Recommender System Using Social Network Analysis

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Information Technology

under the Supervision of

Dr.Pooja Jain

Assistant Professor, Dept. of IT

By

Akash Srivasatva

Enrollment No: 111465

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "Recommender System Using Social Networks", submitted by Akash Srivastava in partial fulfillment for the award of degree of Bachelor of Technology in Information Technology to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Dr. Pooja Jain

Assistant Professor (Senior Grade),
Dept. of IT,JUIT.

Acknowledgement

I would like to thank everyone that has contributed to the development of this project, which is the final chapter of my Bachelor education in Information Technology at Jaypee University Of Information Technology, Waknaghat, Solan.

Thanks to my supervisor Dr. Pooja Jain, for her guidance and valuable advice during this ongoing development of the project. I would also like to thank my parents who provided me with the opportunity to study in this university and enlightened by life and career.

Date: Akash Srivastava

Table of Content

List of Figures

Abstract

Recommender systems are a hot topic in this age of immense data and web marketing. Shopping online is ubiquitous, but online stores, while eminently searchable, lack the same browsing options as the brick-and-mortar variety. Visiting a DVD store in person, a customer can wander over to the science fiction section and casually look around without a particular author or title in mind. Online stores often offer a browsing option, and even allow browsing by genre, but often the number of options available is still overwhelming.

Commercial sites try to counteract this overload by showing special deals, new options, and staff favorites, but the best marketing angle would be to recommend items that the user is likely to enjoy or need. Unless online stores want to hire psychics, they need a new technology. The field of data mining has a developing field of research in recommender systems, which fits the bill.

“Recommender systems are systems that based on information about a user's past patterns and consumption patterns in general, recommend new items to the user.“

The research in this scope has discovered many methods to get, through the opinion of otherpeople, the relevant items for a specific person. The most of these methods work around theidea of finding similarities in the taste of the people, using Social Network platforms, such as Facebook and Twitter. Then, the prediction for a specific person is based in the opinion of the most similaruser to the person present in the network. This procedure is known as *Collaborative Filtering*.

In the last years, the importance of the social networks is growing and they become relevant forresearch studies. The information provided by users in social networks is a

powerful source ofinformation for the recommender systems in the user's profiles and also in the links that exist inthe structure of the network.

The idea of this project is to analyze different algorithms devised for making predictions to users of the Social Networks, where some use similarity between users and some between items, and then to devise and implement an efficient algorithm which counters the short-comings of the existing algorithms.

1. RECOMMENDER SYSTEMS

1.1 Introduction

Recommender Systems are best known for their use on e-commerce Websites, where they use input about a customer's interests to generate a list of recommended items. Many applications use only the items that customers purchase and explicitly rate to represent their interests, but they can also use otherattributes, including items viewed, demographic data, subject interests, and favorite artists.

These systems use recommendation algorithms to personalize the online store for each customer. The store radically changes based on customer interests, showing programming titles to a software engineer and baby toys to a new mother. The click-through and conversion rates — two important measures of Web-based and email advertising effectiveness — vastly exceed those of untargeted content such as banner advertisements and top-seller lists.

In fact, there are several reasons as to why service providers and e-commerce websites may want to exploit this technology:

- *Increase the number of items sold.*
- *Increase user satisfaction.*
- *Increase user fidelity.*
- *Better understanding of what the user wants.*

1.2 Challenges

E-commerce recommendation algorithms often operate in a challenging environment. For example:

• A large retailer might have huge amounts of data, tens of millions of customers and millions of distinct catalog items.

• Many applications require the results set to be returned in real-time, in no more than half a second, while still producing high-quality recommendations.

• New customers typically have extremely limited information, based on only a few purchases or product ratings.

• Older customers can have a glut of information, based on thousands of purchases and ratings.

• Customer data is volatile: Each interaction provides valuable customer data, and the algorithm must respond immediately to new information.

1.3 Categories

To tackle the above mentioned challenges, two major categories of Recommendation Systems include:

- Content Based Filtering
- Collaborative Filtering

2. CONTENT BASED FILTERING

The system learns to recommend items that are similar to theones that the user liked in the past. The similarity of items is calculated based on thefeatures associated with the compared items.In a content-based recommender system, keywords are used to describe the items; beside, a user profile is built to indicate the type of item this user likes.

For example, if a user has positivelyrated a movie that belongs to the Sci-fi genre, then the system can learn to recommend other movies from this genre.

Fig 2.1: Content Based Filtering

If the user has few purchases or ratings, content based recommendation algorithms scale and perform well. For users with thousands of purchases,however, it's impractical to base a query on all theitems. The algorithm must use a subset or summary of the data, reducing quality. In all cases, ***recommendation quality is relatively poor***. The recommendations are often either too general (suchas best-selling drama DVD titles) or too narrow(such as all books by the same author). Recommendations should help a customer find and discover new, relevant, and interesting items. Popular items by the same author or in the same subjectcategory fail to achieve this goal.

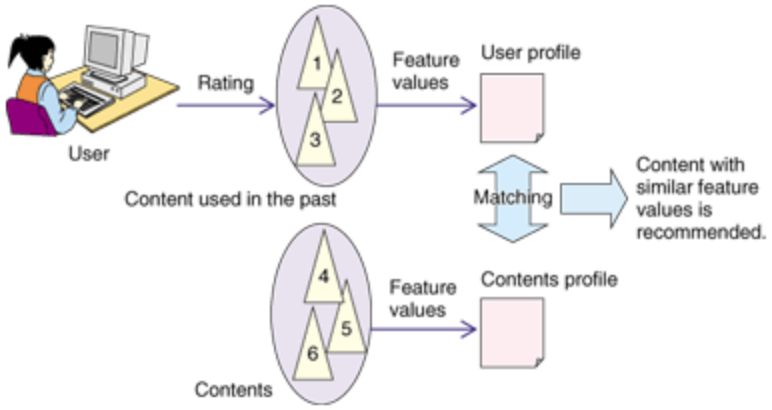

Fig 2.2: Content Based Filtering Algorithm

The main problem with the Content Based Filtering approach is twofold:

1. Domain and problem dependency: for each application area one has to select the appropriate metadata describing the contents the best and ensure its availability. The availability of the right metadata content may not always be guaranteed, for instance, when the sites aggregate contents of different content providers, or products of numerous sellers/retailers. Typical examples are auction or classified sites.

2. Scalability: if the catalog is large (millions of content items) then the selection of the right content requires comparing the user profile with all available content, which may take relatively long time.

3. COLLABORATIVE FILTERING

The collaborative filtering is a technique for recommender systems that generates recommendations using the preferences and tastes given by others users of the system. Thistechnique tries to simulate the collaboration in the real world between users that shareopinions about recommendations and reviews.

In many cases, people have to choose between different alternatives without a complete knowledge of them. In these cases, the people believe in the recommendation of other familiarpeople or people whose opinion is valued by them.

The collaborative filtering systems use this idea, trying to get the users of the system that havethe best opinion about an item for a user (based in his or her taste) and calculate the utilityofthe items for the specific user, using the opinion of the other users.

3.1 Appropriate scenarios to use Collaborative Filtering

The collaborative filtering can be applied in many domains, but to work properly it is better toapply it in scenarios with some characteristics.

