OPENMORE: A CONTENT-BASED MOVIE RECOMMENDATION SYSTEM

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

BY

ÖZNUR KIRMEMİŞ

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

MAY 2008

Approval of the thesis:

OPENMORE: A CONTENT-BASED MOVIE RECOMMENDATION SYSTEM

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

Prof. Dr. Canan Özgen ______________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Volkan Atalay ______________________
Head of Department, **Computer Engineering**

Dr. Ayşenur Birtürk ______________________
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Faruk Polat ______________________
Computer Engineering Dept., METU

Dr. Ayşenur Birtürk ______________________
Computer Engineering Dept., METU

Assoc. Prof. Dr. Nihan Kesim Çiçekli ______________________
Computer Engineering Dept., METU

Asst. Prof. Dr. Tolga Can ______________________
Computer Engineering Dept., METU

Asst. Prof. Dr. Reza Hassanpour ______________________
Computer Engineering Dept., Çankaya University

Date: ______________________

ABSTRACT

OPENMORE: A CONTENT-BASED MOVIE RECOMMENDATION SYSTEM

KIRMEMİŞ, Öznur
M.Sc., Department of Computer Engineering
Supervisor: Dr. Ayşenur BİRTÜRK

May 2008, 103 pages

The tremendous growth of Web has made information overload problem increasingly serious. Users are often confused by huge amount of information available on the internet and they are faced with the problem of finding the most relevant information that meets their needs. Recommender systems have proven to be an important solution approach to this problem. This thesis will present OPENMORE, a movie recommendation system, which is primarily based on content-based filtering technique. The distinctive point of this study lies in the methodology used to construct and update user and item profiles and the optimizations used to fine-tune the constructed user models. The proposed system arranges movie content data as features of a set of dimension slots, where each feature is assigned a stable feature weight regardless of individual movies. These feature weights and the explicit feedbacks provided by the user are then used to construct the user profile, which is fine-tuned through a set of optimization mechanisms. Users are enabled to view their profile, update them and create multiple contexts where they can provide negative and positive feedback for the movies on the feature level.

Keywords: Web Personalization, Recommender Systems, Content-Based Filtering, Personalization, User Model, Open User Profile, Context Filtering, Trust.

ÖZ

OPENMORE: İÇERİK BAZLI FİLM TAVSİYE SİSTEMİ

KIRMEMİŞ, Öznur
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Danışmanı: Dr. Ayşenur BİRTÜRK

Mayıs 2008, 103 sayfa

Web ortamındaki hızlı büyüme, bilgi yükü problemini artan bir hızla ciddi bir sorun haline getirmektedir. Kullanıcılar genelde internetten erişebildikleri çok büyük miktarlardaki bilgi karşısında şaşırmakta ve kendi ihtiyaçlarını karşılayacak en uygun bilgiyi bulabilme sorunuyla karşı karşıya kalmaktadırlar. Tavsiye sistemlerinin, bu probleme önemli bir çözüm yaklaşımı olduğu kanıtlanmıştır. Bu tez çalışmasında, temeli içerik bazlı filtreleme tekniğine dayanan, OPENMORE film tavsiye sistemi sunulmaktadır. Bu çalışmanın ayırt edici özellikleri, kullanıcı ve öğe profillerini oluşturmada ve değiştirmede kullanılan yöntem ve kullanıcı profillerinin iyileştirilmesinde kullanılan optimizasyon tekniklerinde yatmaktadır. Önerilen sistem, film içerik bilgilerini farklı boyut kümelerinin özellikleri olarak düzenlemekte, her özelliğe filmlerden bağımsız olarak sabit bir özellik ağırlığı atamaktadır. Bu özellik ağırlıkları ve kullanıcı geribildirimleri, kullanıcı profilini oluşturmakta kullanılmaktadır ve oluşturulan profiller kullanılan optimizasyonlarla iyileştirilmektedir. Kullanıcılar profillerini görebilmekte, güncelleyebilmekte ve pozitif ve negatif geribildirimlerini film özellikleri bazında birden fazla durum önceliği olarak düzenleyebilmektedirler.

Anahtar Kelimeler: WWW Kişiselleştirme, Tavsiye Sistemleri, İçerik Bazlı Filtreleme, Kişiselleştirme, Kullanıcı Modeli, Açık Kullanıcı Profili, Durum Filtreleme, Güven.

ACKNOWLEDGEMENTS

First of all, I like to express my sincere thanks to my advisor Dr. Ayşenur Birtürk for her support, understanding and guidance throughout this thesis work.

I am greatly indebted to my family: my father Davut Kırmemiş and my sister Özlem Kırmemiş for their unconditional support and patience.

My special thank is to my husband Ozan Alkan for listening to my complaints and encouraging me to complete this work.

Lastly, thanks to anonymous reviewers of eChallenges 2007 Conference, WEBIST 2007 Conference, ACM RecSys'07 Conference, AAAI'07 Workshop on Recommender Systems, and AAAI'08 Workshop on Intelligent Techniques for Web Personalization & Recommender Systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

RS	Recommendation System
IR	Information Retrieval
IF	Information Filtering
IFW	Item Feature Weight
UPC	User Profile Construction
MAE	Mean Absolute Error
ROC	Receiver Operating Curve

CHAPTER 1

INTRODUCTION

1.1 Background

We are in the information society and the quantity of new information available every day exceeds our limited processing capabilities. When we want to choose something for a particular domain, we face far more choices than we can try. The problem here is the huge size of the search space of all the available alternatives in a particular domain. Therefore, we need some tool/mechanism that will be able to suggest us only the worthwhile information. By using recommendations, we can make the search space of available choices for any particular domain smaller, which will result in saving a lot of time.

Recommender systems automate the facility of obtaining recommendations. Therefore these systems are a valuable solution for the information overload problem.

Considering these; “a recommendation system is a specialized program which attempts to predict items that a user may be interested in, given some information about the user’s profile” [1]. The domain of a recommendation system can be anything on which it is worth taking recommendations on, like movies, music, books, news, web pages, and articles.

The customers of electronic businesses are often overwhelmed by the multitude of the options available to them and depend on recommendation systems for a personalized set of product recommendations. Recommendation systems have been deployed within many e-commerce applications concerning different domains like;

Amazon [3] for books; IMDb [2], MovieLens [5], and MovieFinder [4] for movies; Pandora [6], and Last.fm [7] for music domains.

1.2 Recommendation Systems: An Important Research Area

Recommender systems have become an important and independent research area since the appearance of the first papers in mid - 1990s when researchers started to focus on recommendation problems that explicitly rely on the rating structure [15]. Recommendation systems' roots go back to cognitive science, approximation theory, information retrieval, forecasting theories, and also have links to management science and to consumer choice modeling in marketing. Recommendation systems today are built using a synthesis of ideas from different areas including Artificial Intelligence, Natural Language Processing, Human-Computer Interaction, Sociology, Information Retrieval, and the technology of the WWW. There has been much work done both in the industry and academia for constructing new methodologies to recommender systems over the last decade. The interest in this research area still remains high because it needs solutions to many open-ended problems like trust in recommendation systems, explanation facility, etc. In addition, it needs improvements in better methods to represent user behavior and information about items, more advanced recommendation modeling techniques, adding context data, utilization of multi criteria ratings, etc. Many conferences and workshops were organized and have been planned about recommendation systems, including ACM Recommender Systems [28] and AAAI Workshop on Recommender Systems [27].

1.3 Problem Definition

This thesis focuses on development and evaluation of a movie recommendation system, called OPENMORE. Several features are integrated in OPENMORE, in order to maximize user trust and satisfaction for the recommender. During the design and implementation of the system, we have considered the feedback given from the anonymous reviewers of several workshops and conferences on recommendation systems (namely e-challenges [31], WEBIST [23], ACM [28] and AAAI [27]) to which we've made paper submissions throughout the study.

