addVertex(node);
        for(Node successor : subGraph.getOutEdges(node)) {
            if nodes.contains(successor) {
                subGraph.addEdge(node, successor);
            }
        }
    }
    return subGraph;
}
```

where the input is the set of nodes in that particular cluster. If applied, these graphs can then be used in the next steps by the recommender system instead of the main graph. Our objective at this step is twofold. First of all we want to prove if the users' behavior follows the assumptions defined in Assumption 1 and Assumption 2, leading to the possibility to obtain better recommendations by looking at the clusters as communities inside the social network. Limiting the size of the graph the number of possible connections and nodes to consider decreases, hence the recommender system algorithms can process the recommendations more efficiently. We will show the results for both objectives in the next chapter.

4.2.4 Recommender System Algorithms

The recommender system algorithms we propose are used to recommend new links to the users of the graph. Given a user u the recommender system is able to associate a value to each of the possible new user candidates, then considering those with higher values more relevant for u. Table 4.4 shows the two main type of algorithms we have used.

Recommender System	Clusters	Input
K-NN	Yes/No	Perception
Matrix Factorization	Yes/No	Similarity Matrix, Perception

Table 4.4: Recommendation algorithms which we are considering.

K Nearest Neighbors Recommender This algorithm implements a simple K-NN recommender. Given a graph $G = (V, E)$ and the Perception function, for each user u it considers all the new edges of the type (u, v), such that $(u, v) \notin E$. It then calculates the distance between u and v using the Perception function $P(u, v)$. The recommendations will vary accordingly to the Perception, and the goal of the evaluation is then to find which types of Perception features provide the maximum contribution to the relevance of the recommendation.

K-NN recommender can also use the clusters obtained from the clustering phase. In this case, when recommending new edges for a target user u, it only considers the edges which endpoints lie in the same cluster of u. Since the clusters are implemented as directed graphs, if we consider the entire graph as a unique cluster, the K-NN recommender algorithm can be applied uniformly to both the clustered and non-clustered graph. The pseudocode in Listing 4.2 outlines the function that recommends a new set of links to target users.

Listing 4.2: LinkRecommender, recommendLinks

```
1 function recommendLink(User u, Perception p), returns List of Users
2    Graph g = findCluster(u);
3    List candidateUsers = {}
4    for each node v in g
5       if !g.containsEdge(u,v)
6          candidateUsers.add(v);
7    sort(candidateUsers, p);
8    return candidateUsers;
```

The function *recommendLink* receives the target user u as input, together with the Perception function p. After having found the cluster of the target user (the whole graph in case no clustering algorithm has been applied), it finds the nodes with whom u is not connected yet. Using the Perception function to associate each of these new nodes with a value, the function *sort* will sort them in descending order, then it returns the sorted list of candidates. A subset of the K best candidates can be extracted and proposed to the user.

Matrix Factorization Recommender In Section 3.3 we have shown how Matrix Factorization normally addresses the recommendation by using a matrix containing the users' ratings. In our case, since we are not trying to predict the preference of a user towards items, we change the common User-Items rating matrix to a User-User Similarity Matrix, similar to the one defined in Equation 4.10. In this case we are interested in the direction of the edges to

understand the preference of a user for another user, exactly as we would do for an user-item ratings matrix.

The matrix factorization algorithm, using this similarity matrix, behaves exactly as described in Section 3.3, with the difference that now the factorization will look for latent factors representing the preference of a user for other users.

4.2.5 Evaluation of the Recommender System

In this section we present our experiment methodologies, with the different steps needed to perform the recommendation, following the structure in Figure 4.2.

In Figure 4.3 we show the experiment work flow. In the first step we split the dataset in a training set and test set and constructs the graph of the social network. In step 2 we extract the features from the training set. A clustering algorithm can be applied to the graph, and the recommendations will be made on subgraphs. Although the clustering step can be skipped. Then we can choose the type of recommender system algorithm we want to apply. Finally, it is possible to evaluate the results of this recommender with two different methods. Accordingly to the configuration chosen for all these different components, the system will build all the required structures and apply the chosen algorithms.

We now explain the details in the following paragraphs.

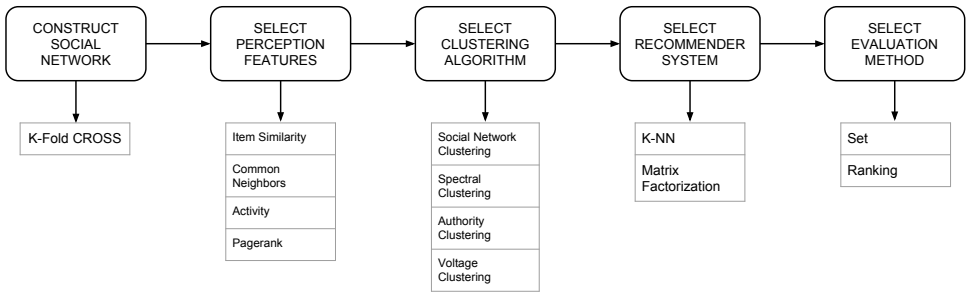


Figure 4.3: The evaluation work flow

Constructing the Social Network

To construct the social graph we collect the data of the follow interactions from the database, namely a set of entries of (u, v), meaning that user u follows v. To do this we query the database requesting the most recent snapshot of the graph with the query from Listing B.1.

At the end of this process we have a data set as described in Table 4.5, that we can use to create the social graph $G = (V, E)$. For each userID in the

user a	user b
u_1	u_2
u_3	u_4
...	...
u_i	u_j

Table 4.5: Follow Set: the data set of the follow interactions between the user a (source) and the user b (target).

data set we create a node $v \in V$, and for each entry in the Table 4.5 we insert an edge in E.

We have already explained in Section 4.2.1 the difficulty of evaluating the social link prediction problem when time data is absent. We overcome this issue by applying K-fold cross validation, a common model validation technique in machine learning and statistics. A generic data set can be randomly partitioned in K subsets of equal size. While one of these partitions is removed from the original dataset to be used as test set, the other $K - 1$ partitions are considered the training set, and can be used to train the model. After the learning process, the removed partition is used to validate the model. The validation step is then repeated K times, where at each iteration a different partition is selected as test set.

In our case we construct the social network from the original dataset. After selecting the parameter K for the number of partitions, we divide the set of out edges of each node in K subsets and, as described before, we remove one of these subsets of edges from the graph. The resulting graph is then equal in the number of nodes, but only a subset of the original edges is present. We call this graph *training graph*, and we will use it in the next steps of the evaluation process to extract the features, to apply the clustering algorithms and to train the model for the recommender system. The subsets removed from the graph are used as test set, and we will use it to validate our predictions.

In a normal social link prediction problem scenario, the edges in the test set would be the new follow actions the user has performed, that can be inferred by her past activity from the training set. In this case, by splitting the data set without considering the time of the interactions, we are not ensured that the edges in the test set represent interactions that have been established after those in the training set, hence the evaluation of such a model could not be fully representative of the actual behavior of the user. This is the reason for using the K-Fold cross evaluation. We repeat the validation step K times, using each time a different partition. When all the iterations are completed, the results of each user are collected and averaged together. With this configuration we smooth the probability that, while splitting the data set in training and test sets, we are over fitting the problem of not considering the time relationships between edges, since that it is not possible to identify which edges have been inserted in the network at a given point of time.

Selecting Perception Features

In this step we use the training graph to extract the features we have described in Section 4.2.2. The only feature that has to be precomputed is Pagerank, that can be calculated offline before the evaluation step. We have already shown the algorithm and the steps required behind this metric in Section 3.4.3. The other functions can be calculated online during the evaluation step. The only computation needed is to find the range of the values to allow all of them to be normalized in the same range, in order to be possible to aggregate them as a linear combination.

Select Clustering Algorithm

After selecting the clustering algorithm and the relevant parameters, the training graph is split in C clusters. Using these clusters we can then extract subgraphs from the social network, consisting of the nodes in these clusters and the edges between them.

Select Recommender System

In this step we train the model for the social link recommender system, by applying either K-NN or Matrix Factorization. Accordingly to the combination of the Perception function and clustering algorithm used, the recommender system is able to define a score for each pair of users.

Listing 4.3: LinkRecommender, getScore

```
1 function Double getScore(User u, User v)
2    return Perception.userSimilarity(u,v);
```

The pseudocode in Listing 4.3 shows the simplicity of the implementation of our recommender system, independently from the type of recommender, from the Perception function and from the clustering algorithm. When the recommender object is constructed it can be used to find the relevance of the recommendation of the user v to the user u.

Evaluation Method

After having trained the model for the recommender system it is possible to evaluate the predictions with two different methodologies. The first one, in Figure 4.4 is a pure recommendation evaluation. For each user u in the graph we ask the recommender system to select the n best recommendations, namely the n users to which u is not connected and that obtain the highest scores from recommender system. We compare these n links to the edges that we have removed from the set of the user u in step a in Figure 4.4. To evaluate the users we require them to have a certain amount of out edges in the test set. We do this to not incur in the case where many of the users with a low activity

in the system will bias our results. In our experiments we will set different values for this threshold, comparing how this will affect the results.

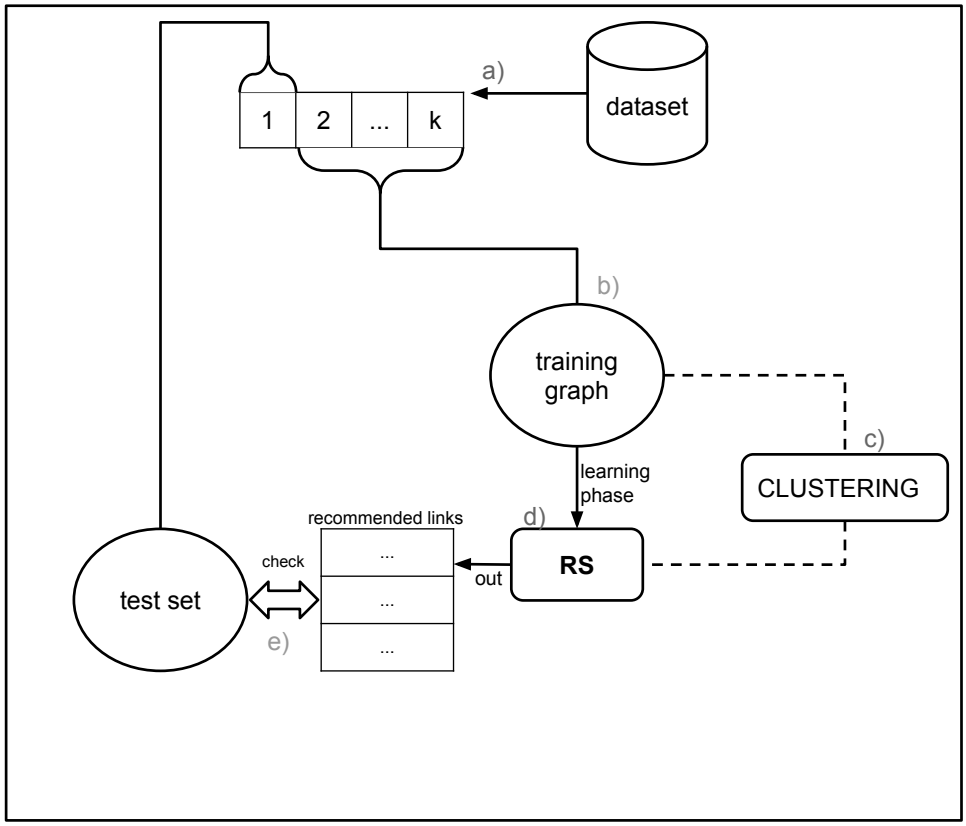


Figure 4.4: Evaluation workflow of a set recommendation.

Table 4.6 shows our definition of what True Positive and False Positive predictions are. We do not define True Negative and False Negative because of the absence of real information that could represent a negative preference of a user, hence it would be impossible to correctly evaluate these parameters. We define a link predicted by the recommender that is not present in the test set as a False Positive. This is an assumption, but leads to a common problem in recommendation system analysis: when using an offline dataset we identify a user v to whom another user u is not connected in the test set as a negative prediction. In reality, this fact only shows that u might never encountered the user v, so it is impossible to say if the recommendation is positive or negative. We have already faced the same problem in our previous research on Sobazaar [14], and our conclusion is that there is no easy answer to this question, so the only consideration to take into account is to carefully read the results as not being fully representative of the real quality of the recommendations produced by the system.