The most important is that the evaluation of the items is based on subjective criteria (e.g. movies)or when the items have a lot of objective criteria and they need a subjective weight to choosebetween them (e.g. computers). In these situations, the collaborative filtering is very powerful, but if the recommendations are only based on objective criteria, the collaborative filtering doesnot make sense. It does not mean that the items have no objectives criteria to be evaluated. Itmeans that the objective criteria are very similar, and they differ only in subjective criteria.

It is important too, that each user can find others users with similar tastes, because the rating ofthese similar users will be an important component of recommendations. The taste of the usercannot change a lot in short time, because then the previous ratings of this user do notrepresent his or her preferences and they become useless for predictions. And about the data that is necessary in one scenario to use a Collaborative Filtering

system, it isimportant that there are many ratings per item, to ensure a good inference of recommendationand predictions and that each user rates multiple items. The system needs enough informationto provide recommendations with high quality.

3.2Collaborative Filtering Algorithms

Most recommendation algorithms start by findinga set of customers whose purchased and rateditems overlap the user's purchased and rateditems. The algorithm aggregates items from thesesimilar customers, eliminates items the user hasalready purchased or rated, and recommends theremaining items to the user. Two popular versionsof these algorithms are *collaborative filtering* and*cluster models*. Other algorithms — including search-based methods — focus on finding similaritems, not similar customers. For each of the user'spurchased and rated items, the algorithm attemptsto find similar items. It then aggregates the similar items and recommends them.

3.2.1 Traditional Collaborative Filtering

A traditional collaborative filtering algorithm represents a customer as an *N*-dimensional vector ofitems, where *N* is the number of distinct catalogitems. The components of the vector are positivefor purchased or positively rated items and negative for negatively rated items. To compensate for best-selling items, the algorithm typically multiplies the vector components by the inverse frequency (the inverse of the number of customerswho have purchased or rated the item), making lesswell-known items much more relevant. For almostall customers, this vector is extremely sparse.
The algorithm generates recommendationsbased on a few customers who are most similar tothe user. It can measure the similarity of two customers, A and B, in various ways; a commonmethod is to measure the cosine of the anglebetween the two vectors:

$$similarity(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \bullet \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$

The algorithm can select recommendations fromthe similar customer's items using various methods as well; a common technique is to rank eachitem according to how many similar customerspurchased it.

Using collaborative filtering to generate recommendations is computationally expensive. It is$O(MN)$ in the worst case, where M is the numberof customers and N is the number of product catalog items, since it examines M customers and upto N items for each customer. However, becausethe average customer vector is extremely sparse,the algorithm's performance tends to be closer to$O(M + N)$. Scanning every customer is approximately $O(M)$, not $O(MN)$, because almost all customer vectors contain a small number of items,regardless of the size of the catalog. But there area few customers who have purchased or rated asignificant percentage of the catalog, requiring$O(N)$ processing time. Thus, the final performanceof the algorithm is approximately $O(M + N)$. Evenso, for very large data sets — such as 10 million ormore customers and 1 million or more catalogitems — the algorithm encounters severe performance and scaling issues.

It is possible to partially address these scalingissues by reducing the data size. We can reduce *M*by randomly sampling the customers or discarding customers with few purchases, and reduce *N*by discarding very popular or unpopular items. It is alsopossible to reduce the number of items examinedby a small, constant factor by partitioning the itemspace based on product category or subject classification. Dimensionality reduction techniques suchas clustering and principal component analysis canreduce M or N by a large factor.

Unfortunately, all these methods also reducerecommendation quality in several ways:

- If the algorithm examines only a small customer sample, the selected customers will be less similar to the user.
- Item-space partitioning restricts recommendations to a specific product or subject area.
- If the algorithm discards the most popular or unpopular items, they will never appear as recommendations, and customers who have purchased only those items will not get recommendations.

3.2.2 Cluster-Model Collaborative Filtering

To find customers who are similar to the user, cluster models divide the customer base into many segments and treat the task as a classification problem. The algorithm's goal is to assign the user to the segment containing the most similar customers. It then uses the purchases and ratings of the customers in the segment to generate recommendations.

The segments typically are created using a clustering or other unsupervised learning algorithm,although some applications use manually determined segments. Using a similarity metric, a clustering algorithm groups the most similar customerstogether to form clusters or segments. Becauseoptimal clustering over large data sets is impractical, most applications use various forms ofgreedy cluster generation. These algorithms typically start with an initial set of segments, whichoften contain one randomly selected customereach. They then repeatedly match customers to theexisting segments, usually with some provision forcreating new or merging existing segments. For very large data sets - especially those with high dimensionality - sampling or dimensionalityreduction is also necessary.

Once the algorithm generates the segments, itcomputes the user's similarity to vectors that summarize each segment, then chooses the segmentwith the strongest similarity and classifies the useraccordingly. Some algorithms classify users intomultiple segments and describe the strength ofeach relationship. Classification can be done on the basis of Euclidean Distance:

$$dist(a,u) = \sqrt{\frac{\sum_{\{i \in S_a \cap S_u\}} (v_{ai} - v_{ui})^2}{|\{i \in S_a \cap S_u\}|}}$$

The rating will be the summation of ratings of the item by the users in thecluster divided by the number of users in the cluster:

$$\mu_{ki} = \frac{\sum_{\{u \in C_k | i \in S_u\}} v_{ui}}{|\{u \in C_k | i \in S_u\}|}$$

Cluster models have better online scalabilityand performance than collaborative filtering because they compare the user to a controllednumber of segments rather than the entire customer base. The complex and expensive clusteringcomputation is run offline.

However, recommendation quality is low. Cluster models groupnumerous customers together in a segment, matcha user to a segment, and then consider all customers in the segment similar customers for thepurpose of making recommendations. Because the similar customers that the cluster models find arenot the most similar customers, the recommendations they produce are *less relevant*. It is possibleto improve quality by using numerous finegrained segments, but then online user–segmentclassification becomes almost as expensive as finding similar customers using collaborative filtering.

3.2.3 User-Based Collaborative Filtering

In this method, we predict the user behavior against a certain item using the weighted sum ofdeviations from mean ratings of users that previously rated this item and the user mean rate.

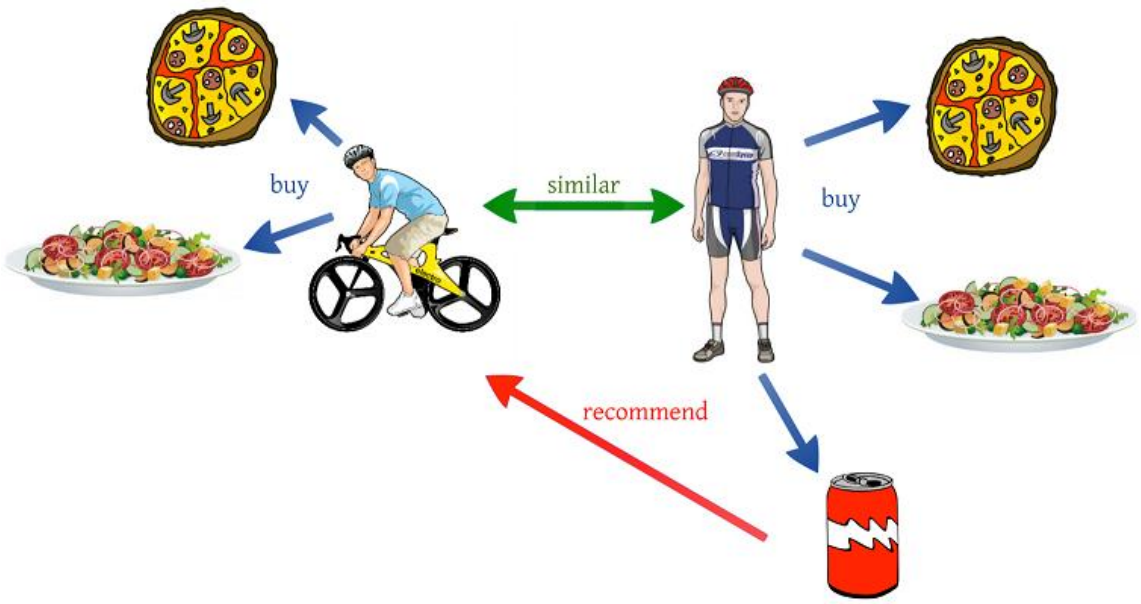

Fig 3.2.3.1: User Based Filtering

First, wecalculate the user mean rate using the following formula:

$$\overline{v_u} = \frac{\sum_{i \in S_u} v_{ui}}{|S_u|}$$

The weight that we previously mentioned can be calculated using Pearson correlation accordingto the following formula:

$$w(a,u) = \frac{\sum_{i \in S_a \cap S_u} (v_{ai} - \overline{v_a})(v_{ui} - \overline{v_u})}{\sqrt{\sum_{i \in S_a \cap S_u} (v_{ai} - \overline{v_a})^2 \sum_{i \in S_a \cap S_u} (v_{ui} - \overline{v_u})^2}}$$

The prediction formula is stated as below:

$$p_{ai} = \overline{v_a} + \frac{\sum_{\{u \in U | i \in S_u\}} w(a,u) \times (v_{ui} - \overline{v_u})}{\sum_{\{u \in U | i \in S_u\}} |w(a,u)|}$$

3.2.4 Item-Based Collaborative Filtering

E-commerce websites extensively uses recommendationalgorithms to personalize its Web site to each customer's interests. Because existing recommendationalgorithms cannot scale to tens ofmillions of customers and products, item-to-item collaborativefiltering, scales to massive data sets and produceshigh-quality recommendations in real time.

Rather than matching the user to similar customers, item-to-item collaborative filtering matches each of the user's purchased and rated items tosimilar items, then combines those similar itemsinto a recommendation list.

Fig: 3.2.4.1: Item Based Filtering

To determine the most-similar match for a givenitem, the algorithm builds a similar-items table byfinding items that customers tend to purchasetogether. We could build a product-to-productmatrix by iterating through all item pairs and computing a similarity metric for each pair. However,many product pairs have no common customers, and thus the approach is inefficient in terms ofprocessing time and memory usage.

It's possible to compute the similarity between twoitems in various ways, but a common method is touse the cosine measure we described earlier, in whicheach vector

corresponds to an item rather than acustomer, and the vector's *M*dimensions correspond to customers who have purchased that item.

This offline computation of the similar-items stable is extremely time intensive, with $O(N^2M)$ asworst case. In practice, however, it's closer to$O(NM)$, as most customers have very few purchases. Sampling customers who purchase best-sellingtitles reduces runtime even further, with littlereduction in quality.

Given a similar-items table, the algorithm findsitems similar to each of the user's purchases andratings, aggregates those items, and then recommends the most popular or correlated items. Thiscomputation is very quick, depending only on thenumber of items the user purchased or rated.

Item-Based collaborative filtering can be further enhanced by the following techniques:

Significance Weighting: It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping) items. Theseneighbors based on a small number of overlapping items tend to be bad predictors. One approach to tackle this problem is to multiply the similarity weight bya *Significance Weighting* factor, which devalues the correlations based on few co-rated items.

Default Voting: An alternative approach to dealing with correlations based on very few co-rated items is to assume a default value for the rating for items that have not been explicitly rated. In this way we can now compute correlation using the union of items rated by users being matched ($I_a \cap I_u$), as opposed to theintersection. Such a *default voting* strategy has been known to improve Collaborative Filtering.

Inverse User Frequency: When measuring the similarity between users, items that have been rated by all (and universally liked or disliked) are not as useful as less common items. The notionof *inverse user frequency*, which is computed as $f_i = \log n/n_i$, where n_i is thenumber of users who have rated item i out of the total number ofn

users. To applyinverse user frequency while using similarity-based CF we transform the originalrating for i by multiplying it by the factor f_i. The underlying assumption of this approach is that items that are universally loved or hated are rated more frequently than others.

Case Amplification: In order to favor users with high similarity to the active user,*case amplification* this transforms the original weights in

$w'_{a,u} = w_{a,u} \cdot |w_{a,u}|^{\rho-1}$

where ρ is the amplification factor, and $\rho \geq 1$.

4. PROJECT IMPLEMENTATION

The application segment developed and mentioned further is for generating recommendations for movies. For the development, the MovieLens dataset is being used consisting of info about 100,000 ratings (1-5) from 943 users on 1682 movies.

As we discussed earlier the traditional user based collaborative filtering will firstly calculate co-relation/similarity between each pair of user that will amount to a 943*943 computations. Further predictions for each unrated movie for each user will amount to approximately 1600*943 computations. As we can observe that this is a lot of work load for the application program to handle.

Initial test on the above application shows us that to generate an overall recommendation to all the movies in the dataset will take about 20-30 minutes. So, to improve the time complexity I am going to use the following devised algorithm for real-time recommendations for a specific genre.

Part-I: Genre specialized - User-Based Collaborative filtering:

In this approach the traditional "User-Based Collaborative filtering" has been modified to generate *Genre* specific recommendations. The algorithm asks the user for a specific genre on which he/she wants to get recommendations on, and then only calculate the predictions for the movies belonging to that category. This approach provides a speed-up of nearly 5, i.e., it is 5-times faster than the actual traditional approach but with a *trade-off* that it will generate specialized recommendations for only the selected category.

Part-II: Social Network Integration

The part-I of the application involves a considerable amount of interaction of the user with the application. This interaction time can be reduced as well as the predictions can be made more personalized using social network integration.