1.4 Interpretation

In this thesis study, we want to focus on the fact that, standard strategies are not always sufficient to reflect a person's preferences. We have worked on the influence of negative feedback, open and editable user profiles and explanation facility on the accuracy of the produced recommendations and increased users' trust and satisfaction towards the recommender.

1.5 Outline

This thesis consists of the following chapters:

Chapter 2 - Recommendation Systems introduces the idea behind recommenders, and how they appear as a solution to information overload problem. Recommendation problem is introduced formally, and the methods currently used by the recommenders are explored in detail.

Chapter 3 – Recommendation System Examples describes popular recommendation systems from different domains. Current popular movie recommenders are examined in detail, and their pluses and minuses are discussed.

Chapter 4 – OPENMORE: A Content Based Movie Recommender presents our movie recommender, OPENMORE. The architecture of the system is given, the data representation scheme is explained, and the system components are described in detail.

Chapter 5 – Evaluation of the system chapter presents the evaluation scheme designed to test OPENMORE.

Chapter 6 – Conclusions draws the conclusion of this thesis work and recommends possible feature work.

CHAPTER 2

RECOMMENDATION SYSTEMS

The purpose of this chapter is to present the history of recommender systems, define what is a recommender system, how the recommendation problem can be formalized, and explore the research conducted on recommender systems.

2.1 The Recommendation Problem

Recommender systems (RS) are computer-based techniques that can be utilized to efficiently provide personalized services in many e-business domains. They do so by connecting users with information about the content of recommended items and the opinions of other users. Such systems have become powerful tools in domains such as electronic commerce, digital libraries, and knowledge management.

Recommendation methods have been in use informally for years. For instance, even in prehistoric days, when people found a new berry, not everyone tried that berry right away; instead, most of them would wait to see if the others who had tried became sick or not after eating the food. If no one became sick, then this acted as a positive recommendation for eating the berry. If people did become sick, then it served as a negative recommendation for the berry in question [21]. This is a rather simplified but accurate view of recommender systems. Positive and negative recommendations help people to find out new things that they will like or avoid trying some bad alternatives which they will not like. This results in saving a lot of time and avoids facing difficulties regarding information overload, which is a serious problem nowadays.

The roots of RS can be traced back to works in different fields including; cognitive science [9], approximation theory [10], information retrieval [11], forecasting theories [12], and also have links to management science [13] and to consumer choice modeling in marketing [14]. Recommender systems emerged as an independent research area in the mid-1990s when researchers started focusing on recommendation problems that explicitly rely on the ratings structure [16] [17] [18]. Over the last decade, a lot of work has been done both in the industry and academia on developing new approaches to recommender systems and the interest in this area still remains.

Informally, the recommendation problem can be reduced to the problem of estimating ratings for the items that have not been experienced by a user. Intuitively, this estimation is usually based on the ratings given by this user to other items and on some other information related to context. Once the ratings for the yet unrated items can be calculated, the items with the highest estimated ratings can be recommended to the user.

Formally, the recommendation problem can be represented as follows [15]. The formal model of the recommendation system consists of three items, a set C of all users, a set S of all possible items that can be recommended, such as books, movies, or restaurants and a utility function named u. The space S of possible items can be very large, like all the movies or all the books that are going to be recommended. Similarly, the user space can also be very large, even millions in some cases. The key part of the formal model is the utility function, which measures the usefulness of an item s for a user c, that is, $u : C \times S \rightarrow R$. Here, R is a totally ordered set. It can contain nonnegative integers or real numbers within a certain range, which depends on the representation scheme used in the recommendation system.

By using $\langle C, S, u \rangle$ model, the recommendation problem is reduced to choosing an item $s' \in S$, for a user $c \in C$ that maximizes the user's utility. More formally:

$$\forall c \in C, s_c' = \arg\max_{s \in S} u(c, s) \qquad \textbf{(1)}$$

In RSs, the utility of an item is usually represented by a rating, which indicates how a particular user liked a particular item.

Each element of the user space C can be defined by using a profile. This is called a user profile, which contains information about user's likes and dislikes for the recommendation domain. Representation of the user profile depends on the approach and the domain, but, for instance, it can include various user characteristics, like age, gender, income, marital status, etc. In the simplest case, the user profile can contain only a single (unique) element, such as User ID.

Similarly, each element of the item space S can be defined with a set of characteristics, depending on the chosen representation scheme. For example, in a movie recommendation application S may be a collection of movies. Each of the movies in S can be represented not only by its unique identifier, but also by its content information, like its title, genre, director, year of release, leading actors, etc, whichever is thought to have effect on the recommendation process.

Table 1 – User-Item Ratings Matrix for Movie Recommendation

	Matrix	**Memento**	**Hannibal**	**Cube**
Jack	3	5	4	$\otimes$
Jane	$\otimes$	2	$\otimes$	4
Jim	5	2	3	2
Bob	$\otimes$	4	4	3

The main concern of RS is that the utility function u is usually not defined on the whole $C \times S$ space. In RS, utility is typically represented by ratings and is initially defined only on the items previously rated by the users. For example, in a movie recommendation system, users initially rate some subset of movies that they have already seen. An example of a user-item rating matrix for a movie recommendation application is presented in Table 1, where ratings are specified on a scale of 1 to 5.

The "$\otimes$" stands for, the users have not rated the corresponding movies. Therefore, the recommendation engine should be able to estimate or predict the ratings of these nonrated movie/user combinations and then it should issue appropriate recommendations based on these predictions. Therefore, if we consider $C \times S$ space as a matrix of ratings, many of the ratings are not provided initially, so the matrix is highly sparse. Therefore, the job of the recommendation system is to extrapolate u to the whole space $C \times S$ matrix. The way that is followed to issue extrapolations, actually defines the utility function. This extrapolation problem has been approached in a variety of ways, and mainly performed by [18] [19] [20]:

1. Specifying heuristics that will define the utility function empirically validating its performance.

2. Estimating the utility function through optimizing certain performance criterion, such as the mean square error [15].

2.2 Information Retrieval and Filtering

Recommender systems can be understood as a kind of information retrieval systems [22]. In information retrieval systems, user queries are considered to be one-time episodic tasks ordered to the system to complete. However, in recommenders, besides the inherent retrieval activity, there is a continuing task, which is, deriving good estimates of ratings for items the user has not experienced yet.

The rapidly expanding Internet has given the users the ability to choose among a vast variety of information, which results in information overload problem. This information overload problem is the reason why several techniques for information retrieval and information filtering have been developed. Although the goal of both information retrieval and information filtering is to deal with the information overload problem by examining and filtering big amounts of data, there is often a distinction made between them [22].

2.2.1 Information Retrieval

Information retrieval (IR) is a data search technology which includes crawling, processing and indexing of content, and querying for content. Crawling is the act of accessing the database in order to fetch the information. The fetched information can be added, deleted or modified in order to produce the resulting document by the processor. Indexing is a process of examining the processed content, making a searchable data structure, called index, which contains references to the content.

Queries can be seen as requests for information. IR systems let users input a query in the form of keywords which describes the information needed through a query interface. Then the query-processor will use the index to find requested information references based on the keywords and then it will display the references. The whole aim is to analyze the user's request from the query, in order to return the most relevant set of results.

In IR systems, information filtering is done by letting the user specify what information is needed by manually typing keywords describing the requested information. IR is very successful at supporting users who know how to describe exactly what they are looking for. This is the key difference between an IR and RS. In RS, users do not know what they want and what they are willing to get as a result.

