A RECOMMENDATION FRAMEWORK USING ONTOLOGICAL USER PROFILES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÇAĞLA YAMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2011

Approval of the thesis:

A RECOMMENDATION FRAMEWORK USING ONTOLOGICAL USER PROFILES

submitted by ÇAĞLA YAMAN in partial fulfillment of the requirements for the degree of Master of Computer Science in Computer Engineering Department, Middle East Technical University by

Date : 7/9/2011

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

ÇAĞLA YAMAN

ABSTRACT

A RECOMMENDATION FRAMEWORK USING ONTOLOGICAL USER PROFILES

Yaman, Çağla
M.Sc., Department of Computer Engineering
Supervisor: Prof. Dr. Nihan Kesim Çiçekli
September 2011, 67 pages

In this thesis, a content recommendation system has been developed. The system makes recommendations based on the preferences of the users on some aspects of the content and also preferences of similar users. The preferences of a user are extracted from the choices of that user made in the past. Similarities between users are defined by the similarities of their preferences. Such a system requires both qualified content and user information. The proposed system uses semantic user and content profiles to more effectively define the relationships between the two and make better inferences. An ontology is defined using the existing domain ontologies and the semi-structured data on the web. The system is implemented mainly for the movie domain in which well-defined ontologies and user information are easier to access.

Keywords: Recommendation, User Profiling, Ontology

ÖZ

ONTOLOJİK KULLANICI PROFİLLERİNE DAYANAN İÇERİK ÖNERİMİ ÇATISI

Yaman, Çağla
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Nihan Kesim Çiçekli
Eylül 2011, 67 sayfa

Bu tez çalışmasında, bir içerik tavsiye sistemi oluşturulmuştur. Sistem, tavsiyede bulunurken, kullanıcının tavsiye edilecek nesne hakkındaki tercihlerini kullanır. Bu tercihler kullanıcının daha önce oylamış olduğu içeriklerin özellikleri dikkate alınarak belirlenir. Tavsiye aşamasında aynı zaman kullanıcıya benzer diğer kullanıcıların da tercihlerinden yararlanılır. Bu tarz bir sistem, yeterli sayıda ve anlamlı içerik ve kullanıcı bilgisine ihtiyaç duyar. Bu çalışmada önerilen tavsiye sistemi, anlamsal kullanıcı ve nesne profilleri kullanarak, bunların aralarındaki ilişkilerin daha etkili olarak belirlenmesini ve daha sağlıklı çıkarımlar yapılabilmesini sağlar. İçerikleri ve kullanıcıları temsil edebilmek üzere var olan alan ontolojilerinden ve veb üzerindeki yarı yapısal verilerden yararlanılarak sistemin kullanacağı bir ontoloji tanımlanmıştır. Hem yapısal ve ontolojik olarak tanımlanmış alan bilgisinin, hem de erişilebilir kullanıcı bilgilerinin varlığı nedeniyle, örnek olarak, bir film tavsiye sistemi gereçkleştirilmiştir.

Anahtar kelimeler: Tavsiye Sistemleri, Kullanıcı Profili Oluşturma, Ontoloji

ACKNOWLEDGEMENT

I want to thank my supervisor Prof. Dr. Nihan Kesim Çiçekli for her guidance. She has been generous in sharing her knowledge and experience. Her patience and efforts made this work better.

I wish to express my greatest gratitude to my mother Reyhan Yaman, my father Ünal Yaman and my whole family. They have always supported me and guided me during my studies and throughout my life.

Special thanks to my friend Engin Egeren for his patience, support and encouragement throughout this work. I also want to thank Özlem Özkan who has been great help when I needed.

Burçin Gürses, Gizem Erbaş, Özüm Akdere, Göze İşcan and Pınar Yıldırım are gratefully acknowledged. You are always there for me.

This work is partially supported by The Scientific and Technical Council of Turkey Grant TUBITAK EEEAG-107E234 and my master studies are supported by the student grant of TUBITAK.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Recommender systems help users discover new objects or contents among a large amount of alternatives by performing the search and decision processes on behalf of users. With the increase of the amount and the variety of information on the Web today, it has become difficult for the users to reach and manipulate all those information while making a decision. Also, the decision process can be too complicated that it would take much effort and time for a user to perform; and still, some aspects of it might be overlooked. Users might not even be aware of all the aspects of their decisions and the reasons behind their actions.

The recommendation task is easier to carry on with computer systems with their ability to manipulate larger amounts of data more quickly. But of course, those systems are adaptations of real life processes so they simulate how people make recommendations and evaluate contents. A person may investigate contents himself and decide what aspects of these he likes or dislikes. Then he makes a decision whether to prefer that content or not and why. He can either ask his friends if they liked it. He can prefer the most popular contents. As we have these options in real life, there are recommender systems that implements one of these methods or a combination of them.

The main recommendation methods are content-based filtering and collaborative filtering. Content-based filtering learns the taste of the user in terms of properties of contents and recommends contents similar to user's taste. Collaborative filtering uses the ratings given by the user to determine similar users that rated the same contents in the same way, it then recommends contents these users have given high ratings.

Both methods have some problems, so hybrid systems are developed to solve those problems or at least minimise their drawbacks. Moreover, researchers have focused on other contributions to the recommendation research area such as semantic technologies and ontological user profiling for quite some time.

Using user profiles enables recommender systems better understand the preferences of the user; make more specific and accurate recommendations and explain the reasons of these recommendations. Different studies have handled the profile generating problem with different strategies or even different points of views.

They differ in the elements and details they choose to include in profiles such as whether to include the individual items users prefer or combination of different preferences or the whole concept the individual is related to. For example, a person may express in his profile that he went to a movie theatre to watch “Titanic”. From this expression, system may infer different predictions such as “Person likes romantic movies.”, “Person likes romantic movies starring Leonardo DiCaprio”, “Person likes all movies starring Leonardo DiCaprio”, “Person likes all art forms performed by Leonardo DiCaprio (in movies, in plays, as actor, as director, as writer etc.)”, “Person may also like romantic books and music”. Each of these would lead a recommender to different choices some of which would be very unsuccessful. But still, none of the predictions are totally wrong. There is no exact answer to this design problem since we, in our personal lives, don’t know the exact motives behind our decisions.

Of course, more detailed explanations from user will help the system. But when the system lacks that information, it has to extract it from the user’s past behaviours known to the system. Still, in this case, more user history will help system understand the decision pattern of the user. But, how much information do we need? Actually, the more information there is, the better the recommendation gets. However, it will bring the system huge work load and making recommendations will be impossible or ineffective because of performance issues. System will be very slow to answer its users' needs or very inconsistent; even it is able to handle the operations it will be using much effort than needed.

Also, despite the large amount of information, this information may still be useless. For example, if the same user says he went to "Transformers" the next day, it would not help us improve our predictions about "romantic movies" or "Leonardo DiCaprio". But if he watches "The Departed", another movie by Leonardo DiCaprio which is a thriller, we will validate the predictions "Person likes all movies starring Leonardo DiCaprio", "Person likes all art forms performed by Leonardo DiCaprio" until new information comes. Quality of the information gathered is very important in recommendation processes. However there isn't a way to control what comes with user histories, or controlling it is not realistic. So we need to rely on the availability of sufficient amount of information that is much enough to make meaningful inferences and small enough for system to work with minimum effort.

The other aspect that different recommender systems diverge is the expression of user's contentment from a concept. We know that user watched "Titanic" but we also need to know if he liked it or not or what level he liked it. In real life, our expressions of our appreciation for something are ambiguous. In a recommender system, the levels of interest for concepts are determined through rating values. These values may be too sensitive between larger scales with smaller intervals or simply binary valued "like" and "dislike". Though, it first seems using sensitive measurements is more advantageous, usage of both may be necessary or preferable under different circumstances even in the same recommender system. Some preferences may be mandatory such as preferring only the movies in English or not watching horror movies. In these cases, the movies that do not meet these criteria should not be recommended to user.

Sometimes systems may be in need of converging scaled ratings to binary ratings. Some systems do this while detecting the weight and importance factor of attributes of items; but all systems need to do it when deciding whether to recommend an item at the end. The most used way of doing this is to determine a threshold value that indicates a positive opinion. But what should that value be? Is "average" enough to recommend a movie to a friend or should it be "good" or "fantastic"? This decision also depends on the structure of the system and distribution of inputs. If users of the system poorly rated the items the threshold should be reduced; in the other case it should be higher than the average.

The question naturally arises is, can the same threshold be applied to every user? In fact, it is generally the case; most systems use one specific threshold for every user. What we think is that it is not realistic since the idea of a good movie is different for everyone. One person may use "fantastic" for a movie he liked while other uses "good" for a movie he liked better. The rating distributions of these users will differ, so different threshold values personalized for these people should be used.

Now that we know that our user watched the movies "Titanic" and "The Departed", what if he liked "Titanic" and hated "The Departed"? The system that considers only the good ratings is not able to detect this conflict. The other way, we are not able to decide if he likes Leonardo DiCaprio or not. We can still say he likes romantic movies and predict that he hates thrillers, but how coherent these predictions are now? We see that it was too soon to decide with only one movie rated. How to deal with this problem or whether to deal with it is a design issue. Again, there is no standard solution.

If our user liked both movies, we can implicitly infer that "Person doesn't care about the genre of the movie, but the actor". This is another aspect of user preferences more affected by the amount of information available because different examples of one attribute are needed to decide.

Even with all this information available there is still a question of which of these predictions are meaningful. Does our prediction “Person may also like romantic books and music” always make sense? Some domains may be closely related to each other that this kind of inferences may add strengths to recommendation process whereas for some domain it may make no sense. Even in the same domain same rules may not be applicable. For movie and music domains, a user may like both the movies filmed in 1960s and music from 1960s which validates the accuracy of our prediction. On the other hand, he may like movies from 1970s but not like the music from that time.

Recommender systems research is about dealing with these conflicting issues and does not aim to find a general solution that is applicable to all recommendation problems, because, people do not have one pattern for recommendation. Our decisions and recommendations to others differ according to what we know about the concepts, who we are with and even how we feel when we make decisions. This is why most of the recommendation systems are specific to their domain and the information sources they use. They try to find the most efficient solution specific to their needs. The issues discussed above are relatively less complicated to solve in these specific works.

The research to realize a general purpose recommendation system suffers from these conflicting design issues. In this thesis, we suggest and implement a general purpose recommender system and show our approach to attack these problems. In this work, we

- investigate different aspects of design decisions and how they affect the recommendation process,
- benefit from the ontological user profiles and ontological item descriptions in recommendation,

- introduce a more effective user profile model using tastes of user and decision criteria together,
- introduce a user profile model that can be used to describe user taste of different type of items,
- define an easy-to-adapt recommendation framework to use as a base for different recommendation problems of different domains.

A general purpose recommendation framework independent of the domain is implemented. That system consists of modules that can be modified to answer the needs of different recommendation problems. That enables the same system be used for recommending different types of items and use different kinds of data sources.

Proposed system is a hybrid recommendation system using content-based and collaborative filtering. For collaborative filtering, system clusters similar users. Different parameters and design decisions affects the performance of the clustering phase. We investigated the effects of these issues such as number of users, number of clusters, the number of common preferences of users in a cluster, number of elements in a user profile. These parameters define how specific a cluster should be to make more successful recommendations.

For content-based filtering and also as input model to collaborative filtering, user profiles are used in the system. These user profiles consist of the history of the user and the preferences. The criteria of the users for choosing an item may be different. Different users may pay attention to different attributes (referred as properties) of items. These criteria are included in the user profiles as well as the tastes of the user.

To design such user profiles, a user ontology is defined and also ontological descriptions of the items are used as input to the system. We investigated the effect of the user profile model proposed and the use of ontologies.

User profiles are also designed to contain different preference groups independent of each other. We name these groups *domain profiles*. A domain profile represents the preferences of a user over one type of item such as movie domain profile, music domain profile, etc. Whether to enable those profiles affect each other is a strategic issue to be decided on. We decided to keep those domain profiles separate because a preference in one domain may not always be meaningful or preferable in other domains. Each domain profile should be extracted separately and can use different types of data sources as input.

Finally, a movie recommendation system is implemented as a realization of the framework and its performance is evaluated.