The comparison between recommendations and test sets can be described

	Link in Test set	Link not in Test set
Link predicted	True Positive	False Positive

Table 4.6: Definition of True Positives and False Positives in our evaluation.

in terms of Precision and Recall, that we have defined in Section 4.1.1.

In Section 4.1 we have described alternative methodologies for evaluating a recommender system, that has lead us to the definition of a different experiment methodology, shown in Figure 4.4. The difference in this setup is that we try to evaluate how accurate the recommender system is, at recognizing a good recommendation from a bad one. To do this, for each user u that we want to evaluate we create a list containing the users in his test set followed by a the same amount of users that are disconnected from u in the training graph, namely users that are not in the neighborhood of u. We tag the first set of users of this list as *followings* and the rest as *disconnected*, as in point c in Figure 4.5. This list is then sent to the recommender system that, by using the *getScore* function in Listing 4.3, will associate to each of the entries of the list a score value.

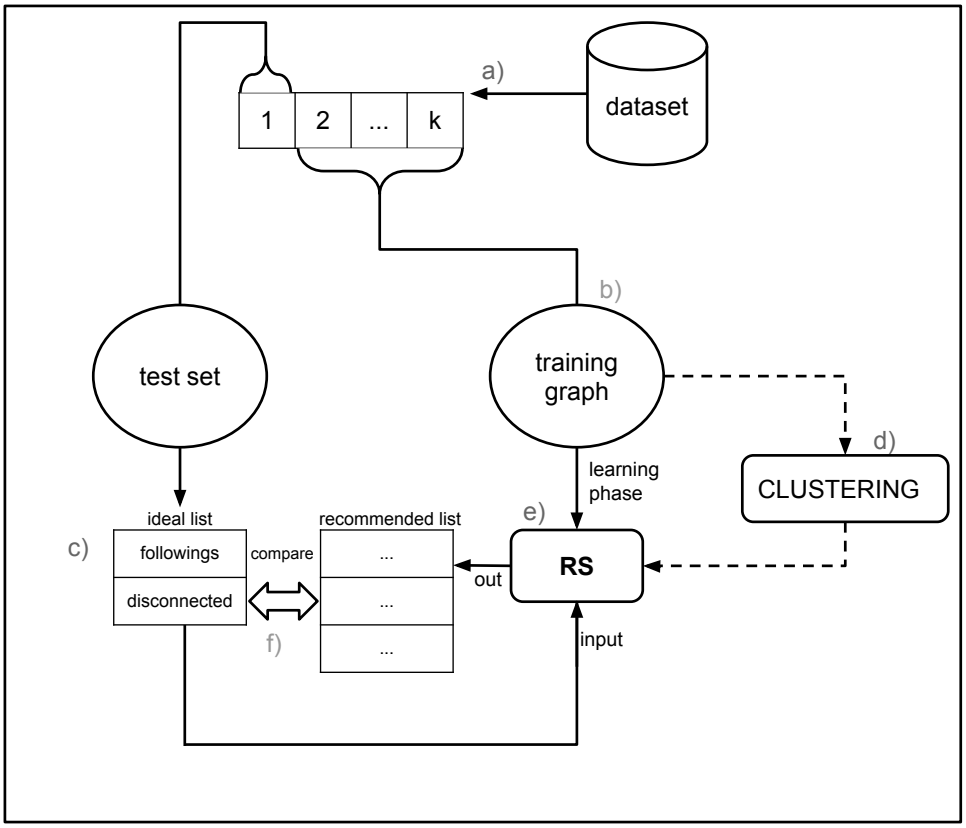


Figure 4.5: Evaluation workflow of a ranking recommendation.

The list can be sorted in descending order, accordingly to this value. By comparing the original list, with all the users labeled as *followings* on top, with the one sorted by the recommender system using Normalized Discounted Cumulative Gain, as described in Section 4.1.2, we can evaluate if the recommender system is able to associate the highest scores to those users that the user u is already following in the test set, compared to those that are instead disconnected.

A perfect recommender should be able to sort the list having all the users with the label *followings* on top, that we assume being a good recommendation.

By using this methodology we also avoid the possibility of recommending users which u has never seen, that will be evaluated as negative recommendations. In this case we instead focus only on the users that we know u is already following, and only analyzing the ability of the system to discern between users that are connected to users that are not.

4.3 Overview of SaLT

We now present our implementation of the model shown in the previous section: SaLT, Social Analytic Link Transformer. The goal of SaLT is to allow the testing of possible approaches on user to user recommendations. In this section we describe the tool and elaborate on the design choices.

SaLT uses several libraries and the most notables are Java Universal Network/Graph Framework (JUNG) [3], the Apache Commons Lang [1] and the Apache Mahout [2].

4.3.1 Packages

SaLT is implemented using Java 8 build 1.8.0. The implementation is separated into several packages sorted by functionality.

SaLT
SaLT.algorithms
SaLT.algorithms.graph
SaLT.algorithms.graph.clustering
SaLT.database_connection
SaLT.eval
SaLT.graph
SaLT.recommender
SaLT.utils
data_model

Table 4.7: Complete list of packages in the SaLT implementation

The Table 4.7 shows the table of packages which exist in the SaLT implementation.

The base package, SaLT contains a main class file API which contains a menu which allows to run the data analysis, the clustering algorithm and the evaluation. It also contains functionality related to displaying small graphs using the jung library.

The largest package in the implementation is the SaLT.Algorithm package. A UML of this package containing all the sub-packages and their respective classes can be found in Figure C.3. The package contains the SocialNetworkClusterer which was implemented using the definitions and specifications described in the paper [21] and previously described in Section 3.5.1. The SaLT.algorithm.graph package contains the classes which are used to analyze the social graph. It contains an implementation for Dijkstra's shortest path and breath first search, a class which contains the implementation of the data analysis, which was made in the Chapter 2, namely SocialGraphAnalysis, it contains functions which return the density, diameter the top k most influential users, the degree of the nodes in the graph, the cluster similarity of the clusters and more.

The SaLT.algorithm.clustering packages contains the clustering algorithms which are available for use and also contains the data structure for the clusters. SaLT uses clustering in order to make recommendation, therefore several clustering algorithms are implemented: betweenness centrality, authority clusterer, voltage clusterer, social network clusterer and spectral clustering.

SaLT.database_connection contains the functionality which is used to access data from a database. The specifications of the database used for this project can be found in Section 2.2. Specifically it allows to mine data directly from the Sobazaar API, storing and accessing it.

The SaLT.eval contains the evaluation process initializing and running the algorithms specified and evaluating them using evaluation methods such as normalized discounted cumulative gain.

SaLT.graph contains the implementation of the graph structure of SaLT. In order to accommodate several different graph structures we implemented different types of edges and graph functionality. The class diagram for the edges is presented in the form of a UML diagram in Figure C.1, there exists functionality for directed and undirected edges. The AffinitEdge is especially useful in the domain of this report, it has a weight associated with it which in this case is referred to as similarity and allows to store the items which both users have in common, which has a direct relation to the similarity if it is also associated by the items users have in common, this allows us to save computation time in a large graph by storing this information directly, instead of recomputing it.

A UML diagram of the class hierarchy can be found in the Figure C.2, as can be seen the social graph in this implementation is modeled after a directed weighted graph, which implements additional functionality in order to create subgraphs, retrieve domain specific values such as getFollowersOfuser and testing functionality.

4.4 Conclusion

In Section 4.1 we have described the different challenges of evaluation a recommender system, and also proposed techniques that can applied to our specific domain.

In Section 4.2 we have presented the general design and objective of our recommender system, together with the specifications of all the different components of which the model is made of.

Section 4.2.5 continues describing the methodology used to apply our recommender system to the Sobazaar data, with the entire worflow process used for evaluating the predictions.

Finally, in Section 4.3 we have given a quick overview of the implementation of the recommender system, with the definition of the classes and the most important features.

In the next chapter we will present our analysis of the results of different experiments in which we have applied our recommender system for addressing the social link prediction problem in the Sobazaar application.

Chapter 5

Evaluation of the Experiments

5.1 Experiments

In this section we will show the results of the different experiments we have performed on the Sobazaar data set. In these experiments we address the social link prediction problem, therefore we limit ourselves to user to user recommendations. We will compare the different algorithms and techniques we have introduced and we will answer the questions we have posed as basis of our research, focusing on the following two ideas which are extensions of Problem 1 and Problem 2:

- Is it possible to extract valuable information from the social network's structure and from the implicit feedback of the users, in order to increase the performance of user to user recommendations?
- Is it possible to to find clusters in the social network, to improve the performances of the recommender system both in terms of execution time and quality of the recommendation.

In Section 5.1.1 we show a comparison between the two main categories of recommender system algorithms, in Section 5.1.2 we show how different configurations of the Perception function affects the recommendations and in Section 5.1.3 we analyze the contribution of the clustering algorithms to the system.

5.1.1 Experiment 1: Recommender System Algorithm Comparison

In this experiment we test both K-NN and Matrix Factorization based on two simple configurations. In Table 5.1 and Table 5.2 we show the different configurations of the two algorithms, according to the parameters that we have

already described in the previous sections. The objective is to understand which algorithm, in a basic configuration performs the best with our data set. As baseline for this comparison we utilize a random recommender. In this experiment we only perform an evaluation based on a set recommendation, considering *precision*, *recall* and *F1*-score obtained from the K-Fold cross validation.

Perception Function	Adjacency/Item Similarity
Followings Threshold	{0,5,10,20,30}
Number of Recommended Links	{20}
K-fold cross	{5}
Evaluation	Classification

Table 5.1: Tested configurations of the KNN recommender systems

User-User Matrix	Adjacency/Item Similarity
Number of latent factors	{10,20,30}
Followings Threshold	{0,5,10,20,30}
Number of Recommended Links	{20}
K-fold cross	{5}
Evaluation	Classification

Table 5.2: Tested configurations of the Matrix Factorization recommender system

We performed the first analysis in order to find the correct parameters for the Matrix Factorization algorithm, namely the number of latent factors to extract. We compared three different numbers of latent factors, $10, 20, 30$. We have trained the model using both an adjacency matrix, results shown in Figure 5.1, and an Item similarity matrix where the values are based on the Item similarity defined in Section 4.2.2. Precision and F_1 score has been used as evaluation metric.

We can see in Figure 5.1 that while using only the adjacency matrix we obtain unclear results, with the F_1-score decreasing when rising the following threshold, and with indefinite difference in performances when changing the number of factors.

In Figure 5.2 we show how by utilizing the Item similarity matrix we obtain both an increase in the F1-score and we can more clearly define 30 features as a better choice. We hence decide to set 30 as the initial configuration for the next experiments.

We compared the two algorithms showing both the performances using the adjacency distance and the Item similarity. We propose that the results are poor in both cases as we lack the necessary data to receive a better score.

By comparing the results in Figure 5.3, we can see how utilizing the implicit feedback, namely the Item similarity, we can increase the quality of the recommendation.

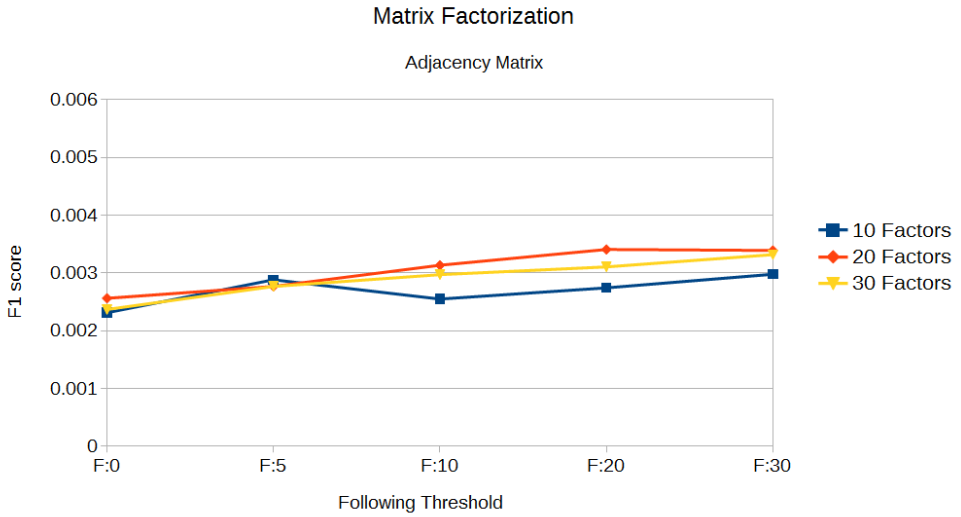


Figure 5.1: Matrix Factorization algorithm based on an adjacency matrix, different latent factors. The x-axis shows the increasing threshold for the number of out edges for a user, the y-axis the F_1 score.