2.2.2 Information Filtering

Information filtering (IF) systems focus on filtering information based on a user profile. User profile can be constructed explicitly from the user by letting the user specify and combine interests, or by letting the system implicitly monitor the user's behavior. Filtering within an IF system is done when the user automatically receives the information needed based on his profile. The advantage of IF is its ability to adapt to the user's long-term interest, and bring the information to the user.

The idea behind RS is closely related to IF in the sense that the aim is to construct a system that acts as a personalized decision guide for users, and aiding them in decision making about matters related to user preferences.

2.3 Recommendation Techniques

Recommendation techniques can be classified in a many different ways [24]. However, the interest of this thesis study is not the type of interface or the properties of the user's interaction with the recommender, but rather the focus is on the sources of data on which recommendation is based and the use to which that data is put. Specifically, recommender systems have the following data and tools in order to do their work [25]:

- *background data;* the information that the system has before the recommendation process begins
- *input data;* the information that user must communicate to the system in order to generate a recommendation
- *algorithm;* that combines background and input data to arrive at its suggestions.

Considering data and algorithm used in a recommender, we can classify recommendation systems into five classes as shown in Table 2. Following the notation given in section 2.1, S denotes the set of items over which recommendations might be made, C is the set of users whose preferences are known, c is the user for whom recommendations need to be generated, and s is some item for which we would like to predict c's preference.

Table 2 – Recommendation Techniques

Technique	Background Data	Input Data	Algorithm
Collaborative	Ratings provided for items in S from users in C.	Ratings provided for items in S from c.	Identify similar users of c from C and extrapolate from their ratings of s.
Content-based	Features of items in S.	Ratings provided for items in S from c.	Construct a classifier that fits c's rating behavior and use it on s.
Demographic	Demographic information about C and their ratings of items in S.	Demographic information about c.	Identify demographically similar users of c and extrapolate from their ratings of s.
Utility-based	Features of items in S.	A utility function over items in S that describes c's preferences.	Apply the utility function to the items and determine s's rating.
Knowledge-based	Features of items in S together with knowledge of how these items meet user's needs	A description of c's needs or interests.	Infer a match between s and c's need.

Although five types of recommendation systems are listed in the Table 2, there are even more techniques to solve recommendation problem. However, as Robin Burke shows in her 2002 work on hybrid recommender systems [25], collaborative filtering and content-based systems are the two main types currently relevant to any type of recommendation domain. Therefore, these two approaches would be discussed more, and the attention will be on content-based recommendation systems, since it is the approach used in this thesis study. In addition to this, examples of recommenders that applied these two main techniques can be found in Chapter 3.

2.3.1 Collaborative Recommendation

Collaborative recommender systems (or collaborative filtering systems) try to predict the utility of items based on the items previously rated by other similar users. Here what is obvious is that, the recommendation system does not need to keep track of item features or any demographic information, since finding similarities between user preferences is enough to build a collaborative filtering recommendation system.

In formal terms, collaborative recommenders estimate the utility $u(c,s)$ of an item s for a user c based on the utilities $u(c_j, s)$ assigned to item s by those users $cj \in C$ who are "similar" to user c [15]. For instance, in a movie recommendation system, in order to recommend movies to user c, the collaborative recommender will try to find the "peers" of user c, i.e., other users that have similar tastes in movies, or who rate the same movies similarly. Then, only the movies that are mostly liked by the "peers" of user c would be recommended to him.

Typically, user profiles are kept as a vector of items and their ratings in collaborative recommendation systems, and the vector is augmented continuously as the user interacts with the system over time. Some systems used time-based discounting of ratings to account for drift in user interests [26]. Ratings may be binary (like/dislike) or real-valued indicating degree of preference. Some of the most important systems using this technique are MovieLens [5], Amazon [3], GroupLens/NetPerceptions [17], Ringo/Firefly [18], and Recommender [29].

2.3.1.1 Flow of Action

A basic algorithm for building a collaborative filtering recommendation system works as follows; first of all, the recommendation system maintains a database of many users' ratings of a variety of items that are going to be recommended. For instance, for movie recommendation systems, there exist datasets on the web, which contain ratings of users. Second, for a given user, other similar users, or peers whose ratings strongly correlate with the current user, are tried to be found. This can be performed by algorithms like k-nearest neighbor classifiers. Finally, items rated highly by those peers, but not rated by the current user, are recommended. This is the basic structure of a collaborative filtering algorithm that is used by almost all existing commercial recommendation systems.

For collaborative filtering approach, finding similar users constitutes the main part of the whole recommendation system. Many similarity methods or formulas are used to find similar users like the *Cosine Similarity Measurement* or *Pearson Correlation Coefficient*. According to [30], algorithms for collaborative recommendations can be grouped into two general classes: memory-based (or heuristic-based) and model-based. Memory-based algorithms are essentially heuristics that make rating predictions based on the entire collection of previously rated items by the users. Memory-based collaborative recommenders compare users against each other using correlation or other measures. More formally, the value of the unknown rating $r(c,s)$ for user c and item s is usually computed as an aggregate of the ratings of some other (usually, the N most similar) users for the same item s.

In contrast to memory-based methods, model-based algorithms use the collection of ratings to learn a model, which is then used to make rating predictions. Model-based recommenders derive a model from the historical rating data and use this model to make predictions [30]. Model-based recommenders have used a variety of learning techniques including neural networks, latent semantic indexing, and Bayesian networks.

2.3.2 Content-Based Recommendation

Main idea behind content based recommendation systems is that, they recommend items to users that are similar to the items users rated highly before. Content-based recommendation is actually an outgrowth and continuation of information filtering [22]. In a content-based system, the objects of interest are defined by their important features. For instance, NewsWeeder [33], which is a text recommendation system, uses the words of their texts as features. A content-based recommender learns a profile of the user's interests based on the features present in objects the user has rated before. Therefore, it constructs a user profile from the item profiles. The type of the user profile derived by a content-based recommender depends on the learning method employed. Decision trees, neural nets, and vector-based representations have all been used. As in the collaborative case, content-based user profiles are long-term models and updated as more evidence about user preferences is observed. In formal terms, content-based recommenders estimate the utility $u(c,s)$ of an item s for a user c based on the utilities $u(c,s_i)$ assigned by the user c to items $s_i \in S$ that are "similar" to item s [15].

Content-based recommendation approach has its roots in information retrieval and information filtering research. However, the improvement of the content-based recommendations over traditional information retrieval approaches comes from the use of the user profiles that contain information about users' tastes, preferences and needs. Therefore, while information is gathered for the user, user preferences are also taken into account.

The user profile can be constructed from the users explicitly, for instance, through questionnaires. It can also be obtained implicitly by watching the user's transactional behavior over time. In addition to these alternatives, the system can use some sort of a machine-learning algorithm to induce a profile of the user's preferences from the examples based on a featural description of content.

2.3.2.1 Flow of Action

The basic flow of action of a content-based recommendation system, displayed in Figure 1, can be described as follows; first of all, the system gathers some sort of information from the user, about his/her preferences. For instance, for a movie recommendation system, user's ratings may be gathered for some movies he has already seen. In addition to this information, the system may have information about the content of the movies in its database, through web mining that it will perform, to extract information about movies. Then using these items' profile information, the system will build the user profile. The system has information about all the items that it can recommend to the user, in advance, and it tries to find most related or similar items in its item space or item database with the user's profile to produce its recommendations.