In the following chapters, the details of the work done are presented. Chapter 2 investigates the aspects of the problem and the research done so far that has influenced the solutions found in this thesis work. The proposed general purpose recommendation framework is introduced in Chapter 3. In Chapter 4, the implementation of an example movie recommendation system is presented. The performance of this system is tested and evaluated in Chapter 5. And finally, Chapter 6 gives a brief summary of the thesis and presents the conclusions.

CHAPTER 2

BACKGROUD AND RELATED WORK

Recommender systems are used in many different areas with many different degrees of complexity varying from education to entertainment. Despite their differences, what they do is mainly to detect similar behaviours under similar circumstances and estimate the properness of the objects of interest. Whatever the complexity of the problem is, finding the behavioural patterns is essential. These patterns can be argued to derive from the characteristics of the objects and the past behaviours of a single user; or the recommendation process can be thought to be a social act where same patterns are valid for more than one user. This duality has let researches to two different recommendation strategies: collaborative filtering (CF) and content-based filtering (CBF) [1].

Collaborative filtering handles the recommendation problem as a social act. People with similar tastes are thought to be helpful for an individual’s decision. People who like same items or evaluate same items in the same way are considered to be similar. Users similar to a user are referred as neighbours of that user [2]. The preferences of neighbours designate the user’s evaluations of an item. The items that a user’s neighbours like are appropriate to recommend to that user.

Some CF systems cluster users instead of finding neighbours of each user separately. Clustering is gathering similar users together in groups and forming clusters that reflect the common tastes and interests of these users. In this case the common agreement of the cluster affects a user’s evaluation of an item.

In content-based filtering, the properties of the items to be evaluated are important. The preferences of a user over those properties determine the likelihood of recommending an item to a user. The items that resemble the ones high rated by the user are recommended.

Both methods have their strengths and weaknesses. CF requires a lot of users voting same items for recommendations to be successful. However, this might not always be the case. Users of the system might have evaluated different items which makes it difficult to find similarities between users. This is called the *sparsity problem* [3,4]. *Cold-start problem* [3] of CF systems refers to either the problem of a new item which is not rated by any of the users yet or a new user problem who has not rated many items so cannot be similar to any other users [4]. Also, some users may have rare tastes and may not have rated same items in the same way as the rest of the users. CF systems also make recommendations highly limited by the items chosen before.

CBF suggests solutions to some of the problems that occur in CF. It is capable of evaluating an item using its properties even if it has not been rated by any of the users before. It still requires that the user has rated enough number of items to understand his preferences and make good recommendations, but still, it can evaluate an item using the limited number of information about that user. CBF does not suffer from the sparsity problem [3].

On the other hand, the risk of CBF is that its success depends on the definition of the items [4]. The number and the significance of the properties of items highly affect the evaluation process. There is also a strong probability that the recommendations made will be too similar because these systems will only recommend items that are suitable with the preferences of the user.

A more effective approach to recommendation problem is hybrid systems [5, 6, 7, 8, 9, 10, 11] which aim to eliminate specific drawbacks of both methods by combining them. Hybrid methods use CF and CBF together with different strategies as summarized in [5] and [4]. Some hybrid systems switch between CF and CBF under different circumstances; while some implement a staged process and execute both filtering methods sequentially on inputs or use output of one method as input to the other. Some use two methods interleaved, by adding CF features into CBF or vice versa.

Besides the choice of the filtering method, researchers apply some techniques and strategies to improve the performances of their systems. Generating user profiles or using semantic technologies in the recommendation process help systems make better recommendations. Many researches generate user profiles using the content-based information of the items user has rated. Content-based user profiles give better understanding of users' interests and choices, and they can be used both in CF and CBF systems. The use of semantic technologies improves the performance of a recommendation system because they give more meaningful and compatible information about domains, items and user profiles when used. Some researches use only ontologies to benefit from semantics of the domain studied[12] while some use more advanced semantic technologies like reasoning [3, 13].

In [14], they try to improve the performance of CF methods by replacing the most widely-used user-based CF with item-based CF and using semantic knowledge. When evaluating an item for a user, the system finds the similar items, then uses the ratings given by the user to these similar items to predict the item's rating. The similarity between items is a combination of collaborative and semantic similarity. Items that are rated similarly by some users denote collaborative similarity. Semantic similarity can be thought of content-based similarity as it is the similarity between the ontological descriptions of items, which indicates the properties of them. Using semantic similarity enables the system to detect and recommend also the items that are not rated by many users. By using semantic knowledge of items, this work aims to eliminate the sparsity and the new item problem in CF systems.

[15] is another work using item based collaborative filtering. The difference from the previous work is that it generates clusters of items considering their attributes. Then, each user is included in a cluster according to their previously rated items.

In [16] the proposed system uses semantic technologies to improve its CBF method. It defines users' preferences of the item attributes ontologically also including the complex preferences which reflect conditional preferences representing more than one attribute affecting each other. The candidate items are evaluated using these profiles.

AVATAR [7] is a hybrid multimedia content recommender that switches between two different filtering methods. It uses OWL[1] ontologies for representing items and benefits from semantic inference. When evaluating an item, it first executes CBF strategy. It uses the ratings of items already rated by the user which show some semantic similarities with the item to be evaluated. It considers hierarchical semantic similarities and inferential semantic similarities. Hierarchical relationships are ancestor-descendant relationships between concepts that are explicitly defined in OWL ontologies. Inferential similarities are detected by applying inferential rules and refer to shared semantic characters such as sharing common features or having related features. If the system cannot find enough similarity and make a decision by CBF, it switches to its CF strategy where it finds the neighbours of the user and applies the former strategy to this set of users. The neighbours' ratings of items that show similarities to the unrated item are used to estimate its rating.

In [5], a hybrid recommendation system using user profiling and semantic technologies such as ontologies and semantic spreading is implemented. User profiles consist of areas of interest which correspond to preferred concepts of domain together with their related concepts extracted by semantic spreading. The system clusters users according to their common areas of interests producing

[1] http://www.w3.org/2004/OWL/

communities of interest and recommend items using opinions of these communities.

Hybrid recommendation models suggested in [9], [10] and [11] generate weighted user profiles representing the preferences over the values of attributes of items. Recommendations are done using both opinions of similar users and these individual user profiles.

The hybrid system suggested in this thesis adds CBF features into CF stage and uses the output of the CF as input to the CBF stage as the recommendation strategy. Adding CBF features in CF stage is done by generating content-based user profiles and clustering users using these profiles while defining similarities between them. We define items and user profiles ontologically which gives us the opportunity to better understand the preferences of the users. In the following, we investigate the recommendation systems that show similarities with ours in more detail and discuss how they have affected our designs decisions.

HYBRID RECOMMENDATION STRATEGY

As we mentioned, there are different strategies to combine collaborative and content-based filtering. While investigating different aspects of recommender systems we are influenced from, we will first talk about these strategies and the CBF methods. We leave the details of profile generating and CF mechanisms to later chapters to discuss in more detail.

A hybrid recommendation model following the "collaborative via content" paradigm is implemented in [5] and [9]. This paradigm is used to indicate the model benefits from content-based user profiles in collaborative filtering. System first extracts users profiles from the content-based descriptions of the concepts preferred by users. Then it passes these profiles to a clustering mechanism. The clusters generated are used in CF.

In [10], content-based user profiles are generated considering the most characteristic preferences of users over the attributes of items. When making recommendations, users with most similar profiles are detected and used in CF.

All of these three systems generate their user profiles using CBF. Then they use these profiles in CF. With these characteristics, these works are meta-level hybrid recommenders. Meta-level hybrid recommenders use the model generated by first filtering method as input to the second one. The user profiles generated are the model resulting from CBF and are used as input to the CF mechanism.

Our work follows this approach too, and therefore is a meta-level hybrid recommender. But it also applies CBF over CF results, which is a characteristic of hybrid recommenders based on feature augmentation. This kind of recommenders implements a staged process where the outputs of one filtering methods are improved by applying the other method without changing the filtering methods. In our work we simply find the items that can be recommended to a user by the other users in the same cluster and evaluate the items according to the user's profile to improve the results.

Work in [11] also uses CBF in generating user profiles. But its difference is that it doesn't use these profiles in CF. Instead, it clusters items by their common attributes hierarchically, from general groups that contain different features and more items to more specific groups that contain more mutual features and fewer items. From these clusters, it finds candidate items to recommend to user simply choosing the cluster containing already rated items of the user. Then it applies content based filtering to candidates to determine their similarity degree with the user profile. The similarity degree is calculated by summing the weights of the matching attributes of items and user profile. Using this similarity degree and community's average rating to candidate items, it makes recommendations to user. This system uses CBF as its base recommendation method but benefits from the collaborative information such as the average ratings of items.

Our system resembles this system in a way that they both filter candidates with user profiles. Of course, the candidate lists are generated with different strategies, since we use CF in determining candidates. Our system also implicitly uses similarity degrees of candidates. Different from this system, in ours, the similarity degree of each attribute is considered separately and may also infer negative opinion depending on the values of matching attributes. Instead of using community's ratings to items, we use individual user's rating to values of attributes.

USER PROFILING

Recommender systems using user profiles aim to better understand the reasons of the preferences of the users and make better recommendations. However what they present in the user profiles and how they extract those profiles differ. Different systems give importance to different aspects of users' choices and domain properties. Their representations of user profiles are also different. Some represent profiles with vectors, some in databases, and some as ontologies.

The work in [10] generates user profiles using the features appearing in the rated items of users (user history). It counts the number of times a feature is seen in each user's history. After feature frequencies are extracted for all users, it applies feature-frequency and inverse-user-frequency to these and forms the final user profiles consisting of feature weights. Feature-frequency shows how effective a feature is for a user's preferences while inverse-user-frequency helps eliminate the common preferences of all users which does not indicate a personal choice and is not useful in recommendation. Items are evaluated by calculating the weights of their features using the frequencies of these features among the user's neighbours.

Since this work does not consider the importance of different attributes of items and threats the values of every attribute as plain features in the same category, it falls behind in understanding the preferences of the users in detail.

In [11], a different strategy is followed by generating user profiles consisting of only the preferences of the user about the attributes of items (such as actor, director, genre in movie domain) and these attributes' weights. The weights are the importance factors of each attribute according to the highest frequencies of the values these attributes have among the items that the user rated. This work considers only the importance factors of attributes. When evaluating candidates, it compares the values of attributes of the item and the user profile. If it finds matching values, it concludes the item is similar to user profile by means of the attributes these values belong to. Then, it uses the weights of the matched attributes to calculate the rating of the item.

In this thesis, we also use weighted attributes. We define these weights using the frequencies of the individual values of properties in the items user prefers (highly rated). Different from the work mentioned, we also include the ratings of the individual values in our user profiles. Using weighted properties enables the system consider the importance factor of an attribute in users' choices. The individual ratings enable user profiles to reflect the taste of the user (what he likes and dislikes). This enables system consider both the reasons of user's preferences and the taste of the user. We also use weights of attributes using frequencies of their values, but together with ratings of the individual values of the attribute.

The system in [9] generates user profiles consisting of items rated by the user and property vectors each corresponds to one attribute of the items. In each property vector in a profile, there are weights of individual values this attributes can have. Those weights are calculated using the ratings of that user has given to items. For example, when recommending movies, the attributes of a movie can be genre, director and actor. Therefore, the user profile generated by the system will contain the ratings of the movies and three property vectors. The system first detects the movies with ratings above some threshold defined. For those movies, it finds the values of each attribute and the frequency of these values. For genre aspect, it puts the genres of the movies in “genre” vector and calculates the weight of each according to how many movies have that genre.

Our system also contains item ratings and user preferences together in user profiles. Our user preferences correspond to property vectors suggested in this work. The difference is that we not only give weights to values of attributes but also to the attributes themselves as the previous work does. Furthermore, we consider the items above a threshold value only while calculating the weights of the attributes. When calculating the ratings of the individual values of attributes, we take into account all the items rated by the user, in order to represent both positive and negative opinion about them.

In [5], semantic technologies are used. The profiles are represented as ontologies and divided into areas of interest of users. The system first detects the concepts in the domain ontology and relationships between them. Using the definitions of items high rated by a user and concepts in domain, it generates a layered user profile where each layer defines the preferences of the user about one concept or a few related concepts.