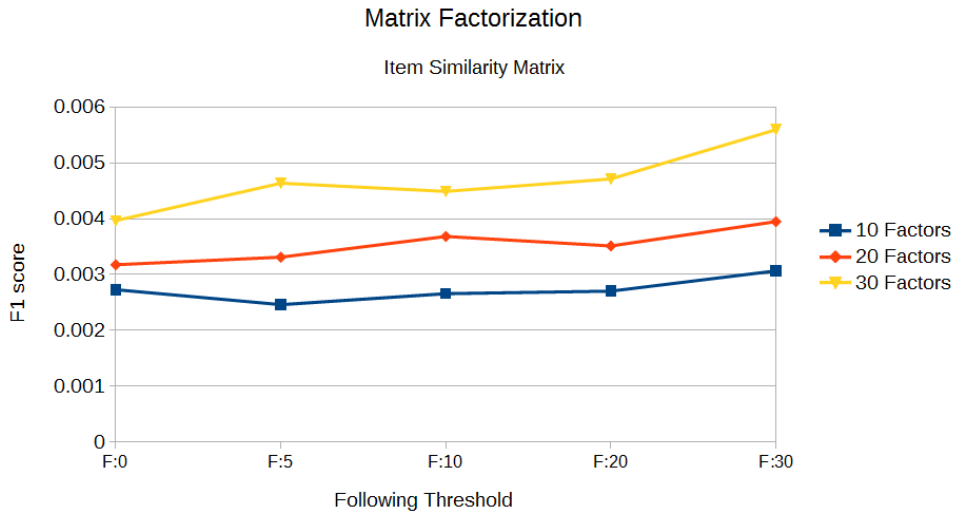


Figure 5.2: Matrix Factorization algorithm based on an similarity matrix with Item similarity, different latent factors. The x-axis shows the the increasing threshold for the number of out edges for a user, the y-axis the F_1 score.

We also observed that it is crucial to set a high following threshold for the users to be evaluated by the recommender system. As expected, the higher the threshold the more accurate the prediction is, we contribute this fact to the Cold Start problem. It is reasonable to assume that a user, to be accurately profiled by the recommender system, needs to have a certain amount of activity. For this reason we set the threshold of out edges a user must have to be considered in the following experiments to 20.

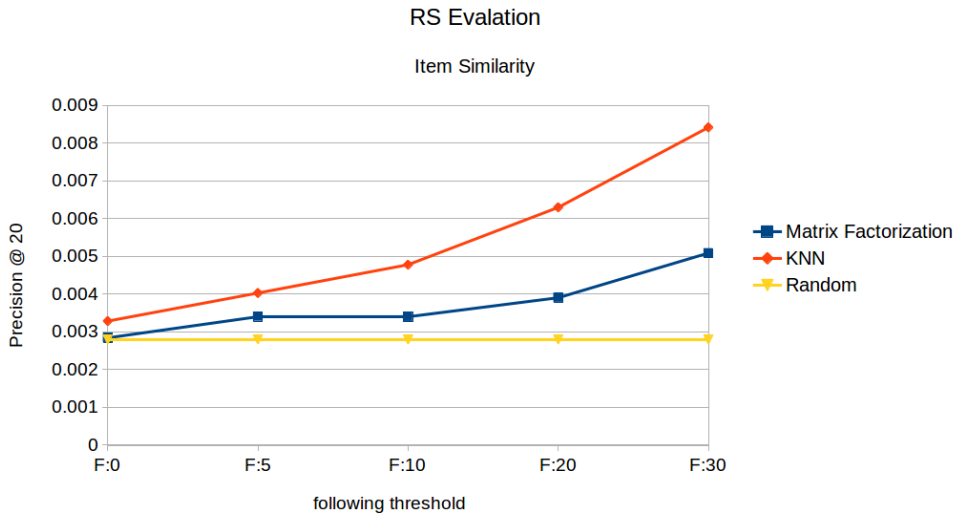


Figure 5.3: Recommender System Algorithms evaluation: K-NN and Matrix Factorization. The Perception function and the similarity matrix are based on the Item similarity.

As we compare the two algorithms in Figure 5.3 we observe how the K-NN algorithm outperforms the matrix factorization. We consider that the reason for this result is the higher complexity of the matrix factorization algorithm, that needs a more accurate tuning in terms of the parameters of the learning process. Since the K-NN allows a more direct evaluation of the configurations we will analyze in the next experiments, we choose this algorithm as our main recommender algorithm.

Finally, we can conclude from this first experiment that there is information present in the implicit feedback, as already hypothesized in Section 2.4.6 we can observe how there is a relation between the preference of the users and the way the connect to each other, although the recommendation results in this experiment are poor. To further investigate to what extent it is possible to use implicit feedback for the social link problem, we continue with our second experiment that we will present in the next section, where we will compare different similarity measures to extract information from the data. The following experiments will be key in gaining an understanding of why users follow each other.

5.1.2 Experiment 2: Defining the Similarity Function

The purpose of the 2nd experiment is to find a similarity measure for which the KNN algorithm achieves the best results.

The Perception features used in this experiments are, *PageRank*, *Item similarity*, *Common Neighbors*, *Activity*, *Popularity*, *PageRank with Item similarity*, *PageRank with Common Neighbors*.

Figure 5.4 shows the results of the experiments for a follower threshold of 20, precision at $k = 20$, meaning that for each user we retrieve 20 recommendations. The metrics used are the F_1 score, *REC* which is the recall and *%POSITIVE* which is the percentage of users for which the recommender is able to give at least one true positive.

We can see that the Common Neighbors similarity and the *PageRank with Common Neighbors* outperform all other similarity measures by one orders of magnitude.

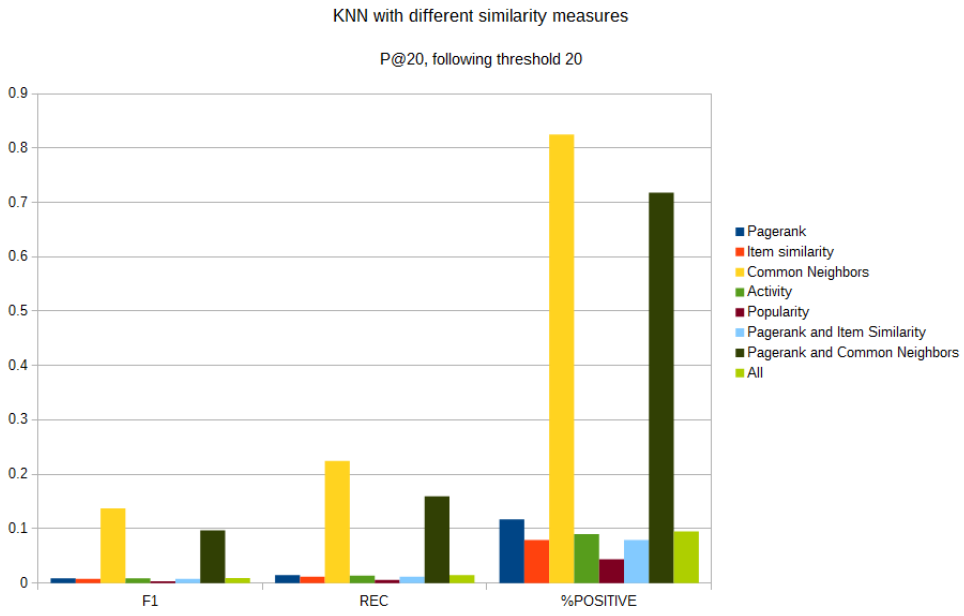


Figure 5.4: F1-score, Recall and percentage of atleast one correct recommendation for the similarity measures.

This result shows that although the Item similarity improve the recommendations slightly, as shown in Section 5.1.1, there is no significant increase when using them as similarity measures for the KNN algorithm. One reason for the poor results using this similarity measure compared to the neighbors can be explained by the lack of data. We mentioned in Section 2.4.6 that we only have 11,413 like lists which corresponds to 27% of the data set.

Figure 5.5 shows the normalized discounted cumulative gain when using the different similarity functions. Since the NDCG does not reduce the score

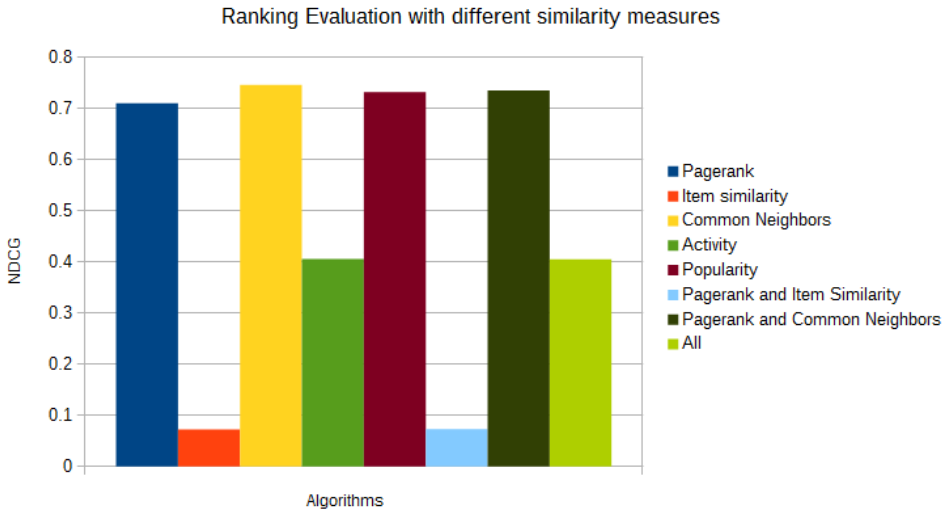


Figure 5.5: The Normalized Discounted Cumulative Gain for the different similarity functions

for recommendations which are not in the test set, and it does not reduce the score for missing users in the test set, we can expect different results from this evaluation method. Figure 5.5 shows that *Pagerank*, *Common Neighbors*, *Popularity* and *Pagerank with Common Neighbors* outperform all other similarity measures. We can expect popularity to perform well in this kind of evaluation as popular nodes naturally have more connections and therefore are likely to have connections to the nodes in the test set.

Based on these results we can see that the similarity based on common neighbors has the best results, however the F_1 score is still at 0.1.

5.1.3 Experiment 3: Clustering

In these experiments we apply the different clustering algorithms described in Section 4.2.3.

Figure 5.6 and Figure 5.7 show the F_1-score with the respective similarity measures for the KNN algorithm with a follower threshold of 20 and precision at 20. We can see that the perception using the Common Neighbor similarity feature outperforms the Item similarity by one order of magnitude. This result is consistent with the previous experiment, Figure 5.4 where we saw a similar increase in the F_1-score by using the Perception with the Common Neighbor similarity feature.

Figure 5.7 shows that spectral clustering and the social network clustering perform worse than the voltage clustering and authority clustering. For the authority and spectral clustering we can observe that the quality of the recommendation decreases with the number of clusters which we contribute to the fact that the clusters these algorithms do not reflect the structure of

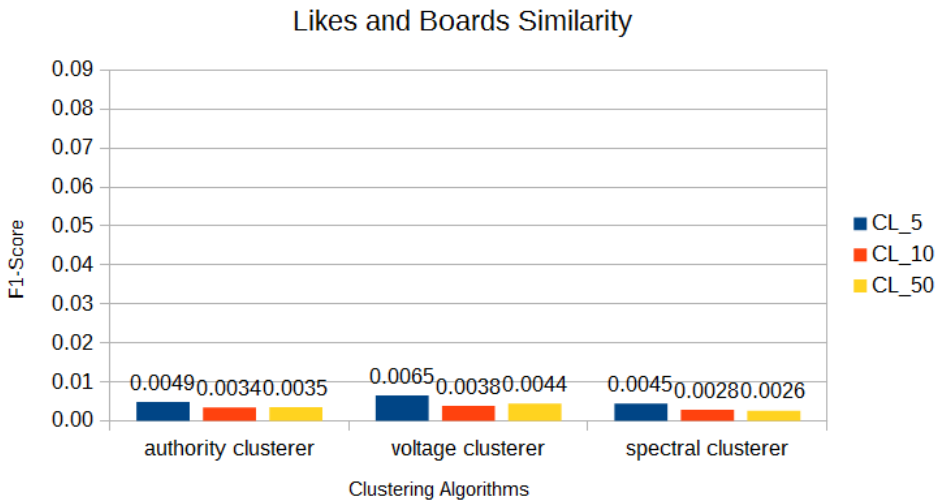


Figure 5.6: F_1 scores for the Authority Clusterer, Voltage Clusterer and Spectral Clusterer with Item similarity measures and KNN recommender, precision at 20 and follower threshold at 20.