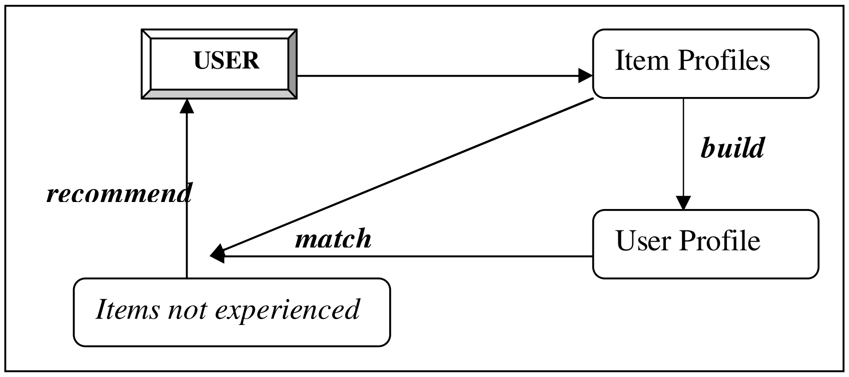


Figure 1 – Content-Based Recommendation

Therefore, there are three building blocks in a content-based recommendation system, namely; item profiles, user profiles, and prediction techniques to make valuable recommendations.

For each item that can be recommended, an item profile will be created. More formally, let $Content(s)$ be the profile of item s [15]. This profile information can include a set of attributes characterizing item s, which will be used to determine the appropriateness of the item s for the recommendation process. For instance, for a movie recommendation system, author, title, actor, director …etc can be possible attributes that characterize items in the item space. Attributes can be determined straightforwardly by just deciding on which attributes to concentrate on, for a recommendation process. Alternatively, attributes of items can be determined through examining some textual data, which can possibly contain a set of some "important" words for that domain. This is usually the case for textual data, like for instance, a web page recommendation system. The "importance" (or "informativeness") of a word in a document can be determined with some weighting measure that can be defined in several different ways. One of the best-known measures for specifying keyword weights in Information Retrieval is the term frequency/inverse document frequency TF-IDF [48] measure.

The profiles of each user, which is shown by $ContentBased\Pr ofile(c)$, are obtained by analyzing the content of the items previously seen and constructed using keyword analysis techniques from information retrieval. For example, $ContentBased\Pr ofile(c)$ can be defined as a vector of weights $(wc1, wc2, ...wck)$, where each weight wci denotes the importance of keyword ki to user c and can be computed from individually rated content vectors using a variety of techniques. This is one way of representing user profiles, and it can be used in a web page recommendation system, for instance. In addition to this, user profiles can be represented by some properties of a person like age, gender, social status... etc which are usually thought as important factors in determining that person's preferences and making accurate recommendations. Other than web page recommendation systems, a vector of features, which will simplify similarity measurement calculations, can be used to represent both user profile and item profile.

For the prediction phase, the important part is the utility function. In content-based systems, the utility function $u(c, s)$ can be defined as follows:

$$u(c,s) = score(ContentBased\Pr ofile(c), Content(s)) \quad \textbf{(2)}$$

Here, as it is observed from the above formula, utility function is defined as a scoring function over item profile and user profile. In content based recommendation systems, both $ContentBased\Pr ofile(c)$ of user c and $Content(s)$ of document s can be represented as TF-IDF vectors of keyword weights. Let's call $ContentBased\Pr ofile(c)$ vector, as vector w_c and $Content(s)$ profile vector, as vector w_s. Moreover, the utility function $u(c, s)$ is usually represented in the information retrieval literature by some scoring heuristic defined in terms of the vectors mentioned above, such as the *Cosine Similarity Measure* [39]. The formula of the utility function can then be represented as follows;

$$u(c,s) = \cos(w_c, w_s) = \frac{w_c . w_s}{\| w_c \|_2 \times \| w_s \|_2} \qquad \textbf{(3)}$$

As a result of these similarity measures, for example, if a user c reads many online articles on the topic of bioinformatics, then content-based recommendation techniques will be able to recommend other bioinformatics articles to the user c. This is the case because these articles will have more bioinformatics-related terms than articles on other topics and, therefore, $ContentBased\Pr ofile(c)$, as defined by vector w_c, will represent such terms with higher weights. Consequently, a recommender system using the cosine or a related similarity measure will assign higher utility $u(c,s)$ to those articles that have high weighted bioinformatics terms in and lower utility to the ones where bioinformatics terms are weighted less.

Besides the traditional heuristics that are based mostly on information retrieval methods, other techniques for content-based recommendation have also been used, such as Bayesian classifiers and various machine learning techniques, including clustering, decision trees, and artificial neural networks. These methods use models learned from the underlying data using statistical learning and machine learning techniques, rather than heuristics.

2.3.3 Demographic Recommendation

Demographic recommendation technique includes categorizing users based on their personal attributes and demographic categories. Grundy [38] was an early example of a demographic recommender which recommended books based on personal information gathered through an interactive dialogue. Data from these dialogues was processed and user responses were matched against a library of manually assembled user stereotypes. Some more recent recommender systems have also taken this approach like the system described in [38], which uses demographic groups from marketing research to suggest a range of products and services. In order to gather the data for user categorization, a short survey is used.

In addition, machine learning techniques can also be applied to arrive at a classifier based on demographic data in order to develop a demographic recommender. The representation of demographic information in a user model can vary greatly. Demographic techniques are similar to the collaborative approach in the sense that, it forms "people-to-people" correlations, however it uses different data. The benefit of this approach is that, it may not require a history of user ratings of the type needed by collaborative and content-based techniques.

2.3.4 Utility-Based Recommendation

Both utility-based and knowledge-based recommendation systems base their recommendations on an evaluation of the match between a user's need and the set of options available and they do not attempt to build long-term generalizations about their users. Utility-based recommenders produce recommendations based on a computation of the utility of each object for the user. Formally, ratings that are going to be suggested to a user c are coded using a utility function which is applied to all the items $s_i \in S$ for defining recommendations.

Of course, the central problem is how to create a utility function for each user. Different techniques can be applied for arriving at a user-specific utility function and applying it to the objects under consideration [34]. For instance, PersonaLogic [25] helps consumers identify which products best meet their needs by guiding them through a large product feature space in the format of a deep interview. The system derives the utility function for the users based on the collection of questionnaires, and then employs constraint satisfaction techniques to locate the best match for the users, with the assistance of the function.

The benefit of utility-based recommendation is that it can factor vendor reliability and product availability into the utility computation, which are non-product attributes, and which will make it possible, for example, to trade off price against delivery schedule for a user who has an immediate need. However, this relies heavily on the information provided by the user.

2.3.5 Knowledge-Based Recommendation

Knowledge-based recommendation systems try to suggest objects based on the inferences about a user's needs and preferences. Actually, all recommendation techniques do the same kind of inference. But what distinguishes knowledge-based approaches from other techniques is the fact that the knowledge-based recommenders have functional knowledge that is, they have knowledge about how a particular item meets a particular user's needs, and can therefore reason about the relationship between a need and a possible recommendation. The user profile can be kept as any knowledge structure supporting this inference. In its simplest case, as in Google [36], user profile can be a query that the user has formulated. In others, it may be a more detailed representation of the user's needs. Several knowledge-based systems employ techniques from case-based reasoning for knowledge-based recommendation.

The knowledge used by a knowledge-based recommender can also take many forms. For instance, Google uses information about the links between web pages to infer popularity and authoritative value [36].

2.3.6 A Comparison of Recommendation Techniques

All recommendation techniques have their own strengths and weaknesses. The possible weaknesses of the recommendation techniques are mentioned below:

New User Problem: This problem refers to the fact that, in order to make accurate recommendations, the system must first learn the user's preferences from the ratings that the user gives. So if a user rates few items, it becomes difficult to produce recommendations. In addition to this, in order to produce accurate recommendations, an item should be rated by a sufficient number of users; otherwise the results may not be meaningful.