This approach is especially beneficial when lots of attributes and individuals exist that are related to each other with various kinds of relationships. They give the example when photographs are examined by users and interests of people are extracted with the help of the objects inside them. But when this approach is applied to the movie domain, a limited number of attributes and individual types are used and it is concluded that the system does not benefit much from its layered structure. We have also designed our profile ontology to be able to include more than one domain to represent users’ preferences over different item sets when needed.

COLLABORATIVE FILTERING STRATEGY

Recommender systems also differ in how and when they use the CF mechanism. While CBF methods are pretty straightforward, there are different approaches to using collaborative properties of a system.

As mentioned earlier, [11] uses collaborative rating of the items as a supporting factor to its CBF while [14] and [15] adds basic content-based features to their base CF mechanism. And the work in [7] switches to CF when CBF is not successful.

Here, we discuss systems that follow collaborative via content paradigm. This is the method we use in our system in which we use content based user profiles in collaborative filtering.

[5] and [9] cluster users after generating user profiles. Clustering is grouping the users that show similar tastes and decision patterns together to form a community. It is a popular idea that recommendation is a group work; people's preferences are affected by not just individuals around him but the community they belong to. Beside that arguable assumption, clustering is effective in the means of computation. Instead of finding the neighbours of each user at runtime every time the system makes recommendations, it detects similar users and keeps them in a group. Furthermore, with the help of user profiling, because systems are able to better understand users, they are able to create more meaningful communities. For these reasons, we have found clustering of user profiles more efficient. After clusters are generated, the individuals in that group or the properties of the group itself can be used to evaluate items.

The work in [9] follows the latter approach. After generating user profiles, the system clusters users and creates group profiles. A group profile represents the rating patterns of its members. Recommendations are made to a user using both his profile and the group profile. When testing a candidate against group profiles, the user's and candidate's similarity to that profile is taken into consideration.

A work following the former approach [5] uses a different clustering mechanism as a result of its different user profile structure. The system generates layered user profiles. Each layer corresponds to user's preferences over some related concepts of the domain. The purpose of this layered structure is to allow people get opinions of the other that they are similar in the means of these concepts. Two people may not be similar in all areas, but their similarities in some are not omitted. To provide this, users are grouped in concept clusters and each user is a member of these clusters at some degree of similarity. While evaluating an item on behalf of a user, the system uses the membership of the user and the item to each cluster, and the ratings of the other users in these clusters proportional to their membership degrees.

We also use clustering in our collaborative filtering strategy. We put the users that have common concepts in their profiles together. These common concepts generate cluster profiles. But we do not define membership thresholds to every cluster. We put a user to one cluster that his profile shows most similarity with. The comparison with the work in [5] also affirms the success of this design principle of ours. Evaluations of items using only the most similar cluster to a user are more successful than the other case.

To improve the accuracy of clustering phase and the performance of the system, we used some mechanisms to reduce sparsity and the space needed for the calculations. We generated profile vectors in order to use only the most relevant attributes of user profiles in clustering. This reduces the memory used by the system. We also define cluster profiles as attribute vectors. A cluster profile consists of only the most popular attributes shared by most of the users in that cluster. This reduces the sparsity problem. The strategies for improving the performance are explained below.

To improve the recommendation performance, Amazon uses item-oriented similarity [17, 18] instead of user-oriented similarity. It generates an item similarity table to determine the most similar items. Two types of tables are generated. The first one contains the items rated by each user. The second one contains the items purchased by the users. The items that are purchased by a high number of users are determined to be popular. The popular items are put in item similarity table. This table contains similarity degrees of items. The similarity degree between two items is determined by the number of users that purchased both items. To make a recommendation to a user, it finds the similar items to the ones the user liked before. Then, the items are filtered by their similarity degrees and the most similar items are recommended to the user.

The strategy of eliminating unpopular items reduces the number of items used in neighbourhood, reducing the space required. It also assures that the similarity matrix is not sparse, so it gives more accurate neighbourhood relationships.

In our work, similarities are detected between users and not the items. But, since we use the content-based user profiles, attributes of the items affects the user-user similarity. To eliminate the unpopular attributes during clustering, we use a threshold value. The threshold value is the percentage of the number of users that give high ratings to the same attribute. The attributes that are preferred by the high percentage of the users in that cluster are put into the cluster profile. Then, when the clusters are updated, the similarity of a user to the clusters is determined using these reduced set of cluster profiles.

Matrix factorization [19] is a way to determine similarity degrees of contents efficiently, even in sparse data sets. It can be used to calculate the rating of an item for a user or to determine similarities between items, between users and between items and users. To detect item-to-item similarities, first, all items (or users) are defined using attribute (factor) vectors. Then two different attribute vectors are defined as axes of the matrix. Then each item is placed on a spot on the matrix according to the relevance of their attribute vectors to the axes. The items placed in the same areas are considered to be similar.

The important factor in this method is the separation of attributes to axes and their order on each axis. The attributes on one axis should be divergent and the item should belong to one attribute more than the others on the same axis. The item should also have an attribute placed in another axis so that it can be placed on the correct spot in the matrix. Furthermore, an attribute should be related to the other attributes on its left and right on the same axis, because the closeness of the items on the matrix indicates the similarity degree between those items.

In this thesis, we also generated profile vectors consisting of the attributes that are preferred by users and the cluster profiles. Cluster profiles are also attribute vectors consisting of the most popular attributes among the users in that cluster. Each user is included in the cluster his profile vector is most similar to. Different from matrix factorization that defines two dimensional matrix and place the user profiles in it, we use two dimensional user and cluster profiles and let the attribute groups be formed at runtime of the clustering algorithm. In this way, we do not need to consider relationships between attributes to organize them in axes before clustering; but our clustering phase requires iterations to make cluster profiles more accurate.

NORMALIZATION OF USER RATINGS

Since each user's rating habits and tolerance to things are different, it is sometimes misleading to evaluate the eligibility of items to all users with a simple rating threshold. The users' tolerance is one of the important factors that determines the coherence of the user profiling, clustering and recommendation processes. Rating normalization methods are used to handle this tolerance differences. In [20], a work that investigates two of the rating normalization techniques is presented: Gaussian normalization method and decoupling normalization method.

In Gaussian normalization method two main issues are pointed. One of them is the *shift of average rating*, which means that different users may give different ratings to the items they like. The second issue, *different rating scales*, indicates the difference in sensitivities of users ratings. A user may use three different rating values in the ten-scale rating system while the other may use ten of them.

In decoupling normalization method ratings are reduced to values to represent a good opinion or a bad opinion.

We used both of these normalization approaches for different purposes. Decoupling method is used in determining the popular items or properties of the items. We defined the threshold as the average of the ratings of the user and assumed the rating above that threshold indicates a positive opinion. The weights of the properties are affected directly by these popularities.

For calculating the rating of individuals that appear in items, we considered the issues pointed in Gaussian method. We eliminated the ratings that are radical from the rating pattern of the user and also used different rating thresholds to indicate positive opinion for different users.

CHAPTER 3

PROPOSED RECOMMENDATION FRAMEWORK

In this thesis, a content recommendation system using user profiles has been developed. In order to get more effective results, content based filtering and collaborative filtering are used together. The system implements three functionalities which are generating the user profiles, clustering users and recommending contents. Figure 1 shows the architecture of the proposed framework whose components are explained in detail in the following sections.

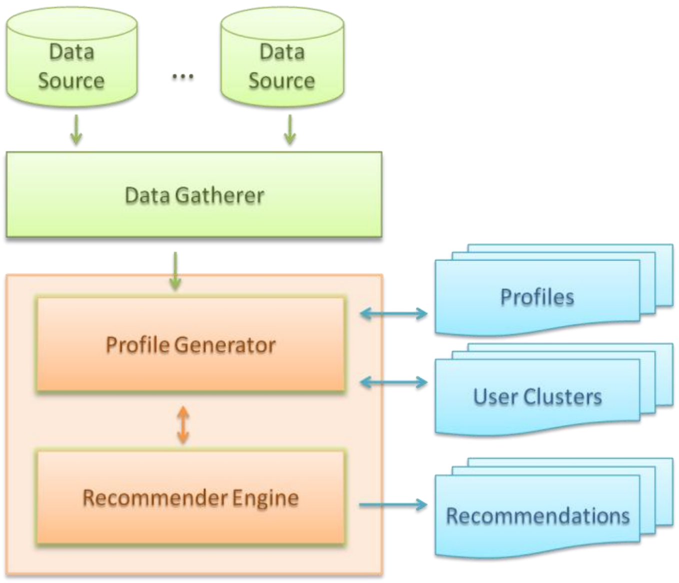

Figure 1 – Framework architecture

In this thesis work, a content recommendation system using user profiles is developed. To get a more efficient result, content based filtering and collaborative filtering are used together. System realizes three functions such as generating the user profiles, clustering users and recommending contents. Figure 1 shows the architecture of proposed framework and below, the components of the framework are introduced.

PROFILE GENERATOR

A user profile is generated by defining the preferences of the user over the contents to be recommended depending on the past behaviour of the user such as ratings of the items the user has evaluated before.

Profile Generator component consists of two main modules: *Profile Builder* which extracts profiles from the user history and *Profile Clusterer* which groups the users according to similarities of their profiles.

User Profile

A user profile defines the parameters that affect the user's choices. It shows which properties of a given concept the user takes into account while making decisions, how much those properties effect his choices, what he likes or may like about the item and how much.

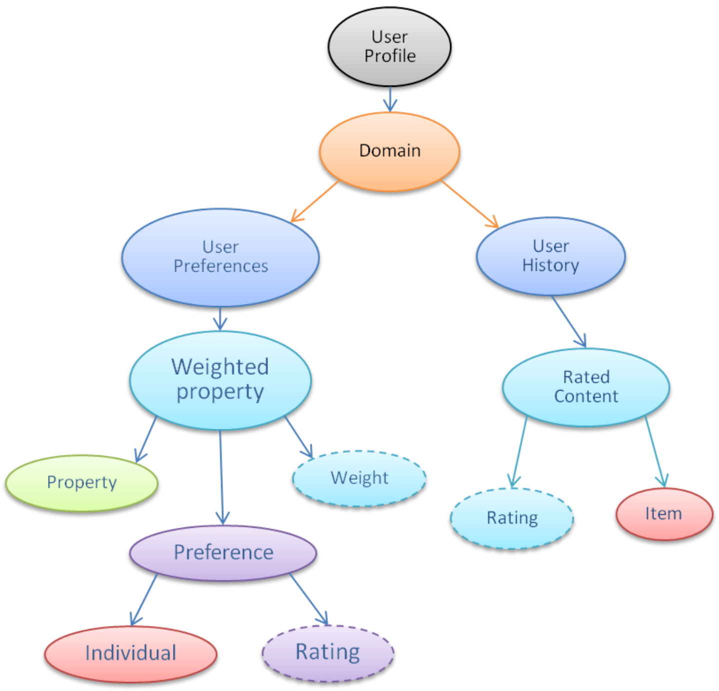

Figure 2 - User profile

Figure 2 shows the entities of a user profile. A user profile consists of the items the user has rated before and the ratings given, as well as the preferences of the user.

The items and their ratings form the *user history*. *User preferences* indicate what the user's criteria are for choosing the items they like and what properties of those items they like at what level. User's taste of different properties is defined by the *rating*s they give to the items that have those properties. The properties that have the same individual values in highly rated items are considered to have high influence over the choices of the user. That is called the *weight* of a property. For example if "Brad Pitt" is an actor of most of the movies the user liked, we can conclude that the user has chosen the movies according to their actors. Therefore, the weight of the actor property is high for that user.

What defines the liking of the individual values of the properties are directly the ratings of all the items they belong to. And each individual together with the property is a *user preference* and user preferences have ratings. The reason of not giving a rating to an individual itself is, the content that individual appears in may affect the liking of that individual for a user. For example a user may like the books written about 60's, while he doesn't like the books written in 60's. So there is a need to separate these two concepts. In Figure 3, you see that the rating is given to "written in 60's" and "written about 60's" preferences instead of individual "60's" because it has different ratings depending on the concept.