4.1 Application Overview

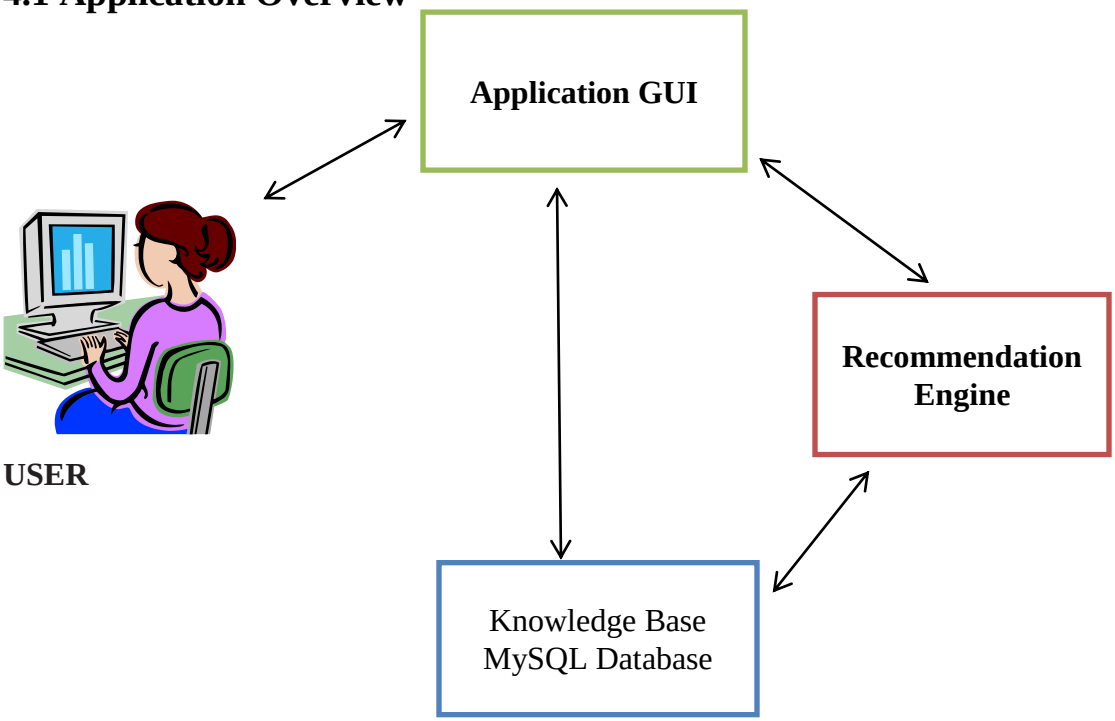

Fig 4.1.1: System Architecture

The personalized recommender system works as follows:

Step 1: The user opens the application and interacts with the GUI home screen to browse and rate the movies he/she has already watched. The home screen GUI also categorizes movie based on genre.

Step 2: After rating the watched movies the user will switch to "My Recommendations" tab and can then query for recommendation of a specialized genre based on its selection.

Step 3: This will now invoke the "Recommendation Engine" to generate the desired recommendation for the user using the proposed algorithm and display it through the UI.

4.2 Requirements and Prerequisites of System

Before building the recommendation framework, the requirements and prerequisites of system ought to be prepared. The aim of the proposed in this thesis solution is to provide personalized recommendations. This requires that systems should fulfill particular functional and quality requirements. The functional requirements are as follow:

- Appropriate data preparation
- Creation of personalized method of recommendation
- Minimize the number of false–positive recommendations

Right data about users ought to be gathered and analyzed. However data acquisition is not done only once. The data about users changes over time and this causes that it should be monitored all the time. Moreover, sufficient data preparation, that includes classification of data in several components, which create user profile should be done in order to create relevant recommendation.

The crucial element is the description that shows how to cope with empty data. Missing data is a big problem because it limits the opportunity to create relevant recommendations. Not only has the correct data decided about the quality of the recommendation, but also the method that is used to investigate the similarity between members of the community. Next requirement is to create function that enables to measure the similarity between the users. Based on this function the person who suits to current user is recommended. This similarity function should take into consideration demographic and interest data, as well as measurements of activity and strength of relationships maintained by members of social networks. This function should be periodically recalculated and enables to pinpoint *k*–nearest neighbor of the person to whom the recommendation is created.

The proposed framework should minimize the number of false–positive recommendations. It means, that we cannot recommend people who do not match current user's profile. It is safer not to recommend the person who suits current user (false–

negative recommendation) than provides many false–positive suggestions, which can be the source of irritation and lack of satisfaction of person who receives the recommendation.

The quality requirements that should be fulfilled at the highest level by the framework are:

• **Performance** – the calculations, especially these made online should not last long, because users will be irritated if they have to wait for the recommendations. Thus, the high performance is required.

• **Maintainability** – the improvements in user profile (e.g. the extension of user profile by adding new components and attributes) and in method of recommendation should be easy to introduce. It will help to ensure high level of the next quality requirement for this system, it means availability. Moreover, the level of adaptability of the system should enable to apply the recommendation framework in different social networks (e.g. *MySpace*, *Friendster*, or *Orkut*).

• **Availability** – the system should be available for most of the time that users spend in the network. It will cause that participant will be accustomed to have the recommendations nearby. This helps to create the bond between the user and system and leads to increase the loyalty of the user.

4.3Database Design

MOVIE table:

The movie table consists of all the details related to a particular movie.

```
mysql> desc movie;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| movieid     | int(11)      | YES  |     | NULL    |       |
| moviename   | varchar(100) | YES  |     | NULL    |       |
| releasedate | varchar(100) | YES  |     | NULL    |       |
| imdblink    | varchar(200) | YES  |     | NULL    |       |
| unknown     | int(11)      | YES  |     | NULL    |       |
| action      | int(11)      | YES  |     | NULL    |       |
| adventure   | int(11)      | YES  |     | NULL    |       |
| animation   | int(11)      | YES  |     | NULL    |       |
| children    | int(11)      | YES  |     | NULL    |       |
| comedy      | int(11)      | YES  |     | NULL    |       |
| crime       | int(11)      | YES  |     | NULL    |       |
| documentary | int(11)      | YES  |     | NULL    |       |
| drama       | int(11)      | YES  |     | NULL    |       |
| fantasy     | int(11)      | YES  |     | NULL    |       |
| filmnoir    | int(11)      | YES  |     | NULL    |       |
| horror      | int(11)      | YES  |     | NULL    |       |
| musical     | int(11)      | YES  |     | NULL    |       |
| mystery     | int(11)      | YES  |     | NULL    |       |
| romance     | int(11)      | YES  |     | NULL    |       |
| scifi       | int(11)      | YES  |     | NULL    |       |
| thriller    | int(11)      | YES  |     | NULL    |       |
| war         | int(11)      | YES  |     | NULL    |       |
| western     | int(11)      | YES  |     | NULL    |       |
| rating      | int(11)      | YES  |     | NULL    |       |
| prediction  | int(11)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
25 rows in set (0.14 sec)
```

Fig 4.3.1: MOVIE table

PROFILE table:

This table will contain the correlation values of each user to each genre.

```
mysql> desc profile;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| USERID      | int(11) | YES  |     | NULL    |       |
| ACTION      | double  | YES  |     | NULL    |       |
| ADVENTURE   | double  | YES  |     | NULL    |       |
| ANIMATION   | double  | YES  |     | NULL    |       |
| CHILDREN    | double  | YES  |     | NULL    |       |
| COMEDY      | double  | YES  |     | NULL    |       |
| CRIME       | double  | YES  |     | NULL    |       |
| DOCUMENTARY | double  | YES  |     | NULL    |       |
| DRAMA       | double  | YES  |     | NULL    |       |
| FANTASY     | double  | YES  |     | NULL    |       |
| FILMNOIR    | double  | YES  |     | NULL    |       |
| HORROR      | double  | YES  |     | NULL    |       |
| MUSICAL     | double  | YES  |     | NULL    |       |
| ROMANCE     | double  | YES  |     | NULL    |       |
| SCIFI       | double  | YES  |     | NULL    |       |
| THRILLER    | double  | YES  |     | NULL    |       |
| WAR         | double  | YES  |     | NULL    |       |
| WESTERN     | double  | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
18 rows in set (0.11 sec)
```

PRS table:

This table will contain the ratings given by 943 users on 1682 movies on a scale of 1 to 5 with a time-stamp of the rating provided.

```
mysql> desc PRS;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| userid    | int(11) | YES  |     | NULL    |       |
| movieid   | int(11) | YES  |     | NULL    |       |
| rating    | int(11) | YES  |     | NULL    |       |
| rate_date | int(11) | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
4 rows in set (0.13 sec)
```