New Item (Cold-Start) Problem: This problem refers to the fact that, new items are added regularly to recommender systems and if the recommender relies solely on the other users' preferences to make recommendations, until a new item is rated by a substantial number of users, the recommender would not be able to recommend it. Similarly, a new item that does not have sufficient ratings cannot be easily recommended. This problem makes it necessary for recommender systems to provide other incentives to encourage users to provide ratings.

Overspecialization Problem: This problem refers to the fact that, recommendation systems may recommend items that are too similar to the ones already observed by the user; therefore this will preclude users from discovering novel items. Diversity of the recommendations is a desirable feature for all recommenders. Because of this, it is good to recommend items that are outside the user's profile. Introducing some sort of randomness by, for instance genetic algorithms, can solve this problem. As a result of this, users will be presented with a range of options and not with a homogeneous set of alternatives, which will result in more user satisfaction.

Sparsity of ratings: This problem occurs in recommenders, which construct recommendations based on the ratings of the other users. It becomes a severe problem for such recommenders when few users have rated the same items, because the results may not be accurate in that case. Sparsity problem evolves with time as new users join the system and new items are added to the database.

"Gray Sheep" Problem [40]**:** This problem occurs in approaches that base its recommendations on similar users' tastes. Problem occurs if there are a small number of users who have similar tastes with the user at hand. In that case, results may not be very accurate, if the system cannot locate a sufficient number of peers for the user.

Considering the above problems, limitations of recommendation techniques are summarized in Table 3.

Table 3 – Limitations of Recommendation Techniques

Technique	Disadvantage
Collaborative	• New User Problem • New Item Problem • "Gray Sheep" Problem • Sparsity of ratings • Overspecialization
Content-based	• New User Problem • Overspecialization • Limited Content Analysis
Demographic	• New User Problem • "Gray Sheep" Problem • Overspecialization • Must collect demographic information
Utility-based	• User must input utility function
Knowledge-based	• Knowledge engineering required

Collaborative recommender systems can face new user, new item, "gray sheep" problems and its success depends on the overlap in ratings across users and can have difficulties in producing recommendations when there is a sparsity of ratings. If the dimensionality of the space is reduced, the sparsity problem will not be that significant. However, this depends on the domain of recommendation. For instance, for domains where many items are available, like news filtering, the commonality between the rated items of users and the user at hand will be less. Also, for the user whose tastes are unusual compared to the rest of the population, there will not be any other users who are particularly similar, leading to poor recommendations, overspecialization and “gray sheep” problems.

Pure collaborative techniques are most appropriate for cases where the degree of user interest is relatively high across a small and static set of items. If there is a rapid change in the set of items, old ratings will not be valuable for the new users who have similar items rated with others. If the set of items is large and user interest thinly spread, then the probability of overlap with other users will be small. It can be stated that collaborative recommenders work best for a user who fits into a group with many neighbors of similar taste.

Content-based approach has also the new user and overspecialization problems. Finding the appropriate features in a particular domain can be difficult with content based recommendation. In addition to this, in order to have a reliable content-based recommender, users should rate a sufficient number of items. The degree of descriptiveness of items is highly important for content-based recommendation. Content-based recommenders’ success is limited with the features that are explicitly associated with the items. However, collaborative approaches do not need any profile data of items, since they rely only on other users’ ratings.

Utility-based and knowledge-based recommendations do not have new user, new item and sparsity problems since they base their approach on statistical evidence. Utility-based approach requires a complete utility function to be constructed over all features of objects under consideration. However, they are powerful in the sense that they can include different factors than just item-specific features into recommendation process. Demographic recommendation has new user, “gray sheep” and overspecialization problems. In addition, it needs enough demographic data to produce recommendations.

Knowledge-based recommenders need knowledge acquisition systems, which is actually a drawback of all knowledge-based systems. Despite this drawback, knowledge-based recommendation has some beneficial characteristics like, demanding less data from users and producing recommendations as wide-ranging as its knowledge base allows.

2.3.7 Hybrid Recommender Systems

Several recommendation systems use a hybrid approach by combining two or more recommendation techniques in order to gain better performance with fewer drawbacks of each individual method. Most frequently, content-based and collaborative recommendation approaches are combined. There exist different ways to combine recommendation methods into a hybrid recommendation system. Some examples of these combination approaches can be stated as follows:

1. Collaborative and content-based methods can be implemented separately and their predictions can be combined, by using a weighting procedure.

2. Some content-based characteristics can be incorporated into a collaborative approach.

3. Some collaborative characteristics can be incorporated into a content-based approach.

4. A general unifier model can be constructed that will incorporate both content-based and collaborative characteristics.

Details of these hybrid methods are described in the following subsections;

2.3.7.1 Combining Separate Recommenders

In this approach, content-based and collaborative recommendation systems are implemented separately. Here, there are two ways to combine these separate recommenders. First, the outputs or ratings of these recommenders can be combined to form a final recommendation using either a linear combination of ratings or a voting scheme. Alternatively, one of the individual recommenders can be chosen to be used at any given moment, especially the one that is "better" than others based on some recommendation “quality” metric. For example, the recommender system can be selected that can give the recommendation with the higher level of confidence or whose recommendation is more consistent with the past ratings of the user.

2.3.7.2 Adding Content-Based Characteristics to Collaborative Models

In [47], the “collaboration via content” approach is used, where traditional collaborative techniques together with the content-based profiles are maintained for each user. These content-based profiles, and not the commonly rated items, are then used to calculate the similarity between two users. As a result, this allows overcoming some sparsity-related problems of a purely collaborative approach, since, not many pairs of users will have a significant number of commonly rated items. In addition, using this approach, users can be recommended an item not only when this item is rated highly by users with similar profiles, but also directly, i.e., when this item scores highly against the user’s profile.

2.3.7.3 Adding Collaborative Characteristics to Content-Based Models

In this method, some dimensionality reduction technique can be used on a group of content-based profiles. For example, a collaborative view of a collection of user profiles can be created, where these user profiles will be represented by some sort of vectors. This will result in a performance improvement compared to the pure content-based approach.

2.3.7.4 Developing a Single Unifying Recommendation Model

Many researchers have followed this approach in recent years. For instance, in [51], content-based and collaborative characteristics, like the age or gender of users or the genre of movies are employed in a single rule-based classifier. Therefore, the features of a content-based recommender and collaborative features are combined together in order to produce accurate recommendations.

CHAPTER 3

RECOMMENDATION SYSTEM EXAMPLES

The best way of gaining knowledge of the recommendation problem is to investigate important recommendation systems that have been built so far, getting some ideas and learning how to solve common practical problems that are not usually reported in scientific literature. In this chapter, classifications of recommendation systems are presented. Recommendation systems can be classified according to the different technical features they possess including the kind of approach they conduct, and their working domains. In section 3.1, all the main technical issues, together with their detailed technical features, that a recommender should address are presented. Then based on this knowledge, examples of popular recommenders are examined. Examples are presented based on the taxonomy according to their working domain.

3.1 Technological Features of a Recommender

There are five main issues a recommender system must address, namely, "knowledge acquisition technique", "shared information approach", "profile representation", "knowledge source", and "recommendation technique".

First of all, some sort of knowledge acquisition technique must be employed in order to gather information about the user interests for the recommendation domain in order to construct a user profile. Knowledge acquisition can be done implicitly or explicitly. Implicit knowledge acquisition has little or no impact on the user's normal work activity, since user is not bothered for this process. Implicitly gathering user preferences can be done by monitoring user's behavior or by using some heuristics to infer information. Implicitly acquired knowledge requires some degree of interpretation to understand the user's real goals; this is an inherently error prone process, reducing overall confidence in any resulting user profile. Explicit knowledge acquisition requires the user to interrupt their normal work to provide feedback or conduct some sort of programming of the system. Explicit knowledge is generally more confident, since it is provided by the users themselves and not acquired from indirect inference.