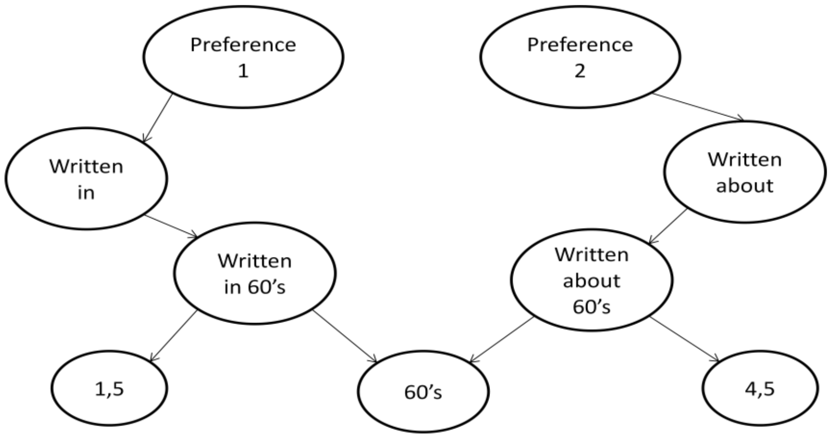

Figure 3 - User preferences with the same individual

We design the user profile consisting of several domain profiles. Since the user's choices of even the same items or properties may vary according to the domain they appear in, we gathered the information about the same kind of items under the related domain profile of the user. Although, in this work, we studied recommendation in only one selected domain, this profile ontology allows a user to have different profiles for different domains. The user profile ontology schema is included in APPENDIX A.

Profile Builder

Profile Builder is responsible for extracting the user’s preferences. To extract this information it takes the ratings of the user from the Data Gatherer component. Then, it examines the properties of the items that the user has rated.

For each property, it counts the number of times an individual value is repeated among the high rated items. The individual with the highest count is the most popular individual for that property. Most popular individual defines the importance factor so the properties can be scaled according to their effects to user’s choices. If an individual value appears in most of the items preferred by the user, the property in which this individual takes place is an important point for that user. The importance factor is represented as the weight of the property. The sum of the weights of all properties is 1. The rating of the item is calculating by adding up the ratings the item collects from its properties. The rating collected from a property is the average rating of the individual values of that property multiplied by its weight

The profile builder then calculates the ratings of user’s preferences. In order to calculate the rating of a preference, first the items that preference appears in are listed. The average of the rating these items gives the rating of the preference. Then, normalization is applied to this rating. The ratings that are too low or too high than the average rating calculated are omitted. If the ratings of items a preference appears in vary too much, this changes in rating are probably not related to this preference but other ones. After eliminating irregular ratings, system checks if there is enough information to rate the preference. If the preference appears in only one item it is not included in the profile; because one rating does not give enough information to rate a preference.

Profile Clusterer

Profile Clusterer groups similar user profiles together. It uses an adaptation of k-means clustering algorithm. Initial clusters are created by choosing random data points as cluster centroids but after the first iteration, clusters have centroids on their own. Profile vectors are used as data points and cluster centroids.

Clusters can be generated based on either the user's preferences over the content properties or the history of the user. These two methods differ in the generation of the user vectors. The rest of the clustering process is the same.

A profile vector based on user preferences is generated by selecting the most valued properties. First, properties are filtered by their weights to assure that only the preferences over the properties which take an important role in the user's decision making are represented in the profile vector. Then, the preferences are filtered by their ratings so that the vector represents what the user likes. The latter however, contains only the rated items that have a high rating from the user. The individual values of selected properties that exceeds user's rating threshold are included in user profile vector.

In Figure 4, a very small subset of a user profile is shown. Assume only the two most important properties are included in the profile vector, and the threshold for the user is 4. Then, the preferences "directed by" and "starring" is chosen because they are more important for the user. "hasGenre Comedy" preference is automatically not included in the profile vector regardless of its rating, which exceeds the threshold. Among other preferences, "Directed by George Clooney" is left out of the profile vector because of its low rating. In this example, the profile vector consists of "Directed by Stanley Kubrick" and "Starrign Geoprge Clooney" preferences.

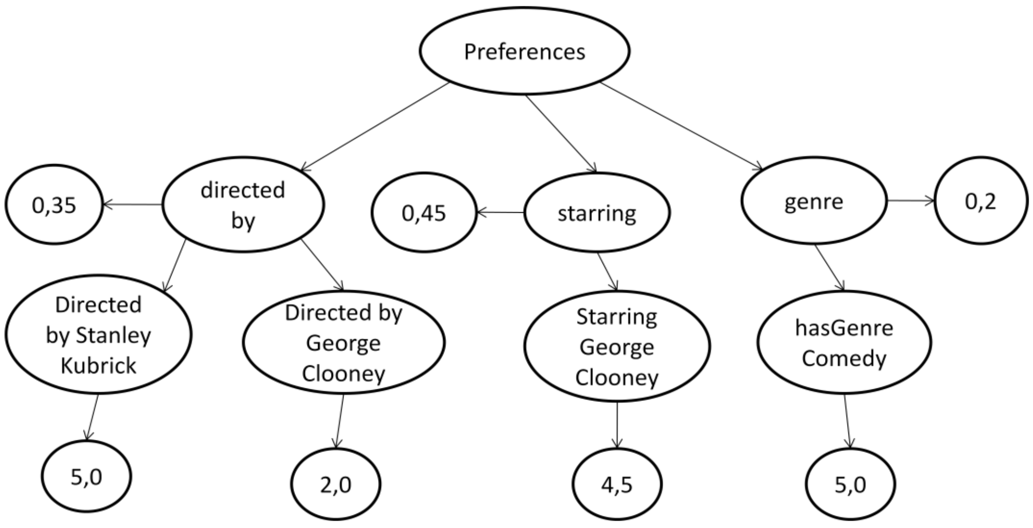

Figure 4 - User preference example

Clusters are created by choosing random profile vectors as centroids. Then all the profile vectors are added to the clusters that they have the most similarity with. The similarity is the ratio of mutual and diverging elements between the user vector and the cluster's centroid. After all users are added to clusters, the elements of the profile vectors (which are user preferences) are investigated. The most repeated elements among the vectors in a cluster are considered to form the profile vector that reflects the mutual taste of the users in that cluster and assigned as the new cluster centroid. This produces a less sparse vector by eliminating the individuals that is not rated by most of the users. The cluster centroid is updated in each iteration, and at the end of clustering, they show the characteristics of the clusters. The items that are highly rated by the users in the cluster are added to the cluster profile. The final cluster profile consists of a profile vector, users in this cluster and items that those users like.

The final products of the Profile Generator component are the user profile and user cluster ontologies as shown in APPENDIX B. Those are used by the Recommender Engine as inputs.

RECOMMENDER ENGINE

The recommender engine makes decisions about contents on behalf of the user. It recommends items to the users using their profiles and the clusters they belong to. First, it finds candidate items to recommend to the user. Candidates are the high rated items of the cluster that the user is a member of. Then, it compares the properties of each candidate with the user's preferences in the profile and estimates the ratings of these candidates. The rating of a candidate is the sum of ratings it gets from its each property. The rating gathered from each property should be a positive function of the ratings of individuals of that property appearing in the candidate item and the weight of that property for the user. In our implementation we used the average of the individual ratings multiplied by the weight of the property.

Algorithm: Evaluating a candidate

```
Extract the properties of the candidate c
For each property pc of the candidate c
  Find the property in the user profile pu
  For each value pind of pc
    Find the rating rind of pind in pu
    Calculate the rating of this property by a function of pu and all pinds
Add up the ratings of all properties
```

After finding the ratings of the candidates, the Recommender Engine applies a filter to them and recommends the ones that the user is expected to like. The filter we used is a rating threshold that is thought to be the lowest rating a user gives to the movies he likes.

DATA GATHERER

Data Gatherer is the component implementation which depends on the data sets and ontologies used. Data Gatherer collects the data needed by the system to generate user profiles, such as ratings of the user and contents of the domain studied.

It consists of modules that retrieves information from data sources and provides user ratings and item profiles to Profile Generator. It may consist of different modules such as ontology dictionaries and query interfaces. The main task of the component is to invert any data of any kind to ontologies the system uses.

The details of this component are explained in Chapter 4.

CHAPTER 4

A CASE STUDY – MOVIE RECOMMENDATION

In order to realize a recommendations system which is capable of understanding the intentions of users, reasons behind their choices and help them in their decision making, a well-defined domain and a sufficient amount of qualified user information is needed more than anything else. However, ready to use information either does not exist or is not accessible at that detail level. Therefore, we have used whatever the available information is and modified it to serve our purposes.

We have chosen the movie domain, since it is one of the ontologically well-defined domains with a relatively good amount of information and detail. It is also one of the most studied domains in research on recommendation so far. This allows us to learn and compare different kinds of strategies.

DATA SETS

MovieLens Data Set

We used MovieLens[2] datasets as the resource for user rating of the movies. MovieLens offers different data sets consisting of different numbers of users. In each data set, there is a file containing the users' ratings to movies and a file where the movie names are written. In ratings file, movies that users voted are represented

[2] http://www.movielens.org/

by their number, so they can be mapped to their names on the movies file. MovieLens uses a rating system of ten scales. Ratings vary between 0 and 5 including the halves.

However this data set does not include any further detail about movies except the genres. The names of the movies and the genres are same with the ones in IMDB. But since IMDB contains semi-structured data and does not have an ontological description of its contents; we wanted to search for another data set that has such properties.

Freebase Database

Freebase[3] is an open database of entities. It keeps structured data as a huge graph of connected contents and things called *entities* and their *properties*. In Freebase, every item is unique and has its own id but at the same time they can be instances of different types and can have different properties according to their roles. It gives the opportunity to view entities as RDF and provides a schema for its contents. This perfectly fits to our needs in implementing our recommendation system. It benefits from DBPedia[4] which is one of the biggest linked data sources on the web today. The reason for not using DBPedia but Freebase is that it contains more matching movies to the ones in MovieLens dataset. It also has an easier to use query api. Freebase is queried by the Metaweb Query Language[5] (MQL) which is similar to SPARQL[6] used for querying RDF.

[3] http://www.freebase.com/
[4] http://www.dbpedia.org/
[5] http://mql.freebaseapps.com/
[6] http://www.w3.org/TR/rdf-sparql-query/

SYSTEM ARCHITECTURE

The architecture proposed in Figure 1 is used for movie recommendation with some adaptations of modules for the movie domain. The final architecture is shown in Figure 5. Also, more details about the components and basic data flow are shown. The Profile Generator and Recommender Engine are remained unmodified and perform their standard tasks. The Data Gatherer and Ontology Reader/Writer components, which are the components that perform domain dependent tasks, are adapted to the data sources and the domain used. So, we give details of these two components in this chapter and discuss the details of others while we explain the functionalities of the system.