Fig 4.3.3: PRS table

4.4 Software Architecture

Database: MySQL

Development Environment: Eclipse LUNA IDE 4.4.1

JAVA: JDK 1.7

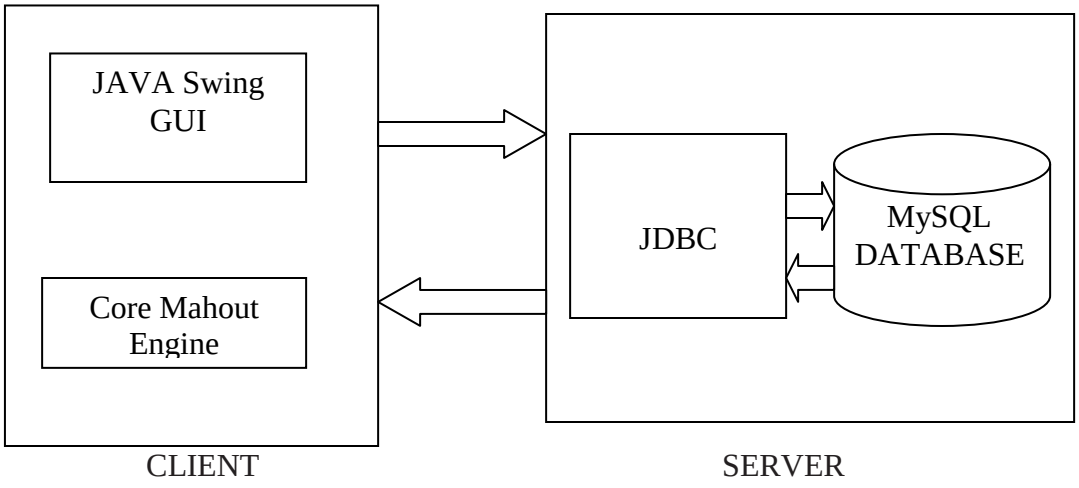

Fig 4.4: Software Architecture

4.5 Modular Description

The application has been developed in the form of following modules:

- User-Interface
- Database Engine
- User and Item Profiling
- Recommendation Engine
- Driver Program

4.5.1 User-Interface

The GUI to interact with the application is developed using Swings in JAVA.

The GUI provides the user with options to:

- View and Rate movies
- View Genre-specific recommendations
- View Help – to tackle nominal problems.