Second, recommender systems may allow information to be shared among users to enhance the overall recommendation performance; however, this shared information must be clearly defined. The type of knowledge that can be shared includes domain knowledge and user feedback. For instance, examples of interesting items can be shared between similar users or previous navigation patterns may also be shared, as they allow new users to receive the benefit from other people's previous mistakes and successes. In addition, domain knowledge can be shared, since it is normally programmed in and hence available to the system from the start.

Third of the main issues that a recommender should address is the representation scheme of the user profile in the system. The standard approach to profile representation is the vector-space model, where profiles are represented as feature vectors. This standard representation technique allows easy application of machine-learning techniques when formulating recommendations. For instance, for content-based recommenders, vectors can be formed from features of items in the domain, while for collaborative filtering, the features could be the keywords commonly used by users in their search queries. Navigation trails can also be used to represent time-variant user behavior. If some initial knowledge engineering for the domain has been conducted, this may provide knowledge about the users available to a profile.

Knowledge source of a recommender keeps all the data about the items in the domain. This knowledge can be kept in the system as an internal database of items, for instance, as crawled web pages, or the system may rely on some sort of external events, such as incoming emails, to provide items for recommendation.

The final requirement of all recommenders is to rely on an appropriate recommendation technique to be employed. Five different approaches are explored in

Chapter 2; however the most common ones are the content-based and the collaborative filtering techniques.

In the next section, examples of recommendation systems shown in Table 4 are presented according to the domain they are working on. During this classification, systems are described considering their technical features, and their approaches that have been conducted for the five main issues discussed above. Attention is on the movie recommenders, and popular movie recommenders are discussed in section 3.3.

Table 4 – Categorization of Recommenders according to Domain

Music	E-commerce	News Filtering	Web	Movie
Pandora Last.fm CDNOW	LIBRA Amazon	GroupLens PHOAKS	Fab	MovieFinder MovieLens Recommendation Explorer Reel.com Netflix MovieCritic

3.2 Classification by Domain

In this section, a summary of recommender systems considering issues discussed in section 3.1, are presented. The recommender systems are listed by application domain, so that similar types can be compared together. When reviewing commercial systems the exact algorithms used are often not published. A more detailed review of commercial E-commerce recommender systems can be found in [24].

3.2.1 Music Domain

3.2.1.1 CDNOW

Technological Features:

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit(rate as liked/not liked) and implicit(purchase history)user feedback
- *Shared Information Approach:* Item feedback and navigation history shared
- *Knowledge Source:* Internal database of items

CDNOW [42] is a commercial music CD shop/recommender system which is now integrated into the wider application of Amazon [3]. It uses collaborative filtering technology. At CDNOW, customers can use the Album Advisor to suggest additional albums in which they may be interested, based on their designation of their three favorite artists. The Album Advisor feature of CDNOW works in three different modes. The first two modes are as follows: Customers locate the information page for a given album or artist. The system then recommends ten other albums related to the album or artist in question. Results are presented as “Customers who bought X also bought set S” or “Customers who bought items by Y also bought set T.” The third mode works as a “gift advisor.” Customers type in the names of up to three artists, and the system returns a list of ten albums that CDNOW considers similar to the artists in question.

Customers buy CD’s through a standard electronic shop web-based interface. As customers shop by navigating through the sites’ web pages, CDNOW presents opportunistic recommendations of items that the user might want. These recommendations are based on the previous navigation patterns and buying habits of the customer.

With “My CDNOW” feature, customers are enabled to set up their own music store, based on the albums and the artists they like. Customers indicate which albums they own, and which artists are their favorites. Purchases from CDNOW are entered automatically into the “own it” list. Although “own it” ratings are initially treated as an indication of positive likes, customers can go back and distinguish between “own it and like it” and “own it but dislike it.” When customers request recommendations, the system predicts six albums the customer might like based on what is already owned. Feedback is provided by customers selecting “own it,” “move to wish list” or “not for me” for any of the albums in this prediction list. The albums recommended change based on the feedback.

3.2.1.2 PANDORA

Technological Features:

- *Recommendation Technique:* Content-based filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Profile Representation:* Vector model
- *Knowledge Source:* Internal database of items

Most of the tools used today for music recommendation are collaborative filtering recommenders. Pandora [6], on the other hand, is a popular and most-noticeable commercial content-based recommender which uses trained musicians to build a model that is used to recommend music based on the content. Unlike collaborative filtering engines, Pandora understands each song in its database. Forty-five analysts, many with music degrees, rank 15,000 songs a month on 400 characteristics to gain a detailed grasp of each. This results in scalability problem since, all the music annotation process is done manually and an estimated cost of analyzing each song is nearly $10. It would become more difficult to keep up this approach as the amount of music generated every day continues to grow. A solution to this may be automating this process in a way which will enable machines to handle analyzing the music instead of humans.

One of the most important features of Pandora is that, whenever the user types in the name of a band or song, he can immediately begin hearing similar tunes that the site’s recommender system has determined he’ll like. By rating songs and artists, user can get refined suggestions, which will allow Pandora to create a truly personalized system behind.

Mainly, Pandora recommender works as follows: The user first provides a song or an artist's name and then the system replies with recommendations. The recommendations are songs that have music qualities similar to the ones that the user has rated. In order to determine similarity between songs, the system uses different features of the items. The descriptions are made using music features stored in the Music Genome Project database [44]. The features are sound qualities only, for instance; minor key tonality, electric guitar riffs, instrumental arrangement etc. Since genre of the music is not used as a feature, the system sometimes provides both successfully surprising recommendations and unexpected songs (i.e. songs that are of completely different musical genre from the one that is liked by the target user). Users are allowed to provide feedback to the system. Each recommended song can be evaluated in the following terms: "*like*", "*not like*". The feedback is used to improve the quality of the recommendations.

3.2.1.3 Last.fm

Technological Features:

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Profile Representation:* Vector model
- *Knowledge Source:* Internal database of items

Last.fm [7] is a commercial, collaborative-based Internet radio and recommender system. The users are able to create lists of their preferred songs and tag songs, albums or artists. Last.fm computes similar users based on their playlists and then provides the target user with a list of items which are not present in his list, but are present in similar users' playlists.

An interesting feature of this system is the formation of user groups between users with something in common (for example, membership of some Internet forum). These groups can have profiles similar to those of individual users, representing the aggregated musical taste of a small community. Members of this group can share recommendations, post messages in a forum and listen to a group radio which is probably created based on the preferred tracks of the members.

While listening to recommended tracks through the web interface, the user can provide a simple feedback to the system (express his love or dislike for a track). In addition to an internet radio music player which requires a download, users can "scrobble" music played from their computers or other devices. "Scrobbling", a term unique to the AudioScrobbler system which powers Last.fm, also requires the download of a widget that records and then uploads what music has been listened to. Any music played on the player is automatically "scrobbled". Users can add friends and the system compares musical tastes based on the "scrobbled" songs. In keeping with the social networking aspect, the system also displays user information about other users who have similar tastes and allows users to contact each other regardless of whether they have established themselves as friends.

Last.fm has data on millions of users listening to thousands of bands. Given its system of 'scrobbling', it has more user data than other systems because it is able to use data generated from sources other than its player. With over 15 million active users acquired without the use of marketing and only word of mouth, Last.fm is a powerful collaborative filtering system.