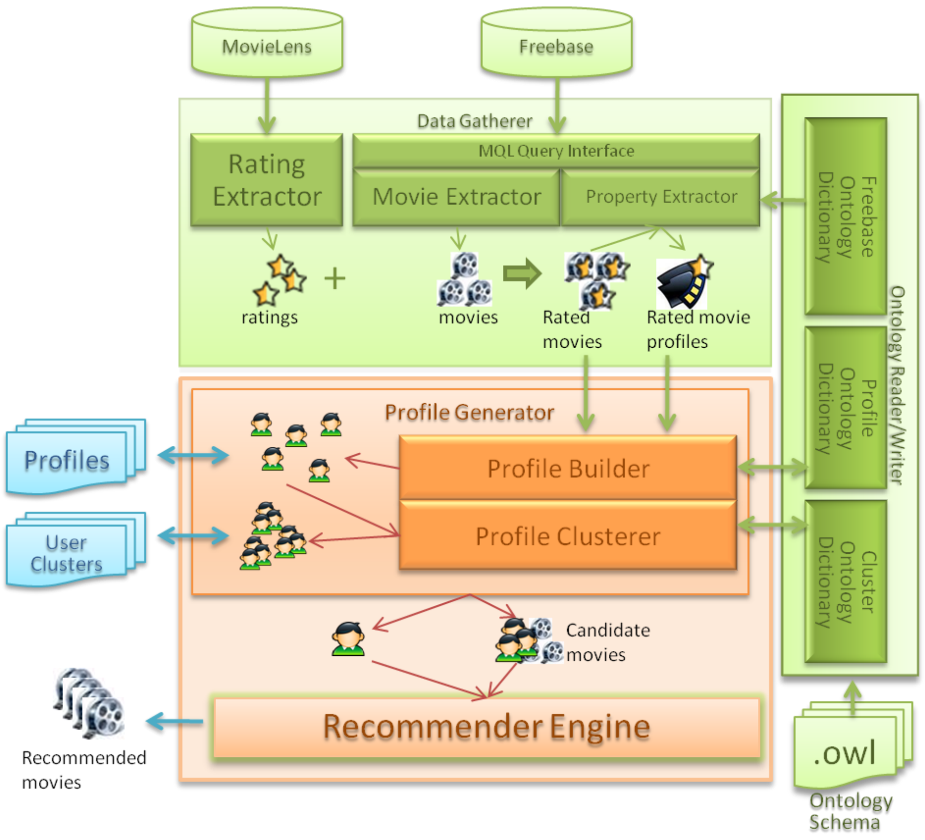

Figure 5 - Movie recommendation system architecture

Ontology Reader/Writer

Ontology Reader/Writer contains dictionaries of the concepts in the ontologies used by the system. *Freebase Ontology Dictionary* maps the properties of the movies requested by the system to their Freebase descriptions (URLs in Freebase database). Not all the concepts in Freebase movie content are used. We chose the properties we most benefit from in the recommendation process. The ones which play or expected to play more important roles in users' choices are included in the dictionary. These properties are "genre", "director", "actor", "character", "writer", "location", "composer", "subject". The properties like "language" or "country of origin" are excluded because they are common for most of the movies therefore they do not provide valuable information for understanding users' tastes.

The profile ontology dictionary helps the Profile Builder generate user profiles according to the profile ontology scheme given in APPENDIX A. And *Cluster Ontology Dictionary* helps Profile Clusterer generate cluster ontologies based on the ontology scheme given in APPENDIX B.

Data Gatherer

Data Gatherer extracts information needed from the external data sources. *Rating Extractor* extracts ratings of the movies by basically reading sequential files provided by MovieLens. *Movie Extractor* is responsible for finding the movies in Freebase by their names. *Property Extractor* finds the properties of the requested movie by querying Freebase. Both Movie Extractor and Property Extractor uses MQL query interface and Freebase Ontology Dictionary. *MQL Query Interface* is responsible for preparing and executing queries. According to the requested information, this component prepares the needed query in MQL syntax using relevant concepts from Freebase Ontology Dictionary and sends requests to Freebase.

The final product of the data gatherer is the set of movie profiles matched with the ratings a user has given. *Profile Builder* gets these profiles to generate the user's profile.

GENERATING USER PROFILES

The process of generating a user profile consists of gathering the existing information about the user, determining the reasons of the user's choices and finally defining user's taste. Profile Builder is responsible for this task and it conducts the process.

Profile Builder requests the rated movies of the user from Data Gatherer. In Data Gatherer, ratings from MovieLens are combined with their corresponding movie concepts in Freebase by matching the names of the movies. Although it doesn't seem to be an accurate way of finding the corresponding movie because of the possible differences in representing the movie names in different sources; with some combination of different queries, we found the results of this matching quite accurate. Those rated movie profiles form the user history. There is an example of user history with two rated movies in Figure 6.

```
<profile:UserHistory>

<profile:hasRatedContent>
<profile:RatedContent>
<profile:rating rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
>2.0</profile:rating>
<profile:item rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
>http://rdf.freebase.com/ns/en/waterworld</profile:item>
</profile:RatedContent>

</profile:hasRatedContent>
<profile:hasRatedContent>
<profile:RatedContent>
<profile:rating rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
>4.0</profile:rating>
<profile:item rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
>http://rdf.freebase.com/ns/en/disclosure</profile:item>
</profile:RatedContent>

</profile:hasRatedContent>
</profile:UserHistory>
```

Figure 6 - User history

After movie profiles are gathered, Profile Builder starts to analyse the taste of the user and to determine his preferences. First, it requests the properties of the movies from Data Gatherer. Then, weights of the features of the domain and the ratings of each individual appearing in the movies user rated are calculated. While calculating the rating of an individual, all movies that individual appears in is taken into account whereas while calculating the weight of a property, only the movies that are highly rated by the user are used. In this example, movies with ratings higher than or equal to 4 are considered highly rated. We give an example of building a user profile of a user that has watched and rated the movies in Table 1 and Table 2.

Table 1 - Rated movies

Movie No	Movie Name	Rating
1	WaterWorld	2
2	Disclosure	4
3	Braveheart	5
4	Die Hard: With a Vengeance	4
5	The Net	4
6	A Perfect World	3

Table 2 - Properties of the rated movies

Movie No	**1**	**2**	**3**	**4**	**5**	**6**
Directed by	Kevin Reynolds, Kevin Costner	Barry Levinson	Mel Gibson	John McTiernan	Irwin Winkler	Clint Eastwood
Starring	Kevin Costner, Jeanne Tripplehorn	Michael Douglas, Demi Moore, Dennis Miller	Mel Gibson, Sophie Marceau, Patrick McGoohan,	Bruce Willis, Samuel L. Jackson, Jeremy Irons,	Sandra Bullock, Jeremy Northam, Dennis Miller,	Kevin Costner, Clint Eastwood, Laura Dern
Genre	Science fiction, Adventure	Drama, War, Action, Adventure,	Drama War, Period piece, Adventure	Thriller, Action, Crime fiction	Drama, Action, Thriller	Drama, Crime fiction
Music by	Mark Isham, James Newton Howard	Ennio Morricone	James Horner	Michael Kamen	Mark Isham, Jeff Rona	Lennie Niehaus

For each property, how many movies an individual appears in and how many of them are high rated are determined. Using Table 1 and Table 2, we can generate Table 3 for the individuals of “genre” property of the movies rated. The genres “drama”, “action” and “thriller” are the most popular ones with the popularity of 3. This popularity is used as a coefficient while calculating the weights. Using these coefficients, the weight of each property is calculated provided that the sum of all weights is 1. Table 4 shows coefficients and the calculated final weights. We can say that, music composer and the director don’t affect user’s choices while genre has the largest impact on the user’s taste.

Table 3 - Popularity of individuals

Genre	Movies	Popularity
Science fiction	1	0
Adventure	1, 2	1
Drama	2, 3, 5, 6	3
War	2	1
Action	2, 4, 5	3
Thriller	3, 4, 5	3
Film adaptation	3	1
Crime fiction	4	1

Table 4 - Coefficients of properties

Property	Coefficient	Weight
Directed by	1	0,14
Starring	2	0,3
Genre	3	0,43
Music by	1	0,14

The weighted properties are represented in the user profile ontology as in Figure 7. “movie:hasGenre” is a subproperty of “profile:hasWeightedProperty” property in the user profile ontology in and only used in user profiles of movie domain. A weighted property node also includes the corresponding uri of the property in Freebase ontology.

```
<movie:hasGenre>

<profile:WeightedProperty>

<profile:property
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://rdf.freebase.com/ns/film/f
ilm/genre</profile:property>

<profile:weight
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.43</profile:weight>

</profile:WeightedProperty>

</movie:hasGenre>
```

Figure 7 - Weighted property

An individual belonging to a property together with its rating is referred as the user preference. Rating of a preference is the average rating of the movies that preference appears in. For example; the rating of “Kevin Costner” as “actor” of the movie is 2,5 since he acts in movies 1 and 6 which gets 2 and 3 from the user. Notice that, we don’t give rating to the individual alone but together with the property it belongs to. This enables us distinguish between how much the user likes “Kevin Costner” as an actor and as a director. You can see the preferences for “Kevin Costner” in different weighted properties in Figure 8.

```
<movie:starring>
<profile:WeightedProperty>
<profile:property
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://rdf.freebase.com/ns/film/
film/actor</profile:property>
<profile:weight
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.3</profile:weight>
<profile:hasPreference>
            <profile:Preference>
              <profile:rating rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
              >2.5</profile:rating>
              <profile:individual rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
              >http://movie_ont/Movie#starring/en/kevin_costner</profile:individual>
              <profile:item rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
              >http://rdf.freebase.com/ns/en/kevin_costner</profile:item>
            </profile:Preference>
</profile:hasPreference>
....
</profile:WeightedProperty>
</movie:starring>
<movie:directedBy>
<profile:WeightedProperty>
<profile:property
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://rdf.freebase.com/ns/film/
film/directed_by</profile:property>
<profile:weight
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.14</profile:weight>
<profile:hasPreference>
            <profile:Preference>
              <profile:rating rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
              >2.0</profile:rating>
              <profile:individual rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
              >http://movie_ont/Movie#directedBy/en/kevin_costner</profile:individual>
              <profile:item rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
              >http://rdf.freebase.com/ns/en/kevin_costner</profile:item>
            </profile:Preference>
</profile:hasPreference>
....
</profile:WeightedProperty>
</movie:directedBy>
```

Figure 8 - User preference

After the extraction of the user history, weights of the properties and the users' preferences, the process of building the user profile finishes. Figure 9 illustrates how a user profile ontology basically looks like.

```
<profile:UserProfile rdf:about="http://userProfile/item#User3262">
  ....
   <profile:hasDomainProfile>
     <profile:DomainProfile>
       <profile:hasUserPreferences>
         <profile:UserPreferences>
<movie:hasGenre>
                             ....
</movie:hasGenre>
<movie:starring>
              ....
</movie:starring>
....
         </profile:UserPreferences>
       </profile:hasUserPreferences>
       <profile:UserHistory>
<profile:hasRatedContent>
              ....
</profile:hasRatedContent>
....
  </profile:UserHistory>
     </profile:DomainProfile>
   </profile:hasDomainProfile>
</profile:UserProfile>
```

Figure 9 - User profile outline example

We used a fixed threshold value in our example, but the system calculates the threshold for each user at runtime if a threshold is not defined from outside. The threshold calculated is the average of the rating of the user to the items in user history. While defining weights of the properties only the items that have rating above the threshold is considered. While calculating the the ratings of individuals the average rating of the movies they appear in is the threshold. This threshold is used to detect radical ratings. The ratings that are too low or too high from the rating threshold are considered to be radical. These radical ratings are not used in calculation of the rating of the individual. We determined the radical ratings as the ratings that exceed the limits of -2 and +2 of the threshold.

CLUSTERING USERS

Our system uses collaborative filtering in determining the candidate movies to recommend to user. For this, adaptation of k-means clustering algorithm is used with two different strategies. One of them is to cluster users according directly to the movies they watched. Users who highly rated same movies are gathered in the same clusters. This is the pure collaborative strategy and does not use the user profiles except for the user history. This strategy can be followed by filtering the candidate movies by the user profile to provide a hybrid recommendation mechanism.

The other strategy is to cluster users according to their preferences in the user profile. This one allows us to add content-based properties to collaborative filtering as in “collaborative via content” paradigm. This strategy can also be followed by filtering the movies again by the user profile.

In both strategies, the clustering mechanism works the same way. Clustering is done by extracting a common characteristic profile of the similar users and grouping them around this profile, which is represented by profile vectors in our system. First, profile vectors of the users are generated. A user’s profile vector is the vector of top rated movies for the pure collaborative strategy. For “collaborative via content” strategy user’s preferences are filtered to form the profile vector that represents the user’s preferences in a simple form.

Preparing User Profile Vectors

Since it would be misleading to consider every preference as the characteristic of the user, we need to design a profile vector that reflects the reasons of the user’s preferences and the taste of the user. So, the preferences of the user are filtered first by the weights of the properties and later by the ratings of the preferences. Top rated preferences in the highest weighted properties are taken to form the profile vector. The filter might be either a constant threshold value or determined specifically for each user automatically by the system according to the weights of the properties and ratings of the user. The ratings the users give to the movies they like may vary, so it may be inconvenient to determine a threshold that indicates the user's high rating. A user may give 4 points to an average movie whereas another user may give 3 to the movies he most likes.