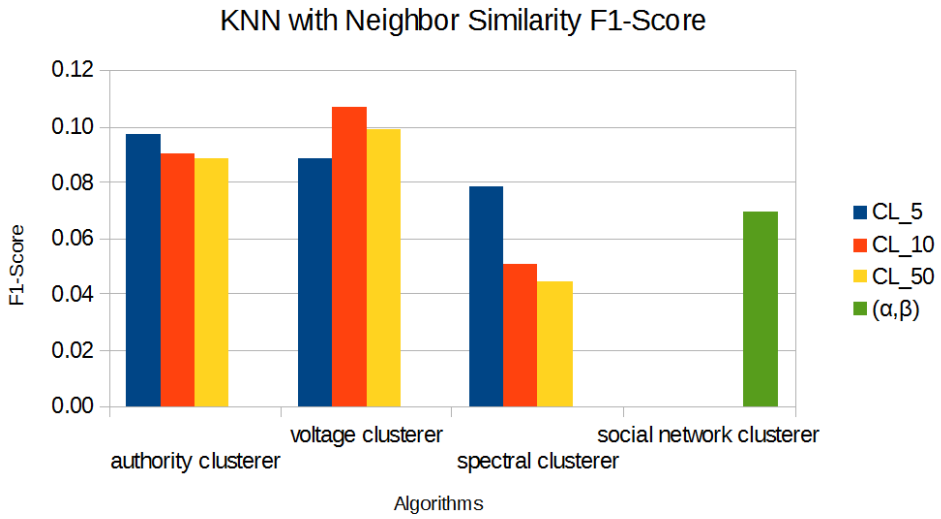


Figure 5.7: F_1 scores for the Authority Clusterer, Voltage Clusterer, Spectral Clusterer and Social Network Clusterer with Neighborhood similarity measures and KNN recommender, precision at 20 and follower threshold at 20, $\alpha = 0.01$ $\beta = 0.009$

.

the community in Sobazaar. The voltage clusterer increases the quality of recommendations for 10 clusters and even on 50 clusters the F_1-score is still higher than with 5 clusters, this observation shows that the voltage clustering approach seems to model the Sobazaar application in the best way out of all four algorithms tested. The data of the results including figures for precision, recall, and the percentage of users for which have at least one true positive, are shown in the appendix Table D.1, Figure D.1,Figure D.2, Figure D.3.

In order to further investigate the quality of recommendations we compare the F_1-score of the best voltage clustering results to the best KNN result without cluster in Figure 5.8.

We contribute this loss of quality of the recommendation to the fact that we have removed important information from the graph by splitting it into clusters.

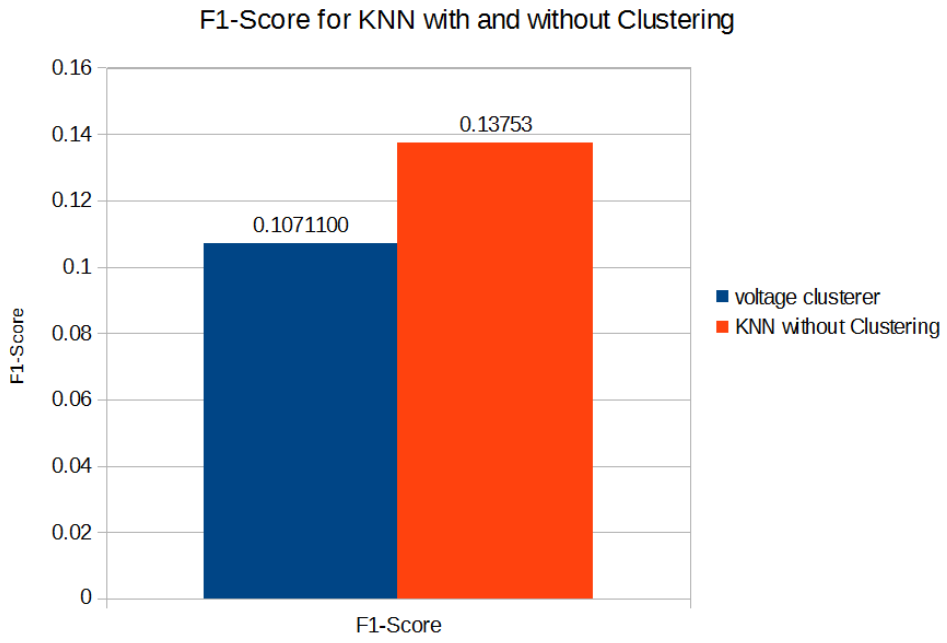


Figure 5.8: Best KNN result with and without clustering on the same configuration: Common Neighbor similarity, following threshold at 20, precision at 20.

Another aspect to take into consideration for our evaluation is if by applying the clustering algorithms we can decrease the execution time of the recommender system. When finding the closest neighbours of a user u, K-NN looks for all the nodes in the cluster of u that are not already followed by this user. If no clustering algorithm has been applied to the graph, all the nodes are contained in the same cluster, namely the entire graph. Therefore, in the worst case K-NN has to calculate the distance between all possible $|V| \cdot (|V| - 1)$ pairs of nodes in the graph. Since some Perception functions, as the Common Neighbors similarity, can be expensive in term of computational

time, the execution of the K-NN becomes unaffordable.

On the other hand, if the users can be clustered in smaller subgraphs the number of pairs K-NN needs to consider decreases. For each node the recommender system calculates the distance between the pairs of nodes that are contained in the same cluster. If we assume that it is possible to uniformly distribute the nodes among the different clusters, then the cost of K-NN becomes $|V| \cdot (|V|/C - 1)$, where C is the number of clusters we extract from the graph. When C increases the execution time decreases, since the number of pairs in each clusters diminishes. The objective is to find the *golden spot* where it is possible to obtain the maximum tradeoff between quality of the recommendations and time performance.

In Figure 5.9 we show the difference in execution time by using different clustering algorithm, while we increase the number of clusters. We again test for $5, 20$ and 50 clusters, and we compare the algorithms using different Perception functions. The graph compares the execution time of the K-NN recommender with and without clusters $\frac{time(K-NN)}{time(K-NN-Cluster)}$. Hence a value of y in the y-axis express that K-NN algorithm runs y times faster by applying the clusters. We have run these experiments on the same machine, to allow an equal comparison of the time costs.

Figure 5.9: F_1 scores for the Authority Clusterer, Voltage Clusterer, Spectral Clusterer and Social Network Clusterer with Common Neighbor similarity measures and KNN recommender.

The first result we can observe is that all the clustering algorithm allow a faster execution of the recommendations, from 2.5 times faster up to 50

times faster than without using any clustering. The increase in performance generally rises with higher number of clusters, since the subgraphs become smaller in the number of nodes. Voltage clustering's performance does not follow this common pattern, considering that the execution when extracting 20 clusters is slower than for 5 clusters. This behavior can be associated to the particularity of voltage clustering, of finding non predictable clusters sizes in each execution.

Although Spectral Clustering is obtaining the best increase in performance, being at least 20 times faster than a non clustered K-NN when using more than 20 clusters, it is not achieving the best results in terms of quality of the recommendations. The reason for this fast execution has already been shown in Section 2.5.1, where we can see that the size of the clusters is indirectly proportional to the number of clusters. Notwithstanding, it is not possible to achieve both good execution times and relevant recommendations for this particular algorithm.

In conclusion, the results of this analysis can be considered positive, since the most successful clustering algorithms that we have tested in our dataset, Voltage Clustering and Authority Clustering, contribute with a faster execution of the recommender system.

5.2 Discussion

In this section we will discuss the results of the experiments and address some of the problems we identified after running all three experiments.

5.2.1 Problems with the Evaluation

We wish to address some problems which we have found in the way we evaluate our recommendations which have been discovered after retrieving the results.

The current evaluation scheme causes our results to be discounted. This is due to the static parameters of the amount of user we recommend for the users we evaluate. The current evaluation scheme has a following threshold of 20 users. Splitting these followings into 5 separate sets in order to perform the 5-fold cross validation causes users which have the minimum number of followings to only have $\frac{20}{5} = 4$ links in the test set. For each 5-fold cross validation iteration we recommend 20 users, this means that for the minimum threshold the maximum precision is given by the number of users in the test set, 4, over the number of recommendations 20: $\frac{4}{20} = 0.2$. The reverse situation also discounts the results; if a user has more than 20 users in the test set for each 5-fold cross validation iteration, the recall will always be strictly smaller than 1, since the formula we use for calculating recall is defined as: $\frac{TP}{CP}$, where CP are the links in the test set. A way to address this would be to dynamically recommend the amount of users which are available in each of the test sets, meaning that for the user with the minimum threshold of 20 followings we

would recommend 4 users. This problem does not affect the comparability between the different algorithms, as the parameters affected are kept static throughout all three experiments.

An additional problem which was not addressed during the experiments but mentioned was that we only have like lists for 27% of all users in the graph. This naturally decreased the precision of the algorithms based on the Item similarity. A way to address the sparsity of the likes in the social graph would have been to introduce an additional threshold which would require a user to have a certain amount of likes.

A final addition to the evaluation scheme would have been to run different perception functions for the clustering algorithm and for the KNN recommendation algorithm. In the current evaluation scheme the same perception is used for both, whether using different perceptions would improve the quality of the recommendation remains to be investigated.

5.2.2 Implicit Feedback for User Recommendations

In Section 2.4.6 we investigated the relation between the social network and item similarity. Table 2.8 showed that there a significantly higher amount of nodes which have greater similarity to those users they are connected with compared to the overall similarity in the graph, 0.1142 and 0.0631 respectively. In Section 5.1.1 we saw an increase in the F_1 score when applying a perception based on the item similarity instead of using perception based on adjacency.

The item similarity performing worse than approaches which use the social graph in Section 5.1.2 most likely stems from the fact that 73% of the users have no like lists associated to them, this fact is further discussed in Section 5.2.1. Finally the results from the experiment 3, Section 5.1.3, specifically Figure 5.6 show that clustering using perception based on item similarity worsens the F_1 score, especially when spreading the sparse information available into clusters which further suggests the sparsity of the like lists being the reason for poor results compared to network based approaches.

In conclusion given the results of the experiments we can not say with certainty whether the results of the perception based on the item similarity are due to lack of data or to the item similarity not containing the information required to identify users for recommendations. Based on the similarity inside the graph Table 2.8 and the results of experiments 1, Section 5.1.1, we conjecture that there is a pattern which supports the theory that the implicit feedback is useful and the poor results are due to the lack of data.

5.2.3 Clustering for User Recommendations

In Section 2.5 we analyze the different clusters generated by the clustering algorithms described in Section 3.5. The analysis showed that it is possible to detect communities for which the item similarity inside the cluster was on average 500 times higher than the original graph, seen in Figure 2.10. It

was also possible to find clusters where the most popular items inside the clusters deviated from the whole graph, seen in Figure 2.11 Section 2.5.2. We also found that the connections inside the clusters is higher for all approaches Section 2.5.3. Whether the clusters can be used to predict future connections is asserted in the Section 2.5.4 by splitting the data set into training set and test set and investigating how many nodes for which there was a connection in the test set were in the same cluster. All approaches had at least a positive percentage of nodes which were connected in the test set clustered into the same cluster.

In conclusion by analyzing the clusters we found that the clustering allows to find trends for communities which deviate from the global popularity, clustering allows to find sub graphs that a more densely connected than the complete graph and clusters have been shown to be able to cluster node together for which edges will be established at a later time. This shows that clustering can be a useful tool for the social link problem.