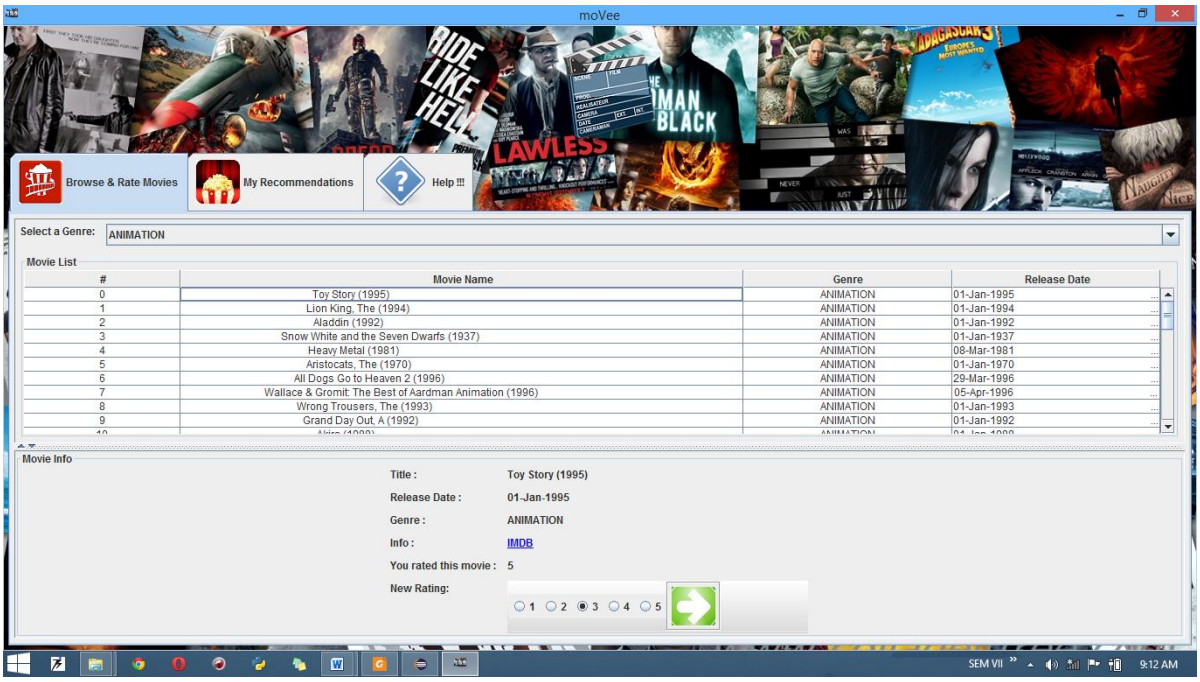

Fig 4.5.1.1: UI to browse and rate movies

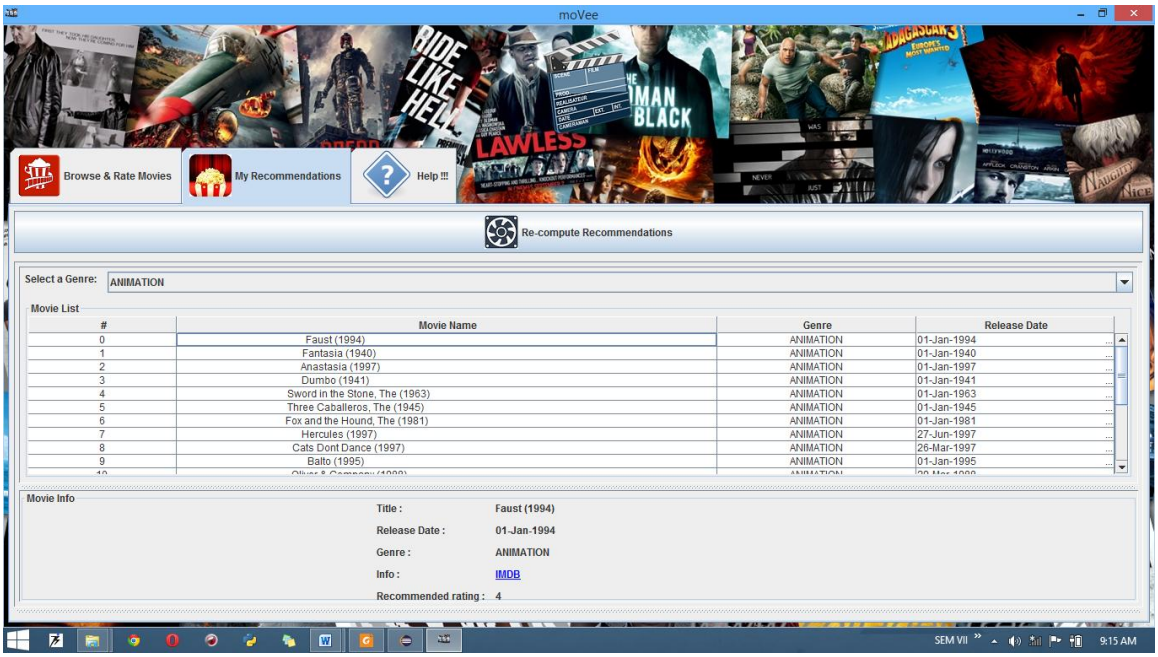

Fig 4.5.1.2: UI to view and compute recommendations

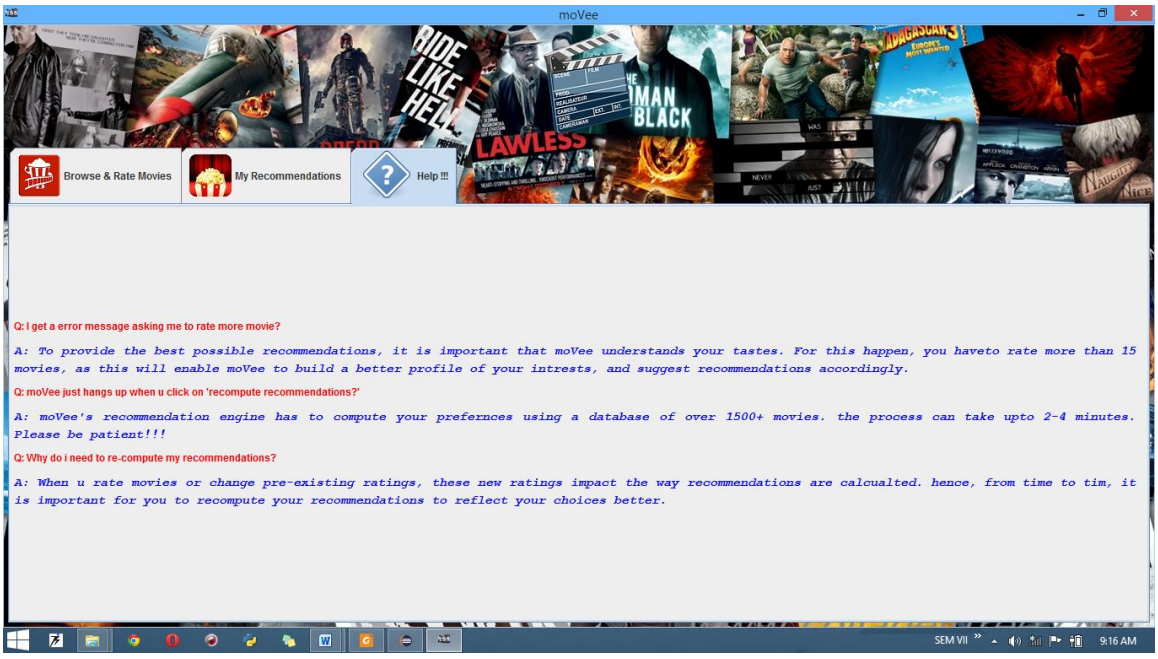

Fig 4.5.1.3: Help Screen

4.5.2 Database Engine

The application uses this module to control interactions with the datasets present in the MySQL database server, using the JDBC driver.

Some of the sub-modules are:

- DBConnectionManager.java :

  This JAVA class is used to establish and close connections with the MySQL server and return the connection object to the driver program.

  The standard function covered by this module are :

  - getConnection();
  - establishConnection();
  - testConnection();
  - closeConnection();

- DBReader.java :

  This JAVA class is used to basically perform all the read operations on the tables present in the MySQL database and return the results in *List* type to the calling module.

  The standard function covered by this module are :

  - setConnection();
  - getRecommendations(String genre);
  - getMovies(String genre);
  - getMovies(int UserId,String genre);
  - getUsers(int movieId);
  - getAvgRating(int userId);
  - getCorrCoff(int userId,String genre);
  - getAvgUserRating(String genre);
  - getMovieRateCount();

- DBWriter.java :

  This JAVA class is mainly responsible to perform all the *Update* and *Write* information in the tables present in the MySQL database server.

  The standard function covered by this module are :

  - setConnection();
  - saveRating(int Rating,movieObj obj);
  - savePrediction(int movieId,int prediction);
  - saveCorrelation(double corr,int userId,String genre);

4.5.3 User and Item Profiling

This module deals with the Item and User profiling to their respective objects for easy and systematic access and use of the data.

- movieObj.java :

  This class is responsible to parse the required movie's data into their respective objects so that they may be further used by the driver program.

- userObj.java :

  This class is responsible to parse the user's info/data into their respective objects for easy access to driver program.

4.5.4 Recommendation Engine

This is the most important module of the application which is actually responsible for generating the recommendations. This recommendation engine uses the Genre specific – User-Based Collaborative Filtering algorithm.

RecomEngineMain.java:

Firstly, the correlations of the active users with users who have liked/rated movies of the specified genre are calculated. The correlation calculations are done using the Pearson's Correlation Co-efficient (Pearson's correlation coefficient is the covariance of the two variables divided by the product of their standard deviations. The form of the definition involves a "product moment", that is, the mean (the first moment about the origin) of the product of the mean-adjusted random variables; hence the modifier *product-moment* in the name.):

$$w_{a,u} = \frac{\sum_{i \in I} (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I} (r_{a,i} - \bar{r}_a)^2 \sum_{i \in I} (r_{u,i} - \bar{r}_u)^2}} \qquad (1)$$

where I is the set of items rated by both users, $r_{u,i}$ is the rating given to item i by user u, and $\bar{r}_u$ is the mean rating given by user u.

Then after selecting the K most similar users the predictions are calculated as:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in K} (r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}} \qquad (2)$$

where $p_{a,i}$ is the prediction for the active user a for item i, $w_{a,u}$ is the similarity between users a and u, and K is the neighborhood or set of most similar users.

The Correlations are calculated as given:

```
publicstaticvoid calculateCorrelations(String genreMain) {
//get all movies for a specific genre X.
ArrayListlistMoviesPerGenre = DBReader.getMovies(genreMain);

//there are 943 users in the movielensdataset. lets start with user A.
for (inti = 1; i< 944; i++) {
ArrayListalstUser = newArrayList();
ArrayListalstFetch = newArrayList();
floatuser = 0;
floatfetch = 0;

//get all movies for user A belonging to a genre X.
ArrayListlistMoviesPerUserPerGenre = DBReader.getMovies(i, genreMain);

// find similar ratings between user and database
for (intx = 0; x<listMoviesPerGenre.size(); x++) {
movieObj moviePerGenre = (movieObj) listMoviesPerGenre.get(x);
for (inty = 0; y<listMoviesPerUserPerGenre.size(); y++) {
movieObj moviePerUserPerGenre = (movieObj) listMoviesPerUserPerGenre.get(y);
if (moviePerGenre.getId() == moviePerUserPerGenre.getId()
&&moviePerGenre.getRating() != 99) {
alstUser.add(Integer.valueOf(moviePerGenre.getRating()));
user = user + moviePerGenre.getRating();
alstFetch.add(Integer.valueOf(moviePerUserPerGenre.getRating()));
fetch = fetch + moviePerUserPerGenre.getRating();
}
}
}

// find average
user = user / alstUser.size();
fetch = fetch / alstFetch.size();

// calculate numerator
floatnumerator = 0, denominator1 = 0, denominator2 = 0;
for (inta = 0; a<alstUser.size(); a++) {
Integer x = (Integer) alstUser.get(a);
Integer y = (Integer) alstFetch.get(a);
numerator = numerator + ((x.floatValue() - user) * (y.floatValue() - fetch));
denominator1 = denominator1 + ((x.floatValue() - user) * (x.floatValue() - user));
denominator2 = denominator2 + ((y.floatValue() - fetch) * (y.floatValue() - fetch));
}
```

```
floatcorrelation = (float) (numerator / (Math.sqrt(denominator1) *
Math.sqrt(denominator2)));
System.out.println("correlation is ::: " + correlation);
if (Double.isNaN(correlation)) {
correlation = 99;
}
DBWriter.saveCorrelation(correlation, i, genreMain);
}
}
```