For our user represented in Table 1, we may simply say 4 is a good rating and shows us the user likes the movie. And the properties “genre” and “starring” should be considered while creating the profile vector because they are the properties that affect the user's taste. Therefore, the preferences of these two properties that have a minimum rating of 4 are put in the profile vector of this user.

Generating Cluster Profiles

After profile vectors of the users are generated, the users that have similar vectors are grouped together and their most common preferences form the characteristics of the cluster. This characteristic is represented as the profile vector of the cluster. Later, possible candidate movies are added to the cluster's profile. Those candidate movies are movies that are watched and highly rated by the several numbers of members of the cluster. This number may be determined according to the wanted number of candidate movies. If one wishes to recommend more candidates, the limit number can be kept low; else it can be increased. Figure 10 shows an example ontology of a cluster.

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:profile="http://userProfile#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:cluster="http://cluster#">
  <cluster:Cluster>
   <cluster:clusterID rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
   >1</cluster:clusterID>
   <cluster:item rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
   >http://rdf.freebase.com/ns/en/se7en</cluster:item>
   <cluster:item rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
   >http://rdf.freebase.com/ns/en/flirting_with_disaster</cluster:item>
   ....
   <cluster:hasDataPoint>
     <profile:UserProfile rdf:about="http://userProfile/item#User148">
       <profile:profileFile rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >C:\UserProfiles\freebase\train\UserProfile148.owl</profile:profileFile>
       <profile:userID rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
       >148</profile:userID>
     </profile:UserProfile>
   </cluster:hasDataPoint>
   ....
   <cluster:hasMean>
     <cluster:Mean>
       <cluster:hasIndividual rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
       >http://movie_ont/Movie#starring/en/kevin_spacey</cluster:hasIndividual>
       <cluster:hasIndividual rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
       >http://movie_ont/Movie#starring/en/pete_postlethwaite</cluster:hasIndividual>
       <cluster:hasIndividual rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
       >http://movie_ont/Movie#hasGenre/en/indie</cluster:hasIndividual>
       <cluster:hasIndividual rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
       >http://movie_ont/Movie#starring/en/r_lee_ermey</cluster:hasIndividual>
              ....
     </cluster:Mean>
   </cluster:hasMean>
  </cluster:Cluster>
</rdf:RDF>
```

Figure 10 - Cluster ontology example

RECOMMENDING MOVIES

For recommending movies to a user, Recommender Engine first determines candidates; then evaluate the candidate movies to check if they really fit the user's taste. Candidate movies are simply the movies that certain number of members of the cluster the user belongs to gave high rating. Since the tastes of the users in the same cluster and their reasons for liking a movie are similar, these candidates are mostly expected to be successful recommendations. However, to make a better recommendation, we filter the results by the user profile and rate each candidate movie on behalf of user, and then recommend them if their rating exceeds the high rating threshold of the user.

Assume we are trying to decide if we can recommend our user the movie "Dances with Wolves", the profile of which is shown in Table 5.

Table 5 - Profile of Dances with Wolves

Dances with Wolves	
Directed by	Kevin Costner
Starring	Kevin Costner, Mary McDonnell, Graham Greene
Genre	Western, Epic, Drama, War, Film adaptation, Adventure
Music by	John Barry

First, we find the individuals of each property of the film in user profile and get their rating. If the individual doesn't exist in users profile, user may either like it or dislike it. So, if we assume that the high rating threshold for our user is 4, we give 4 to the individuals that the user have not evaluated yet. This way, unrated individuals don't decrease the estimated rating of the movie, but still don't indicate positive rating. So, if we compare the movie profile with the user profile as in

Table 6, we give “Dances with Wolves” 3,66, which is below the threshold value. Finally, we don’t recommend the movie to the user.

Table 6 - Rating of Dances with Wolves

Directed by 0,14	Kevin Costner = 2	0,14 x 2 = **0, 28**
Starring 0,3	Kevin Costner = 2,5, Mary McDonnell = 4, Graham Greene = 4	0,3 x (2,5 + 4 + 4) / 3 =**1,05**
Genre 0,43	Western = 4, Epic = 4, Drama = 4, War = 4, Film adaptation = 5 , Adventure = 3,7	0,43 x (4 + 4 + 4 + 4 + 5 + 3,7) / 6 ≈**1,77**
Music by 0,14	John Barry = 4	0,14 x 4 =**0,56**
Total		**3,66**

If the candidate movie is “In Love and War” shown in Table 7, this movie is recommended to the user with rating 4,169, which exceeds the threshold.

Table 7 - Profile of In Love and War

Directed by 0,14	Richard Attenborough = 4	0,14 x 4 = **0,56**
Starring 0,3	Mackenzie Astin = 4, Chris O'Donnell = 4, Margot Steinberg=4, Sandra Bullock =4	0,3 x (4 + 4 + 4 + 4) / 4 = **1,2**

Genre 0,43	Drama = 4,
	Romance = 4,
	War =4,5,
	Period piece = 5,
	Biography = 4
		0,43 x (4 + 4 + 4,5 + 5 + 4) / 5 = **1,849**

Music by 0,14	George Fenton = 4	0,14 x 4 =**0,56**
Total		**4,169**

CHAPTER 5

EVALUATION

We tested our movie recommendation system to determine the effects of different parameters on the recommendation success. Then we compared different recommendation methods and discuss how our approach affects recommendations. We used MovieLens ratings and Freebase ontologies in our tests. We extracted the histories of the first 500 users from MovieLens data and movie profiles from Freebase. We divided the ratings into train and test datasets. Our training data contains maximum 30 ratings per user and test data contains 10 ratings per user. The training and test data for a user are generated by choosing random movies rated by the users from MoviLens data.

Four different filtering strategies that are used in evaluation are:

- Pure Collaborative (CF) : Only the ratings of the users in MovieLens dataset is used. The clusters are formed by grouping users that rated common movies. The movies rated by other users in that cluster that are not seen by the user are recommended.

- Collaborative + content-based staged process (CFCB) : The clusters formed by CF as mentioned above are used. But, the user profiles are also generated. The movies in the cluster are chosen as candidates. These candidates are filtered by the eligibility to the user profile.

- User profile based collaborative (UP-CF): The users with similar preferences are clustered together and the movies in their clusters are recommended to the users.

- User profile based collaborative + content-based staged process (UP-CFCB) : The candidates are chosen from the clusters formed by UP-CF, as mentioned above. Then they are filtered by the user profiles.

Below, we define some parameters we used to evaluate our results.

- Recall: The ratio of matched items recommended by the system that also exist in test data to the number of items in test data.
- Precision: The ratio of successful recommendations to all recommendations.
- MAE (Mean Absolute Error): The average difference of the ratings calculated by the system and real ratings. This difference is always positive.
- F1: Recall and precision can be conflicting parameters. Recall usually increases as the number of recommended movies increases, because there is a better chance of finding the movies in test data among larger movie set. If system makes more unsuccessful recommendations than successful recommendations, the precision decreases. The aim is to maximize these two. So, we used a formula defined in [10] that includes both precision and recall.

$$F1 = \left(\frac{2 \times recall \times precision}{recall + precision}\right)$$

We applied the same rules to calculate recall, precision and F1 for unsuccessful recommendations as well. The “s/u” code used in the graphs indicates the ratio of the related evaluation parameter of successful recommendations to unsuccessful recommendations.

Our hybrid recommendation strategy first clusters user profiles, then determines candidate movies using the cluster profiles. Finally, it filters results against users’ profiles and the filtered movies are recommended. The recommendation performance is highly affected by the clustering procedure. So, we investigated the effects of different clustering parameters to the success of the system. We made tests using different user groups. We included the results of one of these tests with 100 users grouped into 10 clusters. These results reflect the common finding of the tests done unless mentioned otherwise.

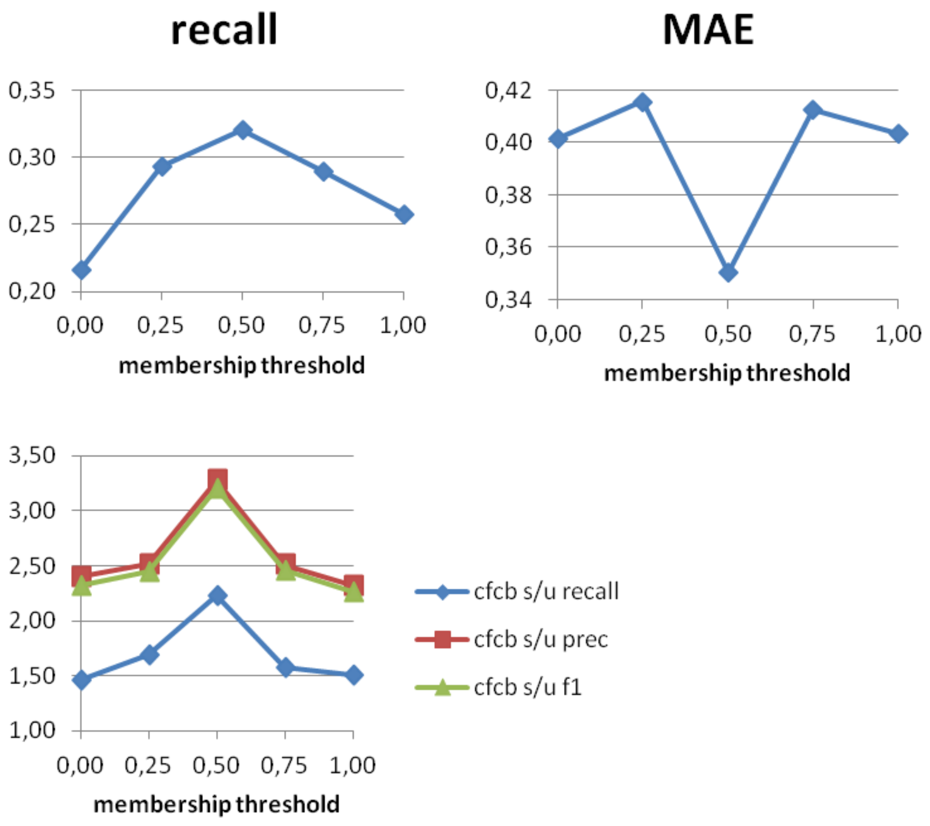

Figure 11 - Effect of membership threshold

One of the parameters that determine the cluster profiles is the membership threshold. It is a manually defined threshold value that indicates the popularity of a preference in a cluster. This popularity of a preference is the percentage of the users that includes that preference in their profile. The preferences whose popularity exceeds that threshold is included in the cluster. As seen in Figure 11, low or high values decrease the performance of the system. The best value is 0,5. This means that the common taste shared by half of the community is proper to reflect the opinions of this community.

For clustering profiles, profile vectors are created first. These profile vectors can be created using different features in user profiles. We chose the features with highest weights. Below, in Figure 12, we show how the number of features in profile vectors affects the recommendation success.

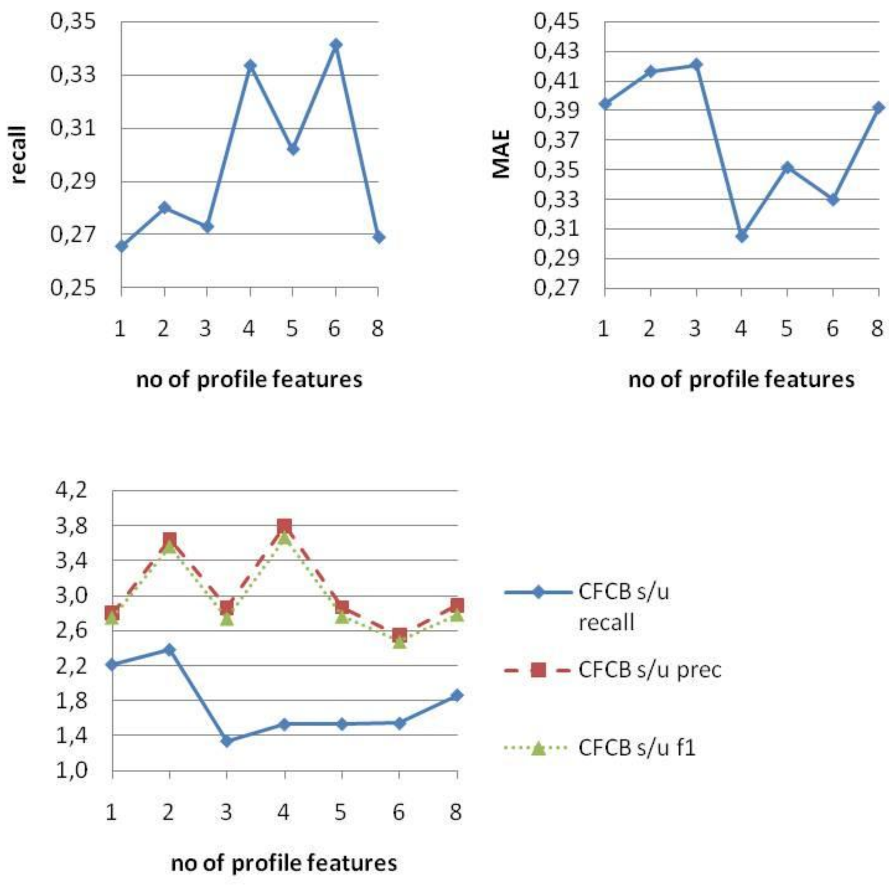

Figure 12 - Effect of number of profile features

We ran the test with different numbers of users and different numbers of clusters. There is not an exact trend of this parameter. The result for 100 users and 10 clusters is shown in Figure 12. The total number of features we extracted from freebase per person is 8 and when 4 of them per each user are used, system shows high performance by means of MAE and recall; however, number of unsuccessful recommendations also increases compared to when 1 or 2 features are used. It is because when 4 features are in user profiles, the features that are less important for a person is taken into consideration. A cluster contains people with different priorities. So the system may conclude that some features are more important than they really are which would lead it to make more unsuccessful recommendations. On the other hand, because it clusters users using more preferences, it groups more similar users together increasing the recall.

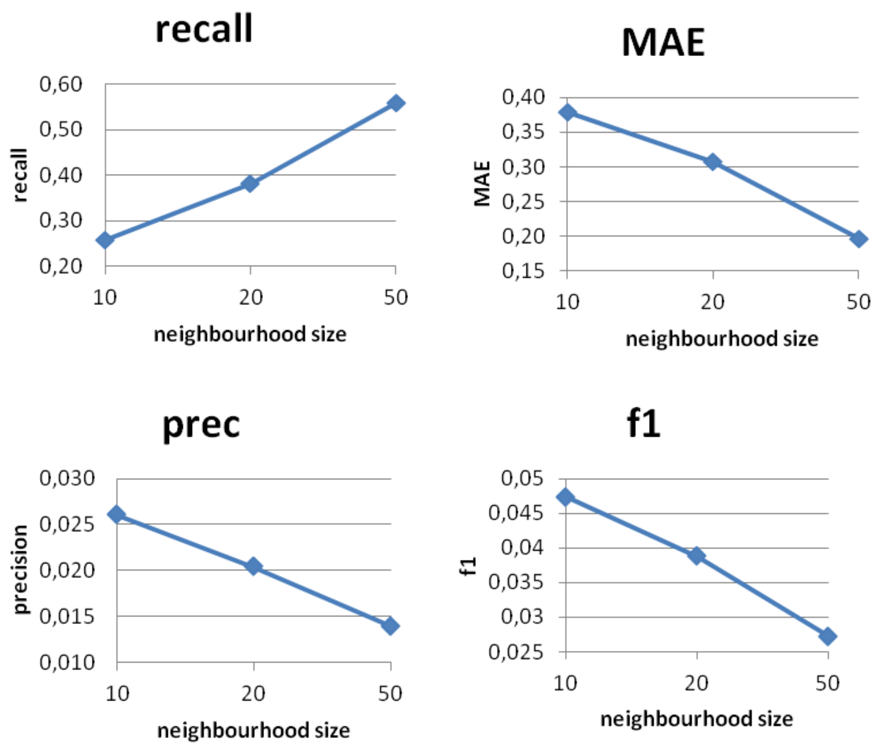

Figure 13 - Effect of neighbourhood size on recommendations with collaborative via content paradigm

Another aspect that affects the performance of the system is the number of clusters. This parameter may also be interpreted as the neighbourhood size of a user or size of the community. As the number of clusters increases, the number of neighbours decreases. The results are seen in Figure 13. We ran this test with different numbers of users and clusters. As the number of neighbours increases, system is more successful in finding the movies in test data and makes more accurate evaluations. However, the precision decreases because there is larger number of candidate movies. The accuracy of recommendations – rates of successful and unsuccessful recommendations – on the other hand, doesn't show a regular trend but tends to be increasing with the neighbourhood size.

The neighbourhood size affects the performance of the pure collaborative filtering algorithm too. We tested four different recommendation methods and observed the same effect on performance. There are conflicting results, recall and MAE improve by the neighbourhood size while precision decreases, so the decision about this parameter can be made considering also the non-functional requirements of the system. As the neighbourhood size increases the number of candidate movies and the elements in cluster profiles increases. Therefore, it takes more time and space to cluster users and evaluate movies.

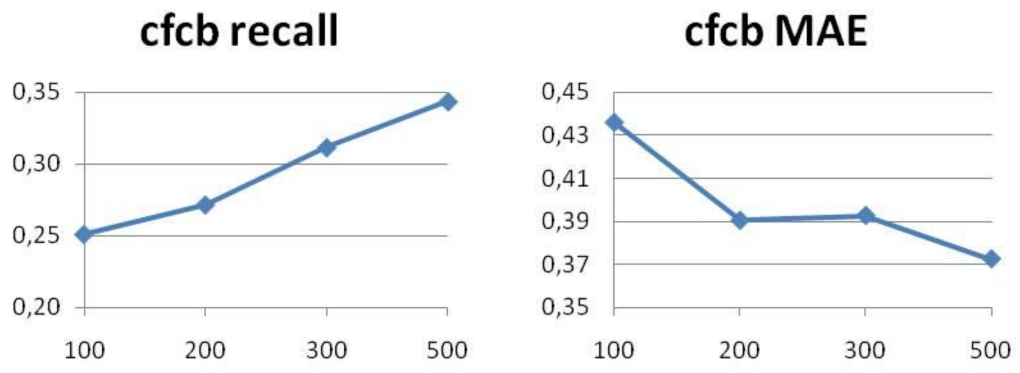

Figure 14 - Effect of number of users to recommendation when the neighbourhood size is fixed to 10

When the neighbourhood size is fixed, total number of users affects the performance as seen in Figure 14. It improves the recall and MAE regardless of the number of features in profile vector. However, it does not have a big effect on precision.

We evaluated our pure content-based filtering strategy using MAE, recall and precision. For each user, we got candidate movies from test data and evaluated them. The results for the first 500 users in MovieLens dataset are shown in Figure 15. Total MAE is the error in rating of all candidate movies. Recommendation MAE is the error in rating of movies recommended by the system.

Precision	**Recall**	**Total MAE**	**Recommendation MAE**
0,85832	0,97786	0,65846	0,530693

Figure 15 - UP-CBF performance