Experiment 3 Section 5.1.3 showed how the different clustering algorithms performed based on their F_1 score. As the analysis of the clusters suggested the authority and voltage clustering have the best performance out of the four algorithms tested using the perception based on Common Neighbors. Figure Figure 5.8 in Section 5.1.3 compares the best result using clusters and the best result without. The F_1 score is 0.03 lower using the clustered approach. We attribute this decrease in quality of recommendation to the decrease of information which occurs when we separate the data into clusters. Nevertheless, we can expect an decrease in the computational time as shown in Figure 5.9.

The results show that the question whether clustering can be applied in the Sobazaar domain in order to do user to user recommendations comes down to a trade-off between execution time performance and quality given the current results, as the separation of data into clusters can not be done by the techniques used in the setup without a loss of generality.

Chapter 6

Conclusion

The Chapter 1 gave an introduction to the domain and presented the problems which this report addresses, namely:

1. What is the underlying cause of one user following another ?
2. How can we make user-to-user recommendations using a social network utilizing implicit feedback and clustering techniques?

It also contains a discussion of how finding the underlying cause of a follow is interesting in respect to identifying behavior in a fashion domain, allowing to recommend users to one another and how the domain differs from other social networks. It further describes the fact of having a social network as being interesting in regards to the implicit feedback as it allows for a combination of information which might contribute to a better understand of the domain and to enable useful user-to-user recommendations. A final point of the discussion is the motivation to find communities which might reflect both in the social network structure and the item similarity, this in turn motivates the use of clusters for the purposes of detecting such communities.

Chapter 2 gave an overview of the Sobazaar domain from which we retrieved the data using the public API and a description of the data crawled. In order to solve Problem 1 the data was then analyzed first from the point of view of the items found in the like lists and boards, then from the point of view of the social network culminating in an analysis which looked at both the social network and items together, Section 2.4.6, which in turn lead into the analysis of clusters.

Chapter 3 introduced the techniques used throughout the report and defined the common terminology. Section 3.1 defined what a recommendation system is. Section 3.3 gave an overview of an already existing approach to item recommendations using a social graph namely SoRec and concluded that this approach is not applicable to the Sobazaar data set as the data required for generating the social trust is not available. Section 3.4 described the social link problem and elaborated on the difficulty in user-to-user recommendations

and gave an overview of state of the art approaches which address these problems. Section 3.5 described and defined the clustering approaches used in this report.

Chapter 4 described the experimental setup and the recommender system developed during the project. In Section 4.1.1 we described the different evaluation methods, namely precision, recall, F_1-score, and NDCG using binary relevance. Section 4.2 gave a description of the design of our recommender system, and the definition of the experimental methodology which was devised in order to analyze0 the questions posed in the problem definition. Furthermore, we described the different perception features, such as item similarity which uses boards and the like list in order to calculate the similarity, the input of the clustering algorithms used and the settings of the two recommendation algorithms used, K-NN and Matrix Factorization. Section 4.2.5 explained how the data was evaluated using K-Fold cross validation.

Chapter 5 described the experiments conducted and discussed the results in relation to the data analysis. We conducted three different experiments, Section 5.1.1 described experiment 1 for which the purpose was to find the recommendation algorithm which performed better on the data set, which we conclude to be K-NN. Section 5.1.2 presented the results for the K-NN algorithm with each perception feature, for which the Common Neighbors similarity performed best. Finally the Section 5.1.3 described experiment 3 where we clustered the graph based on the four different clustering algorithms and evaluated the results, first against each other, where the voltage clustering with 10 clusters using perception based on Common Neighbors similarity performed the best, and then to the result for the same perception feature, without clustering the graph. The results are shown in Figure 5.8 and show that the K-NN without clustering performed better. Although, we conclude that it is possible to lower the computational cost by using clustering approaches, seen in Figure 5.9.

In Section 5.2 we reflected on the results which were achieved, and it is pointed out that the reason for the low results in the experiments can be attributed to a flaw in the evaluation scheme which discounted the results by not adjusting the number of recommendations to the number of available users in the test set. Section 5.2.2 discussed the results of the item similarity perception feature, and we can conclude that the amount of implicit feedback, namely likes and boards, available makes it difficult to clearly state if the implicit feedback can contribute to the recommendations. We conjecture that there exists a pattern based on evidence throughout the analysis and experiment 1. If this conjecture can be proven with additional data or an additional evaluation as proposed in the section Section 5.2.1, we can with confidence answer Problem 2, as we can combine the implicit feedback with a social network graph by applying a clustering approach on the social network graph using perception based on item similarity. Section 5.2.2 discussed whether clustering can contribute positively in recommending users to one another.

We conclude that it is not possible to find clusters without removing valuable data, using the clustering approaches presenting in this report. Therefore, the choice of clustering comes down to the choice of quality versus execution time.

Problem 1 is addressed in several sections. As we obtain the best results using the perception based on Common Neighbors, we argue that the main source of information about the users can be found in the structure of the social network. Nevertheless, the results shown in data analysis give interesting hints on how communities of users based on the fashion trends seem to appear, when accurately applying clustering algorithms. To fully understand the patterns in the social network and the evolution of these trends additional data is required, both regarding the users' activity and the temporal dimension of this activity, currently missing in our research.

With the availability of such data, our research could deepen in two different directions. On the one hand, how can we define a Perception function that can accurately describe the preferences of a user, by investigating and combining different features. On the other hand, how do we analyze in what way users influence each others' interests, and how to utilize this information for understanding the dynamics of the social network.

Bibliography

[1] Apache commons. https://commons.apache.org/proper/commons-lang/. Accessed: 2015-05-15.

[2] Apache mahout. https://mahout.apache.org/. Accessed: 2015-05-15.

[3] Java universal network/graph framework. http://jung.sourceforge.net/. Accessed: 2015-05-15.

[4] R. 2012. Worhshop on recommendation utility evaluation: Beyond rmse. http://ir.ii.uam.es/rue2012/. Accessed: 2015-02-19.

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

[6] C. Burges, T. Shaked, E. Renshaw, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *In ICML*, pages 89–96, 2005.

[7] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

[8] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):pp. 7821–7826, 2002.

[9] C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In D. Losada and J. Fernández-Luna, editors, *Advances in Information Retrieval*, volume 3408 of *Lecture Notes in Computer Science*, pages 345–359. Springer Berlin Heidelberg, 2005.

[10] P. Graneau. Kirchhoff on the motion of electricity in conductors.(gustav kirchhoff). pages 19–, 1994.

[11] L. Hang. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

[12] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 135–142, New York, NY, USA, 2010. ACM.

[13] J. Kekäläinen. Binary and graded relevance in ir evaluations-comparison of the effects on ranking of ir systems. *Inf. Process. Manage.*, 41(5):1019–1033, Sept. 2005.

[14] B. Kock, D. D. Thøisen, D. Frazzetto, and K. P. Jensen. Creating a recommender system using solely implicit feedback for the fashion portal sobazaar. Cannot be disclosed, is under a non disclosure agreement, 12 2014.

[15] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.

[16] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[17] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.

[18] H. Ma, H. Yang, M. R. Lyu, and I. King. Sorec: Social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 931–940, New York, NY, USA, 2008. ACM.

[19] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[20] S. E. Marcus, M. Moy, and T. Coffman. *Social Network Analysis*. John Wiley and Sons, Inc., 2006.

[21] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. Clustering social networks. In *Algorithms and Models for the Web-Graph*, pages 56–67. Springer, 2007.

[22] A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[23] M. E. J. Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Phys. Rev. E*, 64:016132, Jun 2001.

[24] J.-F. Pessiot, V. Truong, N. Usunier, M. Amini, and P. Gallinari. Learning to rank for collaborative filtering. *ICEIS 2007*, 2007.

[25] N. Prize. Netflix prize.

[26] J. B. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[27] J. Scott. *Social network analysis*. Sage, 2012.

[28] Y. Shi, M. Larson, and A. Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 269–272. ACM, 2010.

[29] R. R. Sinha and K. Swearingen. Comparing recommendations made by online systems and friends. In *DELOS workshop: personalisation and recommender systems in digital libraries*, volume 1, 2001.

[30] Sobazaar. Sobazaar - your daily fashion fix, May 2015. Seen May 25th 2015.

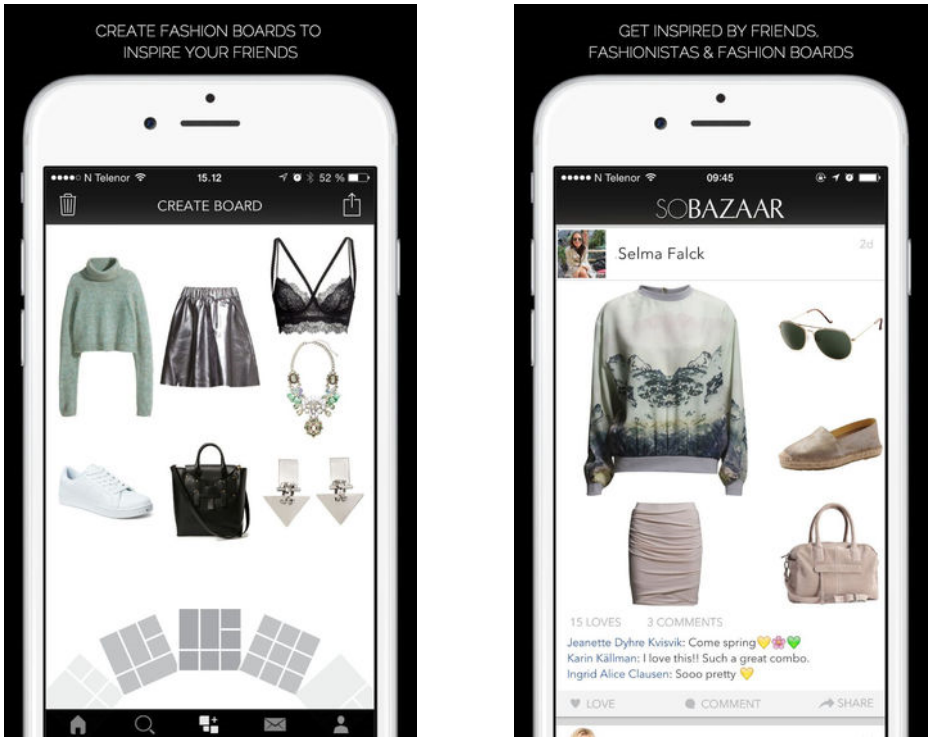
[31] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[32] F. Wu and B. A. Huberman. Finding communities in linear time: a physics approach. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):331–338, 2004.

[33] H. Zhu and B. A. Huberman. To switch or not to switch understanding social influence in online choices. *American Behavioral Scientist*, page 0002764214527089, 2014.

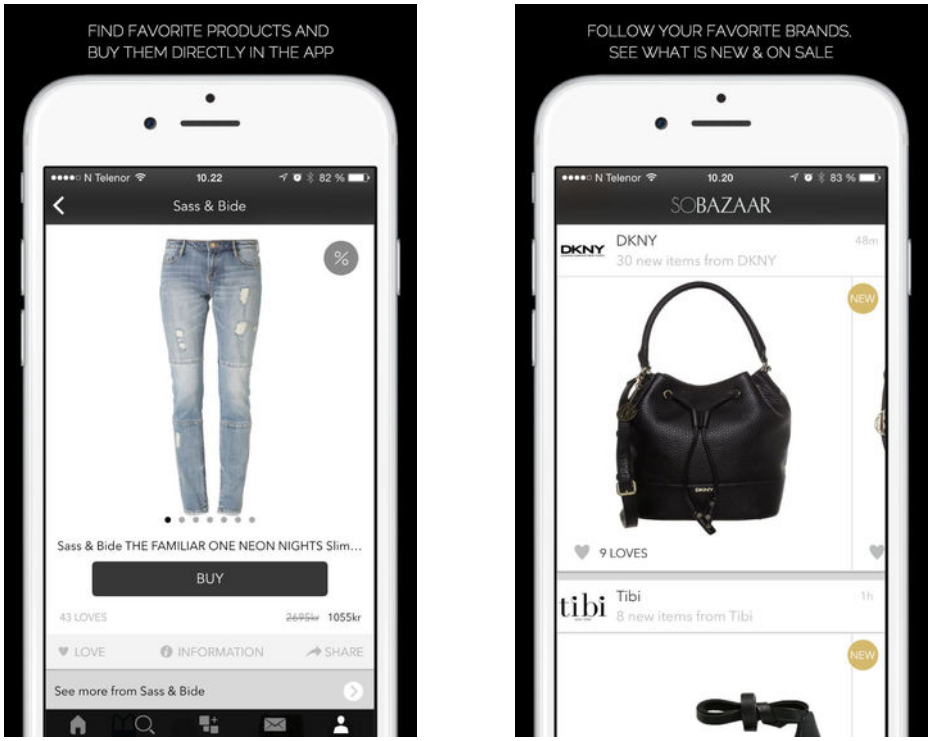
Appendix A

Sobazaar Application



(a) Board Creation (b) Board Published

Figure A.1: Sobazaar App for the iPhone. Taken from [30].