The Predictions are calculated as given:

```
publicstaticvoid calcualtePredictions(String genreMain) {
System.out.println(">>> " + DBReader.getAvgUserRating(genreMain));
if (genreMain != null) {
ArrayListalstMovies = DBReader.getMovies(genreMain);
for (inti = 1; i<alstMovies.size(); i++) {
movieObj obj = (movieObj) alstMovies.get(i);
ArrayListalstUsers = DBReader.getUsers(obj.getId());
floatprediction = 99;
floatnumerator = 0;
floatdenominator = 0;
intavg = DBReader.getAvgUserRating(genreMain);

for (intj = 0; j<alstUsers.size(); j++) {
userObj usr = (userObj) alstUsers.get(j);
floatcorr = DBReader.getCorrCoff(usr.getId(), genreMain);
if (!Double.isNaN(corr)) {
floatusrAvg = DBReader.getAvgRating(usr.getId());
numerator = numerator + ((usr.getRate() - usrAvg) * corr);
if (corr< 0) {
denominator = denominator + (-1 * corr);
} else {
denominator = denominator + (1 * corr);
}
prediction = avg + (numerator / denominator);
} else {
prediction = 99;
}
}
DBWriter.savePrediction(obj.getId(), (int) prediction);
System.out.println(obj.getId() + " prediciton is >>>  " + (int) prediction);
}
}
}
}
```

4.5.5 Driver Program

This module is the main class of the applications responsible for running the application and controlling the various modules specified above.

4.6 Facebook Integration

The application developed till uses a static interface for getting user's interests and ratings. So, to actually use this prototype the user has to spend a considerable amount of time interacting with the application to actually get some considerable recommendations. Further this prototype does not involve the preferences of the user's friends and close ones.

This part will hence involve the integration of a social networking site to consider the preferences of the friends of the user, because of the factor of trust involved with social network's friends, and will hence enhance the recommendations for the user.

Moreover the integration of the social network in the application will further enable the algorithm to analyze the overall/general pattern of preference of the social network users towards a particular item, in this case movies.

4.6.1 Part-II System Design

We stated above that the user is spending a considerable amount of time to interact with the application to browse and rate some movies so as the system is further able to compute recommendations. We can actually enhance this process and also make it efficient by just mining the movies rated by the users on the social network. Social networks like Facebook provides the RestFB API to mine user's personal info like likes, photos and other demographic info.

In social networks like Twitter, all the articles, posts etc. can be referenced through tags assigned through them. These tags can be mined from Twitter, using Scalding, an open-source API provided by twitter, to find relevant posts related to the movies rated by the user. These posts will enable us to find the users who also have posted on those movies and then calculating the movie-movie similarity/correlation score using the co-sine similarity metrics between the movie vectors.

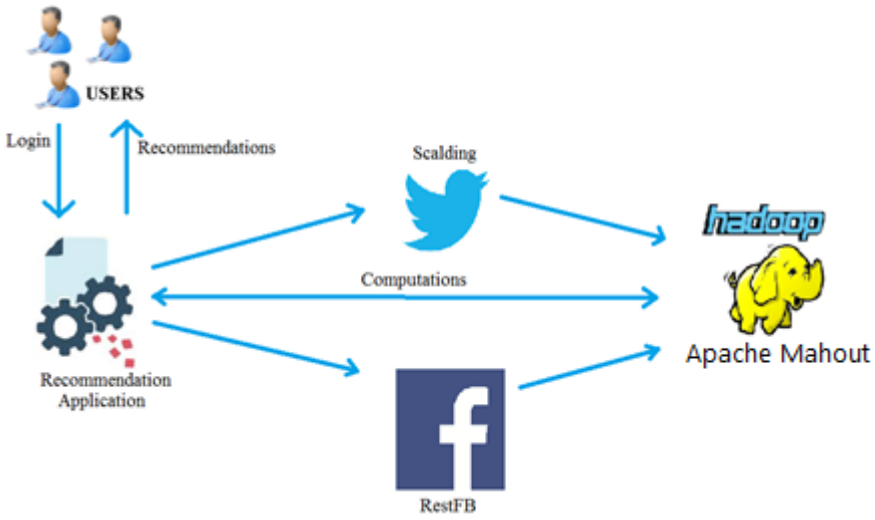

Fig 6.1: Part-2 System Design

We will now feed the above mined data to the recommendation engine and perform the prediction operation using Similarity functions to generate the desired recommendations.

4.6.2 Requirements and Prerequisites

The quality requirements that should be fulfilled at the highest level by the framework are:

• **Performance** – the calculations, especially these made online should not last long, because users will be irritated if they have to wait for the recommendations. Thus, the high performance is required.

• **Maintainability** – the improvements in user profile (e.g. the extension of user profile by adding new components and attributes) and in method of recommendation should be easy to introduce. It will help to ensure high level of the next quality requirement for this system, it means availability. Moreover, the level of adaptability of the system should enable to apply the recommendation framework in different social networks (e.g. *MySpace*, *Friendster*, or *Orkut*).

• **Availability** – the system should be available for most of the time that users spend in the network. It will cause that participant will be accustomed to have the recommendations nearby. This helps to create the bond between the user and system and leads to increase the loyalty of the user.

4.6.3 RestFB – Introduction

RestFB is a simple and flexible Facebook Graph API client and old REST API client written in Java.

It is open source software released under the terms of MIT license.

The motivations to use RestFB as the API are:

Design goals:

- Minimal public API
- Maximal extensibility
- Robustness in the face of frequent Facebook API changes
- Simple metadata-driven configuration
- Zero dependencies

Non-goals:

- Support for non-Graph/non-REST API parts of the Facebook Platform
- Providing a mechanism for obtaining session keys or OAuth access tokens
- Using XML as a data transfer format in addition to JSON
- Formally-typed versions of all Facebook API methods, error codes, etc.

4.6.3 RestFB – FQL (Facebook Query Language)

Facebook Query Language, or FQL, enables you to use a SQL-style interface to query the data exposed by the Graph API. It provides advanced features not available in the Graph API.

Queries are of the form SELECT [fields] FROM [table] WHERE [conditions]. Unlike SQL, the FQL FROM clause can contain only a single table. You can use the IN keyword in SELECT or WHEREclauses to do subqueries, but the subqueries cannot reference variables in the outer query's scope. Your query must also be *indexable*, meaning that it queries properties that are marked as indexable in the documentation below.

FQL can handle simple math, basic boolean operators, AND or NOT logical operators, and ORDER BY and LIMIT clauses. ORDER BY can contain only a single table.

For any query that takes a uid, you can pass me() to return the logged-in user. For example:

```
SELECT name FROM user WHERE uid = me()
```

Other functions that are available are now(), strlen(), substr() and strpos().