The hybrid filtering method used in this system shows better improvement over pure content-based filtering in terms of MAE than the related work [5] it was most influenced from. In [5], system shows approximately %15 of improvement in MAE when 100 users are used with %75 of them in training data and %25 in test data. Under these conditions, we showed in Figure 16 that our system shows up to %52 improvement, and %25 in the chosen case of 100 users with 10 clusters.

We compared the effect of hybrid strategy against pure content-based filtering as shown in Figure 16. The change in MAE between 100 and 500 user datasets using CBF does not point to any design issues; it is caused by the use of different user profiles. The improvement in MAEs for different recommendation techniques and different datasets are shown by the percentage values. UP-CFCB performs better than pure CBF and CFCB. We observe the best MAE results when smaller numbers of clusters are used, forming clusters with more users. But, as we mentioned before, this also reduces the precision.

We observe that, as the input user dataset gets bigger (more user is used), the performance of the system increases and CFCB begins to show closer performance to UP-CFCB. Therefore, we can conclude that the negative effect of the sparsity is reduced.

	CBF **100 users**	**CFCB** **100 users** **10 clusters**	**UP-CFCB** **100 users** **10 clusters**	**UP-CFCB** **100 users** **2 clusters**
Recommendation MAE	0,40619	0,379515 %6,5	0,305016 %25	0,196906 %52
	CBF **500 users**	**CFCB** **500 users** **50 clusters**	**UP_CFCB** **500 users** **50 clusters**	**UP-CFCB** **500 users** **5 clusters**
Recommendation MAE	0,530693	0,379435 %28	0,348087 %34	0,177864 %66

Figure 16 - Improve in MAE of different recommendation methods

Finally, we applied different filtering strategies and observed the difference our proposed strategy makes. In tests of each recommendation model with each number of users, the neighbourhood size for clusters is fixed to 10. UP-CF and UP-CBF, the membership threshold is %50, number of profile features is 2.

The graph in Figure 17 shows the accuracies of different recommendation strategies with different numbers of users. It shows the rate of the recall value of successful recommendations to the recall value of unsuccessful recommendations.

It is observed that applying collaborative filtering as a second filtering method improves accuracy. However, it decreases the recall as seen in Figure 18, because it reduces the number of recommended movies by applying a second filter. User profiling, on the other hand, improves the recall.

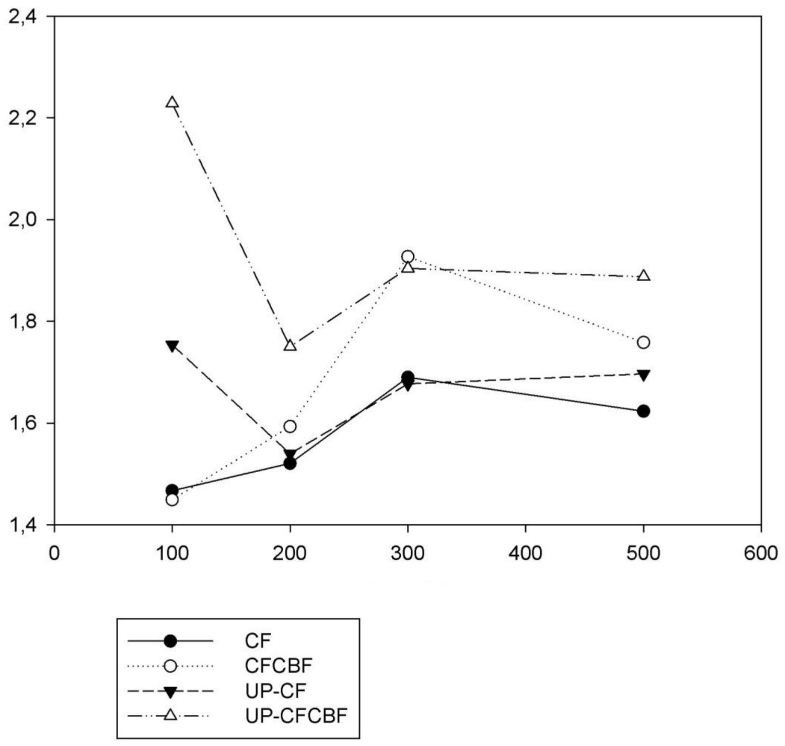

Figure 17 - Comparison of accuracy of recommendation strategies

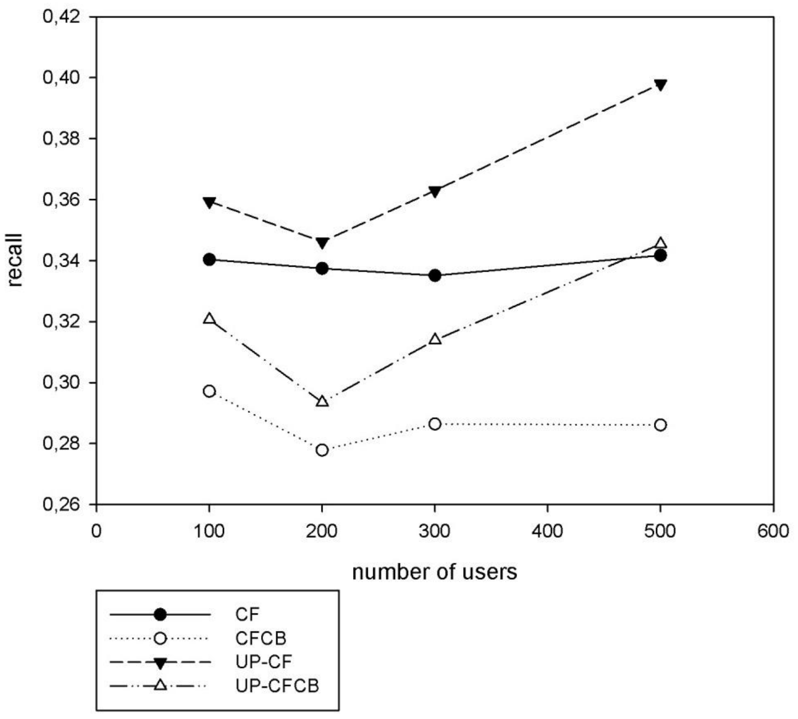

Figure 18 - Comparison of different recommendation strategies – recall

CHAPTER 6

CONCLUSION

In this thesis, different aspects of recommendation, different kinds of recommendation methods and the contributions of user profiling are investigated. The other aim of this work is to propose a general-purpose recommendation system that can be applicable to different problems.

A framework for hybrid recommendation systems is proposed. The framework consists of modules for user profiling, clustering and a recommendation engine, implementations of which can be directly used in realizing different recommendation systems. It also defines interfaces for adopting different domains and ontologies to the system. Therefore, the proposed system is a general purpose recommendation system.

The other characteristic of the work is the user profile structure. The generated user profiles reflect both the taste of the user and the aspects of the domain they give importance to. This improves the accuracy of the recommendations made and gives the power to explain the reasons of them.

As a case study, a movie recommendation system is implemented using the framework proposed. MovieLens ratings and Freebase movie onotologies are used as data sources. There is not a standard criterion to evaluate and compare recommendation systems that is available for every work done; but most of the researches use MAE to express the performance of their methods. Our hybrid recommendation system makes better improvement in MAE over content-based filtering than the work it was most influenced from [5].

We tested our system to understand the effects of design decisions we made. The clustering phase has a great impact on the recommendation performance. Generating not too specific cluster profiles but including the common preferences of a majority of the users in them results better clustering. Using larger datasets as input – greater number of users and greater number of ratings per user – improves the recommendation performance.

We observed that user profiling shows positive effect on recommendation. It eliminates the wrong suggestions and improves the accuracy when user profiling is used in collaborative filtering or after collaborative filtering as the second filtering method.

This system can be improved by adding mechanisms for more complex inferences and benefiting from semantic technologies such as semantic spreading. Semantic spreading will enable system discover new relationships between objects. Using these relationships system will be able to produce similar preferences to the ones in user profiles. When more individuals and preferences are included in the user profile, its ability to evaluate items accurately is expected to increase.

User profile based collaborative filtering method may be improved by adding a mechanism to include the content-based evaluations by the neighbours of the user. Neighbours can evaluate the item itself or its attributes separately to help recommendation process.

Matrix factorization methods [19] can be used for the clustering phase. This may reduce the time spent for clustering; because the clustering may be done in one iteration instead of k-means algorithm which is done in several iterations. The semantic power of the ontologies may be used for separating the attributes to axes and ordering attributes on the same axis. The drawback is that, matrix factorization is more effective when one aspect of an item type is used for defining similarities. Preferences over different attributes may be independent from each other and representing them in the same matrix may not be meaningful.

Implementing different recommendation systems using different data sources based on the proposed framework, will test its flexibility and modularity. By modifying the Data Gatherer component, the system can be used for recommending different types of items in different domains. New ontology dictionaries can be written and attached to the system. The system can use any data that are defined in meta-level. The user profiles can be enriched by adding different domain profiles. By default, the system will treat these profiles as if they are totally independent from each other. But, there may be some cases these profiles can benefit from some or all of the preferences defined in each other. The strategies for detecting these properties and using them in cross-domain recommendations are expected to increase the performance of the system.

REFERENCES

[1] G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems : A Survey of the State-of-the-Art and Possible Extensions," *Knowledge Creation Diffusion Utilization*, vol. 17, 2005, pp. 734-749.

[2] E. Vozalis and K.G. Margaritis, "Analysis of Recommender Systems' Algorithms," HERCMA (Hellenic European Research on Computer Mathematics & Its Applications), Athens, Greece, 2003, pp. 732-745..

[3] Yolanda Blanco-Fernandez, José J. Pazos-Arias, Alberto Gil-Solla, Manuel Ramos-Cabrer, Martín López-Nores, Jorge García-Duque, Ana Fernández-Vilas, and Rebeca P. Díaz-Redondo, "Exploiting synergies between semantic reasoning and personalization strategies in intelligent recommender systems: A case study," *Journal of Systems and Software*, vol. 81, 2008, pp. 2371-2385.

[4] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, 2002, p. 331–370.

[5] I. Cantador, A. Bellogín, and P. Castells, "A multilayer ontology-based hybrid recommendation model," *AI Communications*, vol. 21, 2008, p. 203–210.

[6] S. Spiegel, J. Kunegis, and F. Li, "Hydra: A Hybrid Recommender System," *CIKM (Conference on Information and Knowledge Management) Workshop CNIKM (Complex Networks in Information & Knowledge Management)*, 2009, pp. 75-80.

[7] Y.B. Fernández, J.J. Arias, M.L. Nores, A.G. Solla, and M.R. Cabrer, "AVATAR : An Improved Solution for Personalized TV based on Semantic Inference," *IEEE Transactions on Consumer Electronics, Feb.2006, 52(1):223–231*.

[8] N. Good, J.B. Schafer, J.A. Konstan, A. Borchers, B.M. Sarwar, J. Herlocker, and J.T. Riedl, "Combining collaborative filtering with personal agents for better recommendations," *in Conference of the American Association of Artifical Intelligence AAAI-99*, pp. 439-446.

[9] Q. Li and B.M. Kim, "Constructing User Profiles for Collaborative Recommender System," *In Proc. of Sixth Asia Pacific Web Conf., 2004, pp. 100 - 110*.

[10] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos, "Feature-Weighted User Model for Recommender Systems," *Proceedings of 11th International Conference on User Modelling (UM 2007), Volume 4511, pages 97-106.*

[11] X. Li and T. Murata, "A Knowledge-based Recommendation Model Utilizing Formal Concept Analysis and Association," *Proceedings of ICCAE2010 (The second IEEEInternational Conference on Computer and Automation Engeneering), 2010*

[12] Y. Wang, N. Stash, L. Aroyo12, L. Hollink, and G. Schreiber, "Using Semantic Relations for Content-based Recommender Systems in Cultural Heritage," *Workshop on Ontology Patterns*, 2009, p. 16.

[13] P. Heim, S. Lohmann, and T. Stegemann, "Interactive Relationship Discovery via the Semantic Web," *7th Extended Semantic Web Conference (ESWC2010)*, 2010.

[14] B. Mobasher, X. Jin, and Y. Zhou, "Semantically Enhanced Collaborative Filtering on the Web," *Lecture Notes in Computer Science*, 2004.

[15] P. Mylonas, G. Andreou, and K. Karpouzis, "A Collaborative Filtering Approach to Personalized Interactive Entertainment using MPEG-21," *Proceeding of the 2007 Conference on Emerging Artificial intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in Ehealth, Hci, information Retrieval and Pervasive Technologies I. Maglogiannis, K. Karpouzis, M. Wallace*, vol. 160, 2007, pp. 173-191.

[16] D. Weiß, J. Scheuerer, M. Wenleder, A. Erk, M. Gülbahar, and C. Linnhoff-popien, "A User Profile-based Personalization System for Digital Multimedia Content," *In Proceedings of DIMEA (Digital Interactive Media in Entertainment and Arts)*, 2008, pp.281-288.

[17] G. Linden J. Jacobi and E. 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

[18] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, 2003, pp. 76-80.

[19] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *IEEE Computer, vol. 42*, 2009, pp. 30-37.

[20] R. Jin and L. Si, "A study of methods for normalizing user ratings in collaborative filtering," *Proceedings of the 27th Annual International ACM SIGIR (Special Interest Group of Information Retrieval) '04*, 2004, p. 568.

APPENDIX A

USER PROFILE ONTOLOGY

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns     ="http://userProfile#"
xml:base  ="http://userProfile#"
xmlns:owl ="http://www.w3.org/2002/07/owl#"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
<owl:Class rdf:ID="UserProfile"/>
<owl:Class rdf:ID="DomainProfile"/>
<owl:Class rdf:ID="UserHistory"/>
<owl:Class rdf:ID="RatedContent"/>
<owl:Class rdf:ID="UserPreferences"/>
<owl:Class rdf:ID="WeightedProperty"/>
<owl:Class rdf:ID="Preference"/>

<owl:ObjectProperty rdf:ID="hasDomainProfile">
<rdfs:domain rdf:resource="#UserProfile" />
<rdfs:range  rdf:resource="#UserDomainProfile" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasUserHistory">
<rdfs:domain rdf:resource="#UserDomainProfile" />
<rdfs:range  rdf:resource="#UserHistory" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasRatedContent">
<rdfs:domain rdf:resource="#UserHistory" />
<rdfs:domain rdf:resource="#Preference" />
<rdfs:range  rdf:resource="#RatedContent" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="item">
<rdfs:domain rdf:resource="#RatedContent" />
<rdfs:range  rdf:resource="xsd:anyURI"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="rating">
<rdfs:domain rdf:resource="#RatedContent" />
<rdfs:domain rdf:resource="#Preference" />
```

```
<rdfs:range  rdf:resource="xsd:float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="weight">
<rdfs:domain rdf:resource="#WeightedProperty" />
<rdfs:range  rdf:resource="xsd:float"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasUserPreferences">
<rdfs:domain rdf:resource="#UserDomainProfile" />
<rdfs:range  rdf:resource="#UserPreferences" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasWeightedProperty">
<rdfs:domain rdf:resource="#UserPreferences" />
<rdfs:range  rdf:resource="#WeightedProperty" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="property">
<rdfs:domain rdf:resource="#WeightedProperty" />
<rdfs:range  rdf:resource="xsd:anyURI"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasPreference">
<rdfs:domain rdf:resource="#WeightedProperty" />
<rdfs:range  rdf:resource="#Preference" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="individual">
<rdfs:domain rdf:resource="#Preference" />
<rdfs:range  rdf:resource="xsd:anyURI"/>
</owl:DatatypeProperty>
</rdf:RDF>
```