(a) Product Details (b) Feed

Figure A.2: Sobazaar App for the iPhone. Taken from [30].

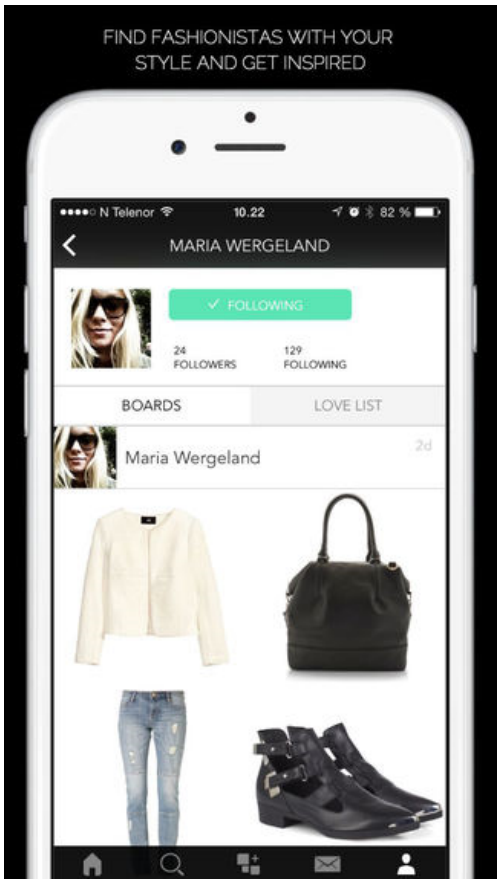


Figure A.3: Profile of a User. Taken from [30].

Appendix B

SQL Queries

Listing B.1: SQL Query to retrieve all follow connections with unfollow

```
1 SELECT follow.user_a_id as user_a, follow.user_b_id as user_b,
2    MAX(IFNULL(follow.added,'1970-01-01 00:00:00')) as follow_added,
3    MAX(IFNULL(unfollow.added,'1970-01-01 00:00:00')) as
         unfollow_added
4 FROM follow
5 LEFT JOIN unfollow ON follow.user_a_id = unfollow.user_a_id
6    AND follow.user_b_id = unfollow.user_b_id
7 GROUP BY follow.user_a_id, follow.user_b_id
8 ORDER BY `user_a`  ASC
```

The SQL query from Listing B.1 retrieves all follow connections and also the unfollow entries. We take the maximum of the added timestamp, i.e. the latest entry. If any of the follow or unfollow added column is NULL, we set the NULL value to be a very early date. The reason for the very early date is to be able to check for which event happened latest. The latest event indicates the current state. For example if the follow.added date is "2015-02-02 12:00:00" and the unfollow.added is "2015-02-03 12:00:00", the unfollow happened after the follow, and we can guarantee that user_a_id does not follow user_b_id anymore. We group by the follow's user_a_id and user_b_id for avoiding duplicates, that can happen if a user has multiple follow and unfollow entries.

Appendix C

Implementation Code

C.1 AuthorityClusterer

Listing C.1: AuthorityClusterer implementation, SearchAlgorithms.java

```
1  public static <T> Map<T, Double> findDistancesToTarget(
       AbstractTypedGraph graph, T target) {
2        Set<T> visitedNodes = new TreeSet<>();
3        Queue<T> frontier = new LinkedList<>();
4        Map<T, Double> distances = new HashMap<>();
5        frontier.add(target);
6        visitedNodes.add(target);
7        distances.put(target, 0d);
8        while(!frontier.isEmpty()) {
9              T head = frontier.poll();
10             double distance = distances.get(head);
11             for(T adj : (Collection<T>)graph.getPredecessors(head))
                   {
12                   if(!visitedNodes.contains(adj)) {
13                         distances.put(adj, distance+1);
14                         frontier.add(adj);
15                         visitedNodes.add(adj);
16                   }
17             }
18       }
19       distances.remove(target);
20       return distances;
21 }
```

Listing C.2: Finding the Distances to clusters, AuthorityClusterer.java

```
1  private <T> T findCluster(Map<T, Double> authoritiesDistances, Map<T
       , Integer> degrees) {
2        T bestAuthority = null;
3        Double maxImportance=  -1d;
4        for (Map.Entry<T, Double> authority : authoritiesDistances.
             entrySet()) {
5              if (authority.getValue() == null) {
6                    continue;
7              }
8              Double importance = (double) degrees.get(authority.
                   getKey()) / (double)(authority.getValue()*authority.
                   getValue());
9              if (bestAuthority == null || importance > maxImportance)
                    {
10                   bestAuthority = authority.getKey();
11                   maxImportance = importance;
12             }
13       }
14       return bestAuthority;
15 }
```

Listing C.3: Finding the Top-k Nodes with the Highest In-degree, SocialGraphAnalysis.java

```
1  public static <T> Map<T, Integer> getTopKInDegreeInfluentialUsers(
       SocialGraph graph, int K) {
2        Map<T, Integer> users = graph.getInDegreeOfUsers();
3        Integer min = null;
4        Integer max = null;
5        for (Map.Entry<T, Integer> e : users.entrySet()) {
6              if (min == null || min > e.getValue()) {
7                    min = e.getValue();
8              }
9              if (max == null || max < e.getValue()) {
10                   max = e.getValue();
11             }
12       }
13       IntegerValueComparator bvc = new IntegerValueComparator<>(
             users);
14       TreeMap<T, Integer> sortedDegrees = new TreeMap<>(bvc);
15       for (Map.Entry<T, Integer> e : users.entrySet()) {
16             sortedDegrees.put(e.getKey(), e.getValue());
17       }
18       Map<T, Integer> topKUsers = new TreeMap<>();
19       int k = 0;
20       for (Map.Entry<T, Integer> user : sortedDegrees.entrySet()) {
21             if (k != K) {
22                   topKUsers.put(user.getKey(), user.getValue());
23                   k++;
24             }
25       }
26       return topKUsers;
27 }
```

Listing C.4: AuthorityClusterer, AuthorityClusterer.java

```
1  public Map<T, Cluster<T>> cluster(int k) {
2        clusteredNodes = k;
3        Map<T, Cluster<T>> clusters = new HashMap<>();
4        Map<T, Set<T>> tmpClusters = new HashMap<>();
5        Map<T, Integer> authorities = SocialGraphAnalysis.<T>
             getTopKInDegreeInfluentialUsers(graph, k);
6        Set<T> authorityKeys = new HashSet<>(authorities.keySet());
7        for (Map.Entry<T, Integer> authority : authorities.entrySet())
              {
8              clusters.put(authority.getKey(), new Cluster());
9              Set<T> c = new HashSet<>();
10             c.add(authority.getKey());
11             tmpClusters.put(authority.getKey(), c);
12       }
13       calculateDistances(authorityKeys);
14       ((Set<T>) graph.getNodesIds()).parallelStream().forEach(node
             -> {
15             Map<T, Double> distancesToAuthorities = new HashMap<>();
16             if (!authorityKeys.contains(node)) {
17                   for (Map.Entry<T, Integer> authority : authorities
                         .entrySet()) {
18                         distancesToAuthorities.put(authority.getKey
                               (), distancesFromAuthorities.get(
                               authority.getKey()).get(node));
19                   }
20                   T closestAuthority = findCluster(
                         distancesToAuthorities, authorities);
21                   if (closestAuthority != null) {
22                         incrementClusteredNodesCounter();
23                         tmpClusters.get(closestAuthority).add(node);
24                   }
25             }
26       });
27 }
```

C.1.1 Class Diagrams

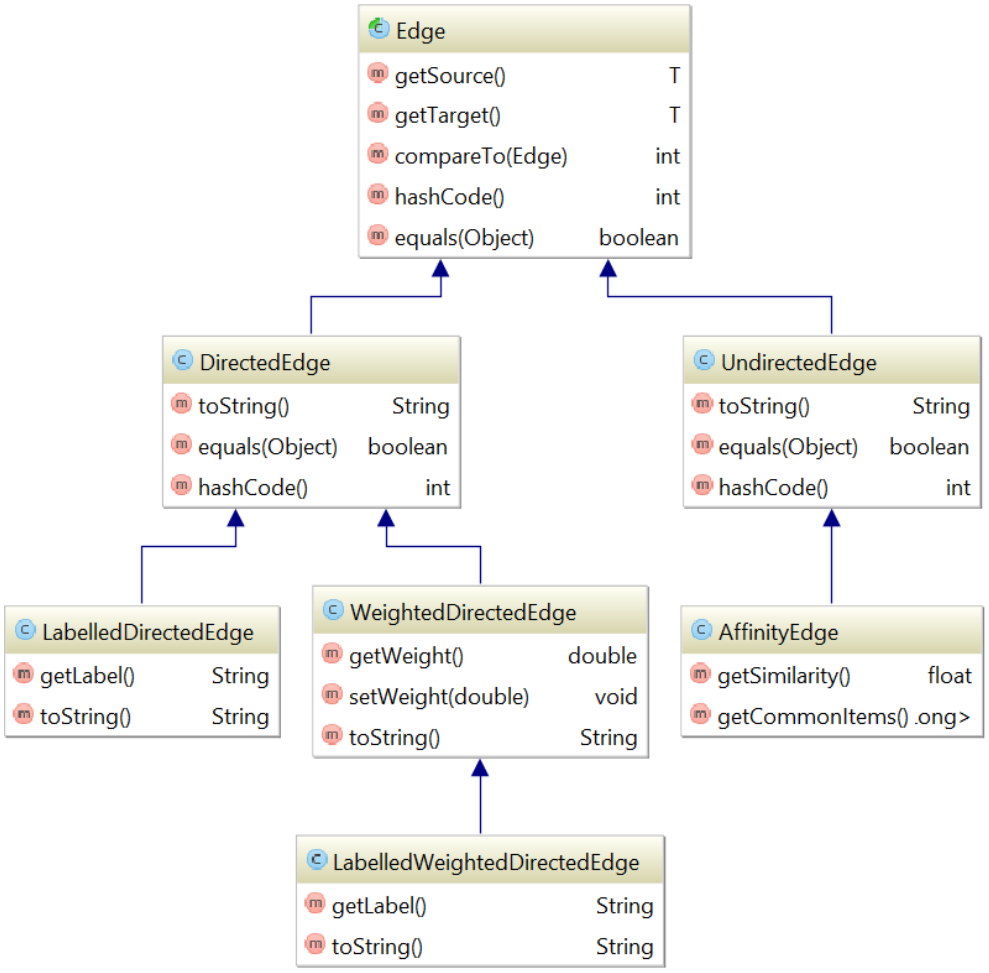


Figure C.1: Class Diagram of the Edge Classes.