3.2.2 E Commerce Domain

3.2.2.1 Amazon.com

Technological Features:

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Item feedback, navigation history, item categories shared
- *Knowledge Source:* Internal database of items

Amazon [3] uses recommendation feature to personalize the web site to each of the customer's interest, which will as a result encourages more business actions. Amazon has millions of items in its catalogue and millions of users, so it creates its own item-to-item collaborative filtering recommendation approach in order to construct high quality recommendations in real time under this challenging environment.

Item-to-item collaborative filtering works as follows [41]: a table of similar-items is built through iterative computing procedure and this forms the basis of Amazon's recommendation algorithm. The algorithm matches each of the user's purchased and rated items to similar items and then it forms a recommendation list from these similar items. In item-to-item collaborative filtering, similar-items table is built offline and recommendations are done online from this table which results in quick answering facility.

Amazon includes an information page about the details of the text and purchase information for each book. It has six main features namely, "*Customers who Bought*", "*Eyes*", "*Your Recommendations*", "*Bookstore Gift Ideas*", "*Amazon.com Delivers*", and "*Customer Comments*". "*Customers who Bought*" feature is found on the information page of each book in their site. This feature includes two distinct recommendation lists. First list recommends books that are frequently purchased by customers who had purchased the selected book. Second one recommends authors whose books are frequently purchased by customers who had purchased the books of the selected book's author. "*Eyes*" feature allows customers to get notifications of the newly added books. Customers can specify notification queries based on several features of the books including author, title, and ISBN or publication date. "*Your Recommendations*" feature is actually the one that presents a recommendation list to the users based on their previous ratings. "*Bookstore Gift Ideas*" feature allows customers to get recommendations from editors based on the specific categories like *Teens* or *Entrepreneur*. This feature works online, that is, customers use this feature while they are using Amazon.com website. "*Amazon.com Delivers*" feature is an offline version of "*Bookstore Gift Ideas*" where users get recommendations from editors via email. "*Customer Comments*" feature, on the other hand, enables customers to get text recommendations from the ideas of other users.

3.2.2.2 LIBRA

Technological Features:

- *Recommendation Technique:* Content-based filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Examples of items and item categories shared
- *Profile Representation:* Vector model
- *Knowledge Source:* Internal database of items

LIBRA (Learning Intelligent Book Recommending Agent) [45] is a project on using machine learning to recommend books to readers by learning a profile of their interests. It is a content-based recommender for using information about titles extracted from Amazon. It does information extraction to organize the extracted data into author, title, editorial reviews, customer comments, subject terms, related authors, and related titles fields. LIBRA uses this extracted information to form "bag of words" for a set of slots including author, title, description (reviews and comments), subjects, related titles, and related authors. While using LIBRA, user rates selected titles on a 1 to 10 scale and LIBRA uses a naïve Bayesian text-categorization algorithm to learn a user model of each user's preferences relative to the content of the items from these rated examples where ratings lower than 6 are counted as negative, and greater than 5 as positive. This learned profile is used to rank all other books in the catalog to produce recommendations based on the computed posterior probability that they are positive. In addition to this, users can also provide explicit positive or negative keywords, which are used as priors to bias the role of these features in categorization.

3.2.3 News Filtering Domain

3.2.3.1 GroupLens

Technological Features:

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Item feedback and item categories shared
- *Profile Representation:* Vector model
- *Knowledge Source:* Internal database of items

The GroupLens [49] recommender system is a classic recommender system which recommends Usenet newsgroup articles. It is a distributed system for gathering, disseminating, and using ratings from some users to predict other users' interest in articles. It includes news reading clients for both Macintosh and UNIX computers, as well as "Better Bit Bureaus" servers that gather ratings and make predictions. Both the overall architecture and the particular components have evolved through iterative design and pilot testing to meet the following goals; *openness*, *ease of use*, *compatibility*, *scalability*, and *privacy*.

A split screen interface presents some recommendations together with the ratings provided by other GroupLens users, as users browse their usenet news. The aim of the split screen interface is to blend into the normal usenet interface. This recommendation interface provides some way to identify what is worth reading and what is not along 50,000+ new messages posted each day.

3.2.3.2 PHOAKS

Technological Features:

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Examples of items and item categories shared

- *Profile Representation:* Vector model
- *Knowledge Source:* Internal database of items

PHOAKS (People Helping One Another Know Stuff) [50] is an experimental system that addresses the problem of finding relevant and high quality information on the World-Wide-Web through collaborative filtering approach. It works by automatically recognizing, tallying and redistributing recommendations of web resources mined from Usenet news messages. A hand-crafted set of filter rules is used to classify web resources into categories. Web references are then given a rating based on the number of authors that recommend the reference. The idea behind this approach is that frequently referenced web pages are actually the good ones. Therefore, each news group has a set of ranked recommendations to web pages.

3.2.4 Web Domain

3.2.4.1 Fab

Technological Features:

- *Recommendation Technique:* Hybrid Approach(Combining collaborative and content-based approaches)
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Item feedback, item categories and domain knowledge shared
- *Profile Representation:* Vector model
- *Knowledge Source:* Crawled web pages

Fab [19] is designed to help users sift through the enormous amount of information available in the World Wide Web. This system combines the content-based and collaborative methods of recommendation in a way that exploits the advantages of the two approaches while avoiding their shortcomings. Fab's hybrid structure allows for automatic recognition of emergent issues relevant to various groups of users. It also enables two scaling problems, pertaining to the rising number of users and documents, to be addressed.

Users' profiles are constructed as a collection of keywords contained in those documents that each user rate highly. If the content of the document matches previous documents that were rated highly, or neighbouring users rate a document highly, then these documents are presented for rating to the users. User profiles are dynamic in the sense that whenever a favourable or unfavourable rating is received, the profile of the user is updated to reflect the new rating.

Collection agents, each of which uses a different set of keywords, are sent out over the web to look for documents with specific content. After the documents are retrieved, they are passed to a central server. Here a selection agent matched to each user's profile, scours through the documents looking for interesting material. Relevant documents are then presented to the user for rating. The resulting rating dynamically affects the selection agent's behaviour and changes the user's profile. The rating also affects the collection agent that retrieved the document. Unpopular collection agents are removed and replaced with more successful ones over time.

As it is described above, the most important features of Fab recommender are; it successfully combines the best features of both content-based and collaborative filtering methods and also manages to keep the system dynamically updated to the current users' tastes.

3.3 Movie Recommender Systems Review

3.3.1 MovieFinder.com

3.3.1.1 Technological Features

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Item feedback, and item categories are shared

- *Knowledge Source:* Internal database of items

3.3.1.2 Overview

MovieFinder.com [4] is a movie site maintained by E! Online. There are three features of this recommender that are important to mention, namely, "User's Grade", "Our Grade", and "Top 10" features.

With the "User's Grade" feature, customers can register to the site and give grades to the movies they have seen with a scale from A to F. All users grades for each of the movies are averaged to compute a general "User's Grade" for the movies. On the other hand, editors of E! Online give grades to movies which are presented through "Our Grade" feature. Similar to this feature, the "Top 10" feature of MovieFinder.com allows the customers to get recommendations from the editors in a category that they can select from a list of predefined categories. After they select the category, the top ten movies of that category are displayed, and customers can examine the descriptions of these movies.

3.3.1.3 Discussion

MovieFinder.com is a collaborative filtering movie recommendation system. The most obvious drawback of the system is the degree of personalization that it serves to the customers. First of all, "Top 10" feature of the recommender is totally non-personalized, since; it provides identical recommendation lists to each customer, without considering customer's interests. Customers are just provided with a list that is manually created by an editor, where the process does not use any computation at all. On the other hand, "User's Grade" feature adds some personalization to the system; however this personalization is not persistent. This personalization is simply a short-lived one that only lives for an entire current browsing session. It does not store persistent profiles of users, which is not preferred by the recommendation system users, at all.