APPENDIX B

CLUSTER ONTOLOGY

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns     ="http://cluster#"
xml:base  ="http://cluster#"
xmlns:profile ="http://userProfile#"
xmlns:owl ="http://www.w3.org/2002/07/owl#"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">

<owl:Class rdf:ID="Cluster"/>
<owl:Class rdf:ID="Mean"/>
<owl:Class rdf:ID="profile:UserProfile"/>

<owl:DatatypeProperty rdf:ID="isInCluster">
<rdfs:domain rdf:resource="profile:UserProfile" />
<rdfs:range  rdf:resource="xsd:anyUri" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="userID">
<rdfs:domain rdf:resource="#profile:UserProfile" />
<rdfs:range  rdf:resource="xsd:int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="profileFile">
<rdfs:domain rdf:resource="#profile:UserProfile" />
<rdfs:range  rdf:resource="xsd:string"/>
</owl:DatatypeProperty>


<owl:DatatypeProperty rdf:ID="clusterID">
<rdfs:domain rdf:resource="#Cluster" />
<rdfs:range  rdf:resource="xsd:int"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="hasMean">
<rdfs:domain rdf:resource="#Cluster" />
<rdfs:range  rdf:resource="#Mean" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="item">
<rdfs:domain rdf:resource="#Cluster" />
```

```
<rdfs:range  rdf:resource="xsd:anyURI"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasIndividual">
<rdfs:domain rdf:resource="#Mean" />
<rdfs:range  rdf:resource="xsd:anyURI"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="hasDataPoint">
<rdfs:domain rdf:resource="#Cluster" />
<rdfs:range  rdf:resource="profile:UserProfile"/>
</owl:ObjectProperty>
</rdf:RDF>
```