C.1.2 Classes

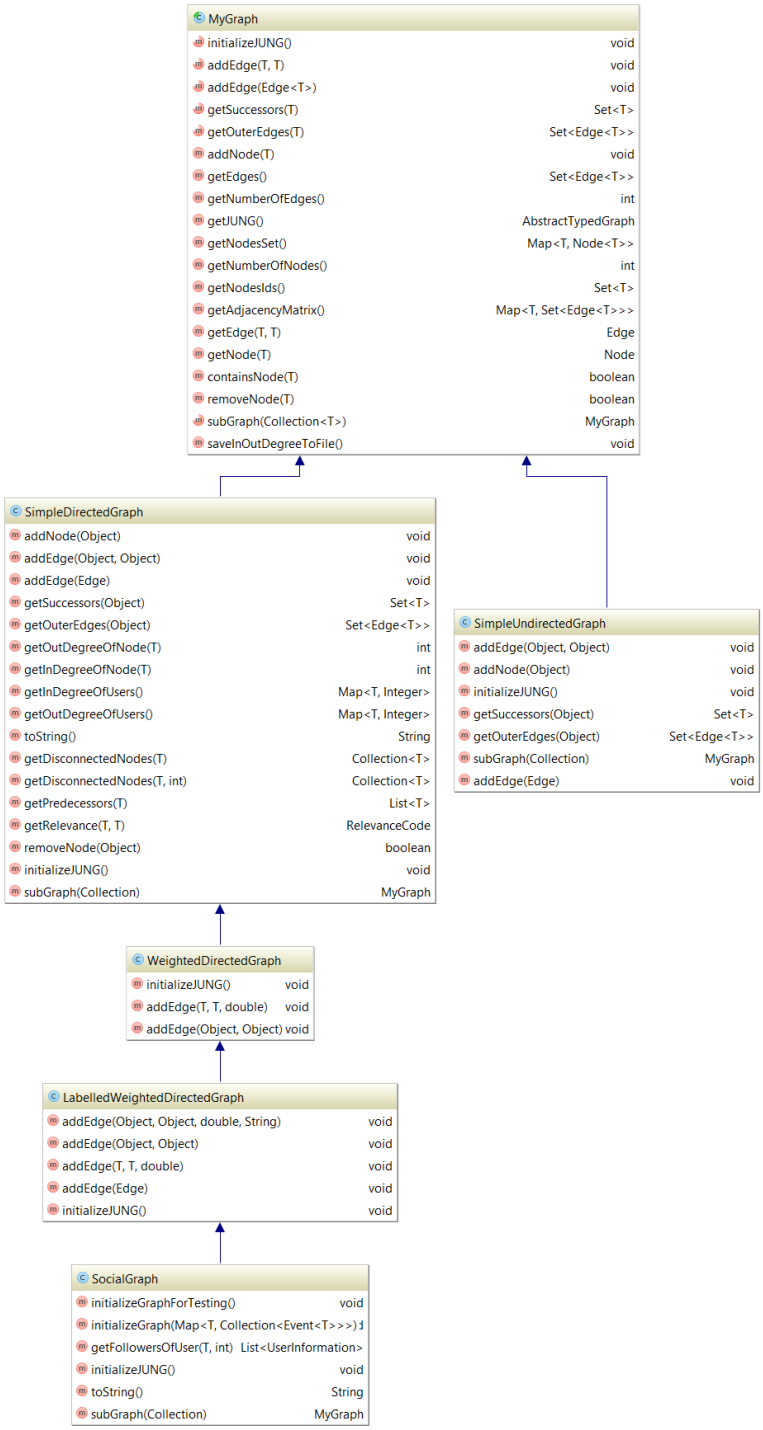


Figure C.2: Class Diagram of the Graph Classes.

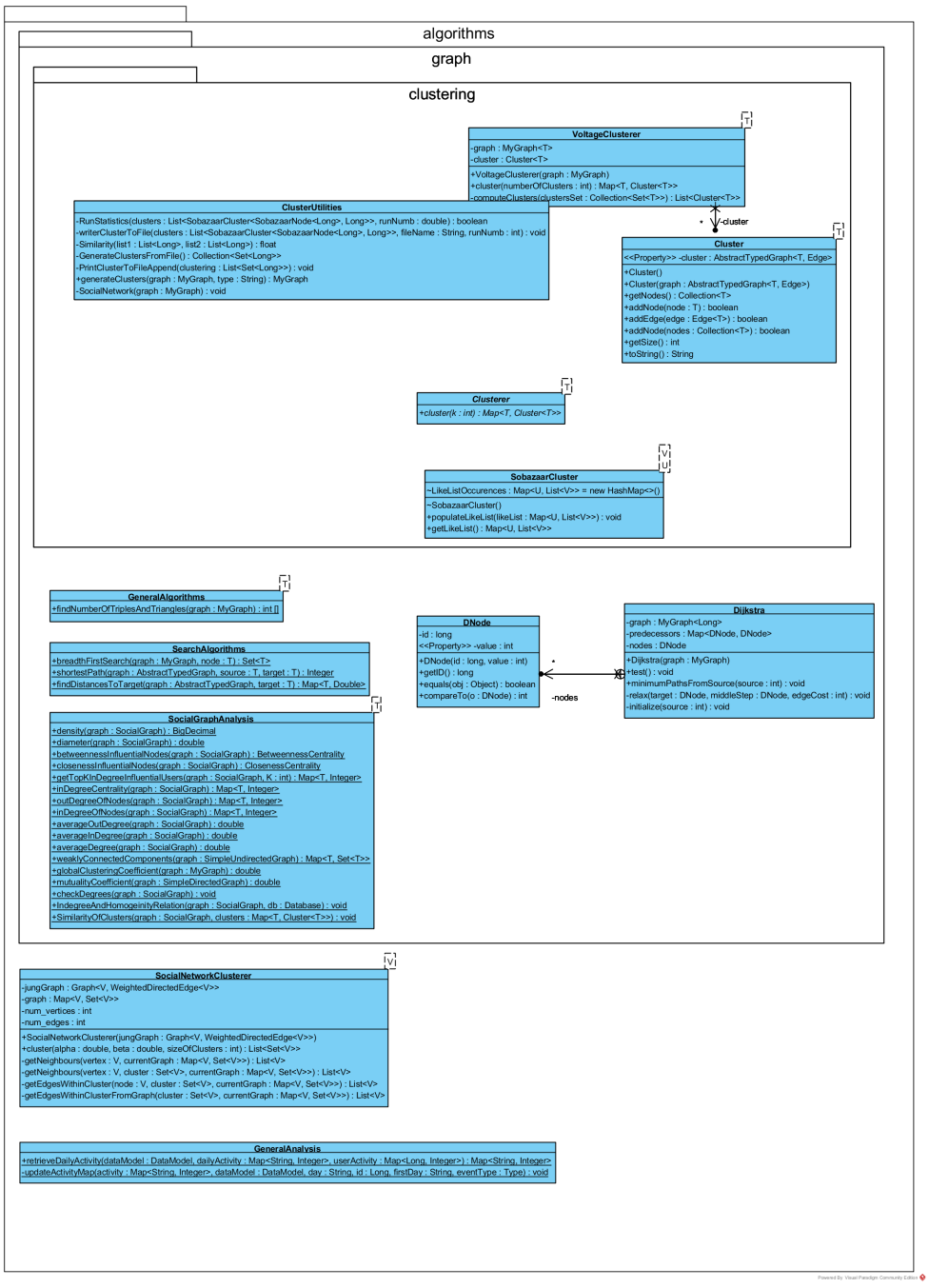


Figure C.3: SaLT.Algorithm and underlying packages including their public functions.

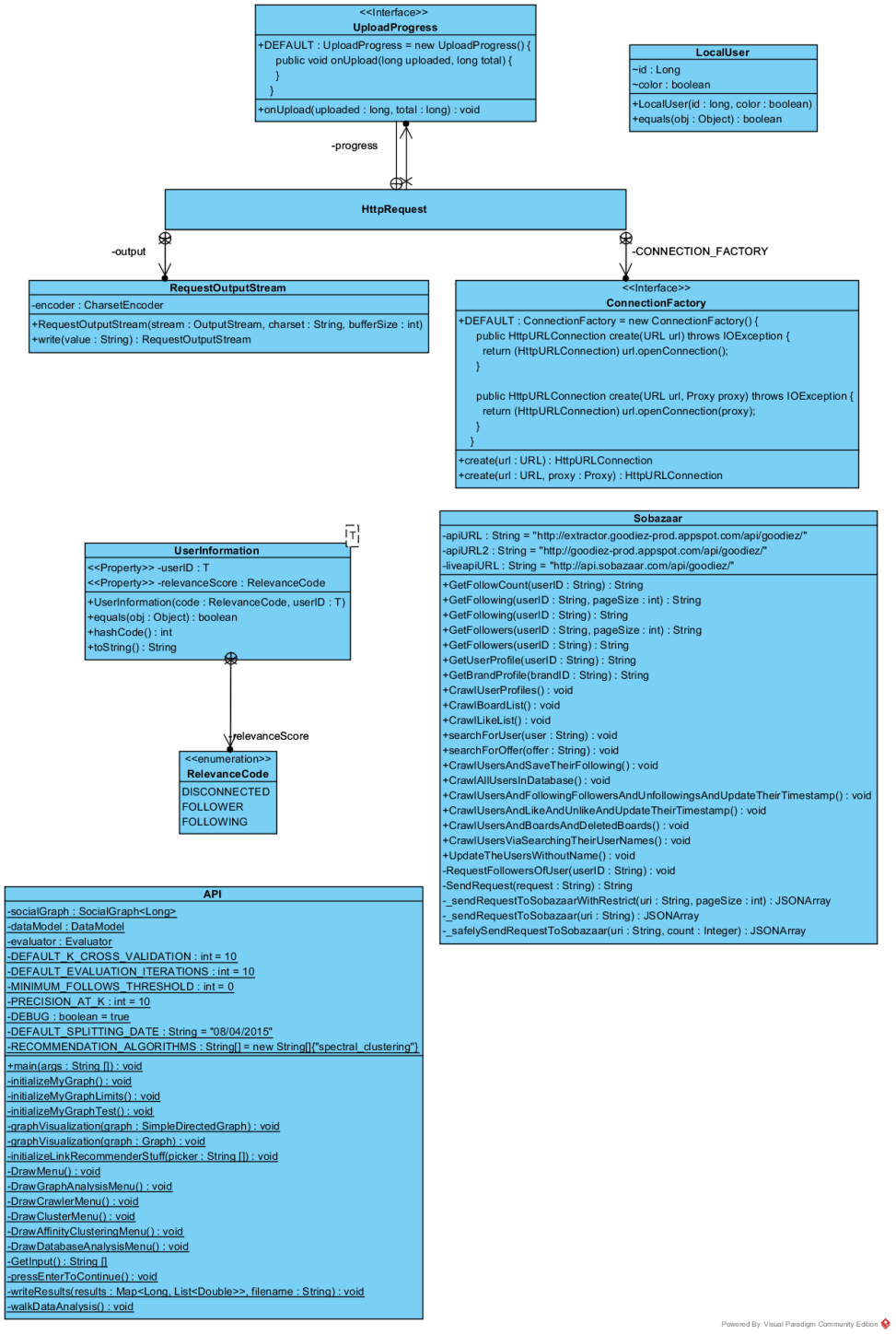


Figure C.4: SaLT package with classes including their public functions.

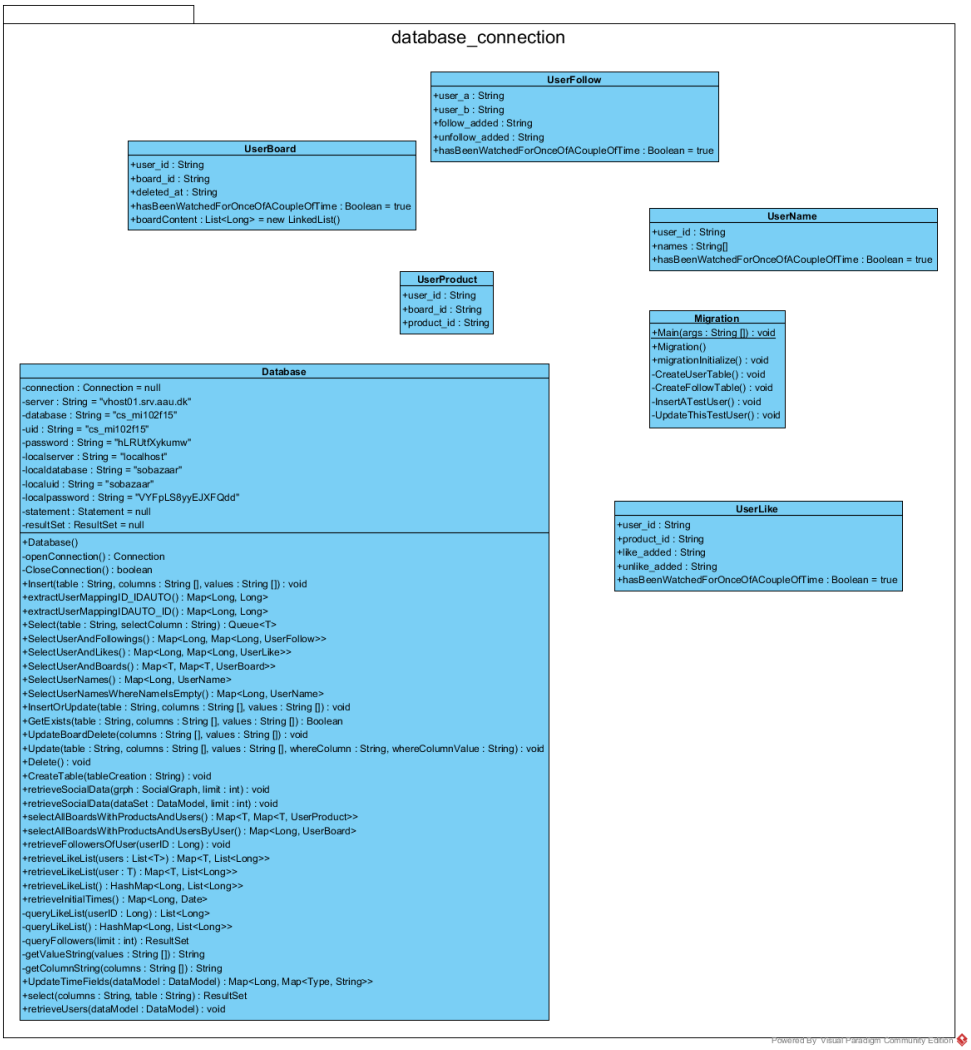


Figure C.5: database_connection package containing the functionality of access information from the database.