Here's an example of a subquery that fetches all user information for the active user and friends:

```
SELECT uid, name, pic_square FROM user WHERE uid = me()

OR uid IN (SELECT uid2 FROM friend WHERE uid1 = me())
```

Multi-query evaluates a series of FQL (Facebook Query Language) queries in one call and returns the data at one time.

This method takes a JSON-encoded dictionary called "queries" where the individual queries use the exact same syntax as a simple query. However, this method allows for more complex queries to be made. You can fetch data from one query and use it in another query within the same call. The WHERE clause is optional in the latter query, since it references data that's already been fetched. To reference the results of one query in another query within the same call, specify its name in the FROM clause, preceded by #.

For example, say you want to get some data about a user attending an event. Normally, you'd have to perform two queries in a row, waiting for the results of the first query before running the second query, since the second query depends on data from the first one. But with fql.multiquery, you can run them at the same time, and get all the results you need, giving you better performance than running a series of fql.query calls. First, you need to get the user ID and RSVP status of each attendee, so you'd formulate the first query – query1 – like this:

```
"query1":"SELECT uid, rsvp_status FROM event_member WHERE eid=12345678"
```

Then to get each attendee's profile data (name, URL, and picture in this instance), you'd make a second query – query2 – which references the results from query1.

- **Initialization**

  DefaultFacebookClient is the FacebookClient implementation that ships with RestFB. You can customize it by passing in custom JsonMapper and WebRequestor implementations, or simply write your own FacebookClient instead for maximum control.

```
FacebookClient facebookClient = new DefaultFacebookClient(MY_ACCESS_TOKEN);
```

  It's also possible to create a client that can only access publicly-visible data - no access token required. Note that many of the examples below will not work unless you supply an access token!

```
FacebookClient publicOnlyFacebookClient = new DefaultFacebookClient();
```

- **Fetching**

  For all API calls, you need to tell RestFB how to turn the JSONreturned by Facebook into Java objects. In this case, the datawe get back should be mapped to the User and Page types, respectively. You can write your own types too!

```
User user = facebookClient.fetchObject("me", User.class);

Page page = facebookClient.fetchObject("cocacola", Page.class);

out.println("User name: " + user.getName());

out.println("Page likes: " + page.getLikes());
```

4.6.4 RestFB – Fetching User's liked Movies

```
package restfbtest;
import com.restfb.*;
importcom.restfb.batch.*;
import com.restfb.exception.*;
importcom.restfb.json.*;
import com.restfb.types.*;
importcom.restfb.util.*;
import java.io.*;
/**
 *
 * @authorakash
 */
publicclass FacebookConnector {

/* Variables */
privatestaticfinal String pageAccessToken =
"CAACEdEose0cBAKzNijkEyOipyCAN57feFo4fpQJAc5Ei22oCUcVD24ZBpxrWrJKn
ORqizKi2dD1HhZBZAhzuXvzX9r2UYN46SlfYvTDe2vY3Xac8QcIwX08X7IBkpFUEj
ORF7ZBsr2RHtO9RHwAC1gUfB8Vz1jZBGo6fHKw49mZA2hcOYiEAvHebJKrojbIX
dv0SiZCf0vtmh9ZB83NyGaKo";
//private final String pageID = "THIS_TOO";
privatestatic FacebookClient fbClient;
privatestatic User myuser = null;    //Store references to your user and page
privatestatic Page mypage = null;    //for later use. In this answer's context, these
//references are useless.


publicstaticvoid main(String args[])throws IOException {
intcounter=0;
        try {

fbClient = new DefaultFacebookClient(pageAccessToken);
myuser = fbClient.fetchObject("me", User.class);
//      mypage = fbClient.fetchObject(pageID, Page.class);
counter = 0;
        Connection<Page>fetchConnection = fbClient.fetchConnection( "me/movies",
Page.class );

for ( Page page : fetchConnection.getData() )
        {
           System.out.println( page.getName() );

//System.out.println( page.getLikes() );
           System.out.println( "**********" );
```

```
        }


/*System.out.println("posts:  \n\n");


Connection<Post>fetchpost = fbClient.fetchConnection( "me/feed", Post.class );

        for ( Post u : fetchpost.getData() )
        {
           System.out.println( u.getDescription() );
           System.out.println( " \\"  );
        }

        */


      } catch (FacebookException ex) {     //So that you can see what went wrong
ex.printStackTrace(System.err);  //in case you did anything incorrectly
      }
//makeTestPost(counter);
   }

/*public static void makeTestPost(int counter) {
      fbClient.publish("me/feed", FacebookType.class, Parameter.with("message",
Integer.toString(counter) + ": Killer Avenger!"));
      counter++;
   }*/

}
```

5. CONCLUSION

Recommendation algorithms provide an effectiveform of targeted marketing by creating a personalized shopping experience for each customer. Forlarge retailers like Amazon.com, a good recommendation algorithm is scalable over very largecustomer bases and product catalogs, requires onlysub second processing time to generate online recommendations, is able to react immediately tochanges in a user's data, and makes compellingrecommendations for all users regardless of thenumber of purchases and ratings. Unlike otheralgorithms, item-to-item collaborative filtering isable to meet this challenge.

In the future, we expect the retail industry tomore broadly apply recommendation algorithms fortargeted marketing, both online and offline. Whilee-commerce businesses have the easiest vehicles forpersonalization, the technology's increased conversion rates as compared with traditional broad-scaleapproaches will also make it compelling to offlineretailers for use in postal mailings, coupons, andother forms of customer communication.

6. REFERENCES

- J.B. Schafer, J.A. Konstan, and J. Reidl, "E-Commerce Recommendation Applications," *Data Mining and Knowledge Discovery*, Kluwer Academic, 2001, pp. 115-153.

- P. Resnick et al., "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," *Proc. ACM 1994 Conf. Computer Supported Cooperative Work*, ACM Press, 1994, pp. 175-186.

- J. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, Morgan Kaufmann, 1998, pp. 43-52.

- B.M. Sarwarm et al., "Analysis of Recommendation Algorithms for E-commerce," *ACM Conf. Electronic Commerce*, ACM Press, 2000, pp.158-167.

- K. Goldberg et al., "Eigentaste: A Constant Time Collaborative Filtering Algorithm," *Information Retrieval J.*, vol. 4, no. 2, July 2001, pp. 133-151.

- P.S. Bradley, U.M. Fayyad, and C. Reina, "Scaling Clustering Algorithms to Large Databases," *Knowledge Discovery and Data Mining*, Kluwer Academic, 1998, pp. 9-15.

- L. Ungar and D. Foster, "Clustering Methods for Collaborative Filtering," *Proc. Workshop on Recommendation Systems*, AAAI Press, 1998.

- M. Balabanovic and Y. Shoham, "Content-Based Collaborative Recommendation," *Comm. ACM*, Mar. 1997, pp. 66-72.

- G.D. Linden, J.A. Jacobi, and E.A. Benson, *Collaborative Recommendations Using Item-to-Item Similarity Mappings*, US Patent 6,266,649 (to Amazon.com), Patent and Trademark Office, Washington, D.C., 2001.

- B.M. Sarwar et al., "Item-Based Collaborative Filtering Recommendation Algorithms," *10th Int'l World Wide Web Conference*, ACM Press, 2001, pp. 285-295.