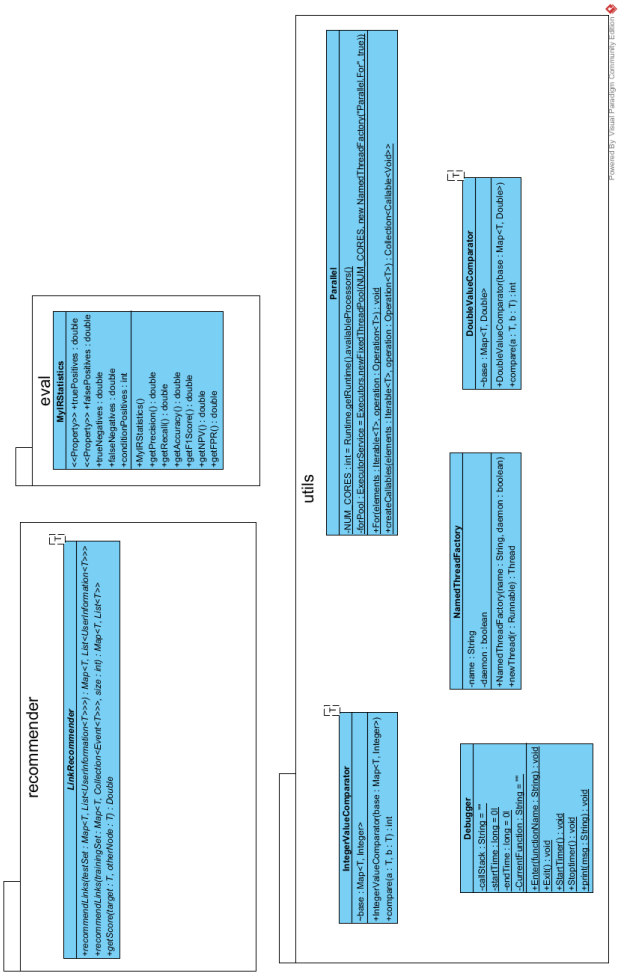


Figure C.6: eval, recommender and util package and their classes.

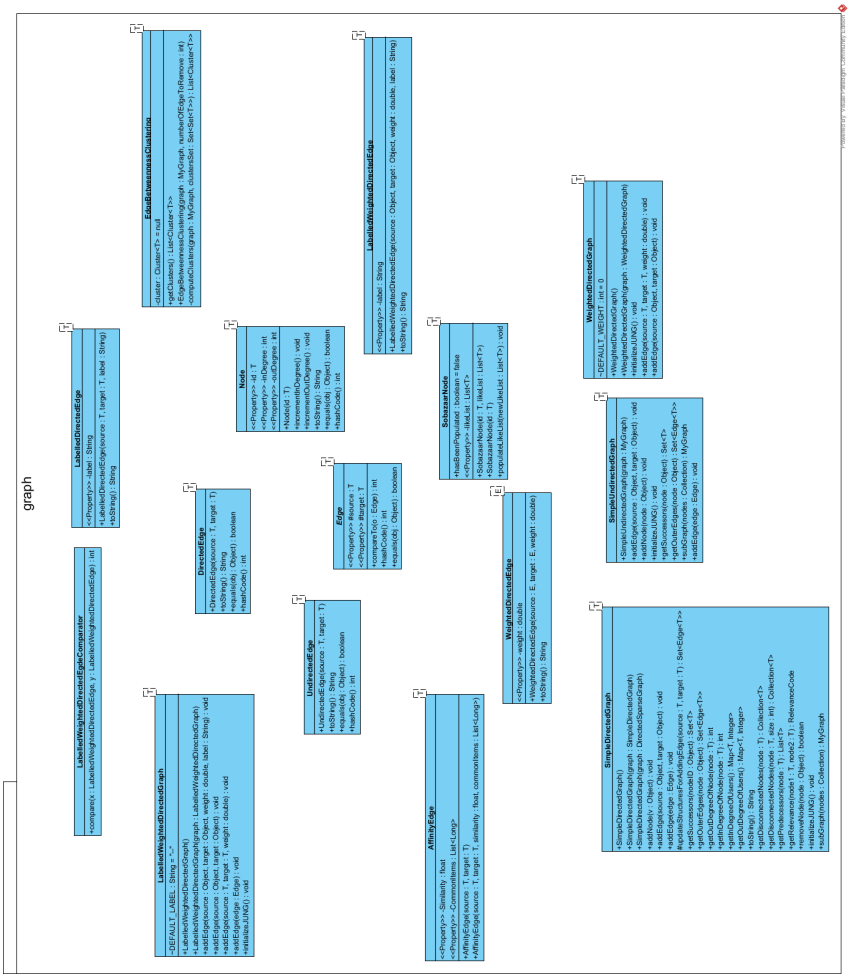


Figure C.7: The graph package containing graph implementation

Appendix D

Results

D.1 Clustering

F1-Score	CL_5	CL_10	CL_50	(α, β)
authority clusterer	0.0974820	0.0900520	0.0888550	-
voltage clusterer	0.0886210	0.1071100	0.0989540	-
spectral clusterer	0.0784910	0.0507640	0.0444220	-
social network clusterer	-	-	-	0.0694290
Positive Recommendations	CL_5	CL_10	CL_50	(α, β)
authority clusterer	0.6898900	0.6534300	0.6473700	-
voltage clusterer	0.5743000	0.7216400	0.6775000	-
spectral clusterer	0.6392400	0.4605000	0.3802500	-
social network clusterer	-	-	-	0.53201
Precision	CL_5	CL_10	CL_50	(α, β)
authority clusterer	0.0741630	0.0682110	0.0673260	-
voltage clusterer	0.0673970	0.0814810	0.0751750	-
spectral clusterer	0.0599240	0.0390860	0.0346030	-
social network clusterer	-	-	-	0.051946
Recall	CL_5	CL_10	CL_50	(α, β)
authority clusterer	0.1579700	0.1467000	0.1446700	-
voltage clusterer	0.1426900	0.1736900	0.1604900	-
spectral clusterer	0.1264800	0.0807910	0.0696680	-
social network clusterer	-	-	-	0.11579

Table D.1: Results of KNN using Clusters for precision at 20, *CL* are the number of clusters, following threshold set to 20 with Common Neighbor perception.

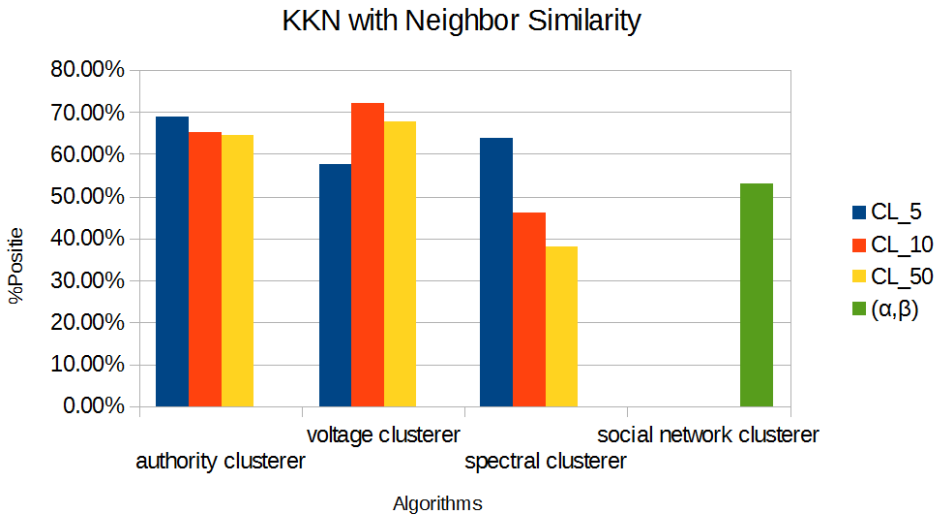


Figure D.1: Result of KNN using Clustering for precision @20, following threshold set to 20 with Common Neighbor perception.

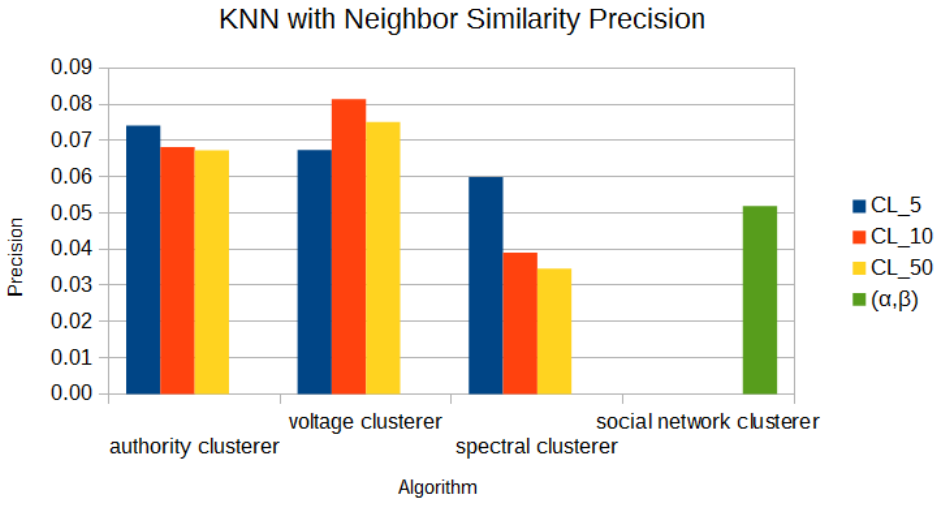


Figure D.2: Result of KNN using Clustering for precision @20, following threshold set to 20 with Common Neighbor perception.

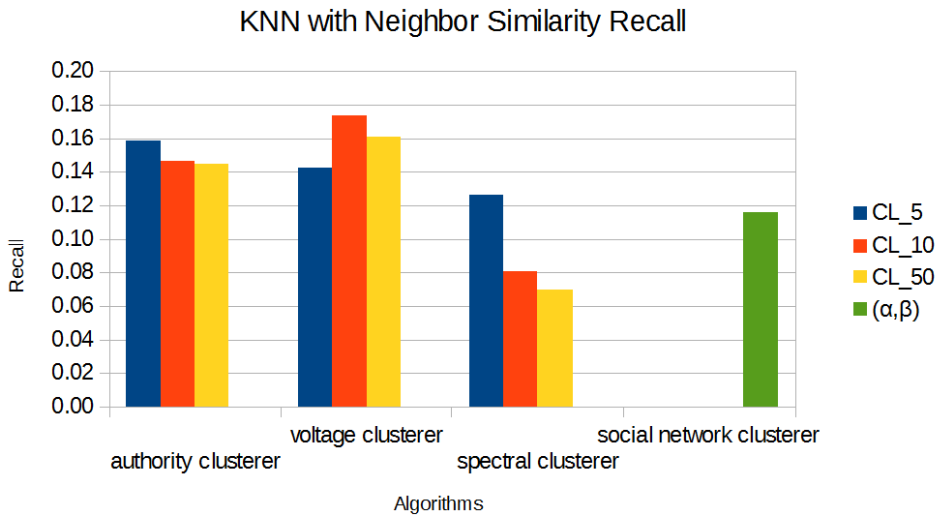


Figure D.3: Result of KNN using Clustering for precision @20, following threshold set to 20 with Common Neighbor perception.