It is important to distinguish recommendation systems that try to get and serve personalization to each user based on the information about that user, from systems that deliver recommendations based on editorial judgment or overall popularity where the main idea is “one size fits all”. It’s the first class of recommenders that are exciting and much more satisfying according to user-centric paradigm — especially when these systems can push personally relevant content to each user without requiring exhausting customization.

3.3.2 MovieLens

3.3.2.1 Technological Features

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Item feedback, and item categories are shared
- *Profile Representation:* Vector model
- *Knowledge Source:* Internal database of items

3.3.2.2 Overview

MovieLens [5] is a research site run by GroupLens Research [52] at the University of Minnesota. GroupLens Research is part of the Department of Computer Science and Engineering at the University of Minnesota. The University of Minnesota has been doing work with collaborative recommenders for more than a decade now. Their original project, GroupLens (described in section 3.2.3.1), was a system for using collaborative filters for recommending Usenet messages. MovieLens is an academic tool, and users are frequently asked to participate in the research project. Many of the papers that have resulted from these experiments are posted at the GroupLens site [52]. GroupLens also makes datasets available to other researchers.

MovieLens [5] uses collaborative filtering approach to make its suggestions. Initially, the system asks the user to rate movies. It searches for similar profiles and uses them to generate new suggestions. Each member of the system has a “neighbourhood” of other like-minded users. Ratings from these neighbors are used to create personalized recommendations for the target user. The database of the system is not comprehensive but new movies are added regularly. The recommender frequently made recommendations using half-stars and users are now allowed to rate movies in half-stars.

Features of MovieLens include “Your Movie Wishlist”, “Movie Buddies” and “Shortcuts”. With wishlist feature, users can identify movies that they might like and collect them as a list. This will allow them to keep track of the movies that they would like to see. For instance, users can print their wishlist and take it near them when they go to buy or rent a movie. With “Movie Buddies”, users can form buddy relationships and find a movie to watch with a group of people. This will allow them to have personalized group recommendations. “Shortcuts” to saved searches will allow users to create and save a search query and run it whenever they want. Finally, MovieLens allows users to see how many votes have been collected for a particular movie and whether the prediction made for the user differs from the average vote.

3.3.2.3 Discussion

One shortcoming of the system that most websites using collaborative filtering suffer from is that they do not have any facility to provide explanations of how recommendations are derived. However, explanations for recommenders provide an interface for users to understand better how it is that the recommender works, and why the recommender is suggesting a certain item. This is addressed in [53] which propose explanation facilities for recommender systems in order to increase users’ faith in the suggestions.

In addition to this, in all collaborative filtering systems, there exists a confidence problem regarding the correctness of the ratings entered to the system, which affects the accuracy of the recommender highly. For example, advocates of particular movie genres or particular movie studios may frequently rate movies high on the MovieLens web site right before the movie is released to try and push others to go and see the movie. Additively, new information has a higher potential of being rated more than old information, and old movies have less chance to get recommended by the users.

3.3.3 Recommendation Explorer

3.3.3.1 Technological Features

- *Recommendation Technique:* Collaborative filtering (More specifically similarity matching is done which makes use of a similarity function to find items matching a content-based profile)
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Examples of items and item categories are shared
- *Profile Representation:* Vector model
- *Knowledge Source:* Internal database of items

3.3.3.2 Overview

Recommendation Explorer [55] is an automatic recommender system being developed at the School of Information and Library Science at the University of North Carolina at Chapel Hill. The recommender uses knowledge discovery techniques to improve its representation of item-item relationships, and provides a graphical user interface that enables users to explore recommendations in the context of their information needs. Recommendation Explorer currently uses a database of 12,726 records to recommend films, but is being designed as a generic system that can be readily adapted to a variety of resource collections. One important feature of the recommender is that, all interaction with the system takes place within a single screen. Rather than forcing the user to navigate to a separate page to view metadata for a recommended item, Recommendation Explorer presents all metadata for each item in a pop-up window. This helps users to maintain context when viewing item details.

3.3.3.3 Discussion

It is mentioned in the web site of Recommendation Explorer [56] that, the aim of Recommendation Explorer is to minimize user effort while producing high quality, and personalized recommendations. It uses knowledge discovery techniques to develop a recommendation engine that aims to minimize user input. One important advantage and success of the recommender is the easy usage of the system where users can explore, manipulate, and preview recommended resources. However, there is a potential defect of the system for the items that are weakly connected to other items in the database, because those items may never get recommended to the users where users may perhaps like those items otherwise. In addition to this, Recommendation Explorer lacks explanation facility for the produced suggestions, and it is not a very easy task to produce explanation facility when item-item relationship model is used.

3.3.4 Reel.com

3.3.4.1 Technological Features

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Item feedback, and item categories are shared
- *Knowledge Source:* Internal database of items

3.3.4.2 Overview

Reel.com [57] is a commercial site that recommends movies based on customer reviews. The customers can enter their preferences for different dimensions of movie domain like, genre, viewing format, etc. and a set of recommendations are produced based on the preferences of other customers, like all other collaborative filtering systems.

One feature of Reel.com is “Movie Matches” which provides recommendations on the information page of each move. Reel.com’s “Movie Matches” presents editors’ picks for movies that will appeal to a customer using the movie a customer is currently browsing as an indication of interest. The picks are made by human editors, and come in two categories, namely, “Close Movie Matches” and “Creative Movie Matches”. “Close Movie Matches” provides safer recommendations, and “Creative Movie Matches”, provides more serendipity in the recommendation.

3.3.4.3 Discussion

Reel.com is a simple collaborative filtering movie recommender with no distinguished recommendation power. Actually, it works like a traditional search engine with a detailed movie database. It has the drawbacks of a pure collaborative filtering technique shown in Table 3. In addition, as in the case of MovieFinder.com described in section 3.3.1, system is not personalized.

3.3.5 Netflix

3.3.5.1 Technological Features

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Shared Information Approach:* Item feedback, and item categories are shared
- *Knowledge Source:* Internal database of items

3.3.5.2 Overview

Netflix[59] is a DVD movie rental site which is one of the first commercial applications of movie recommendation, and it has proven to be very popular. Netflix will recommend a movie or movies based on popular rental lists, customers' viewing habits and how customers rate movies. Netflix use collaborative filtering technology. It produces recommendations using a system called “CineMatch”. “CineMatch” is a database that uses information from three different sources in order to determine which movies to recommend to the customers. These sources are listed below:

- The movies which are arranged as groups of common films.

- Customers’ ratings, rented movies and current queue.
- Combined ratings of all Netflix users.

The “CineMatch” database updates itself constantly. The recommender works like all other usual collaborative recommenders; it matches user’s viewing and rating history with people who have similar histories. It uses those similar profiles to predict which movies user is likely to enjoy.

3.3.5.3 Discussion

Netflix may have the largest user rating data, but this data alone is insufficient for creating an effective movie recommendation system. In order to create a successful recommendation system, many other user attributes must be considered. These include, but not limited to, user profiles, the usage and behavior pattern of a Netflix account. One problem of the recommender lies in the usage behavior of customers. Netflix’s user rating data assumes that the users’ ratings came from the same Netflix account and they were made by a single person. For most of the Netflix accounts, this may not be true. Many families only subscribe to a single Netflix account. The members of the family usually share the same account, and rate movies through this single account. It’s inappropriate to consider these combined ratings as the ratings of a single person. For instance, just because one member of the family likes action movies and other likes comedies, one cannot conclude with full confidence that they as a single Netflix account user like both action movies and comedies. In addition to this, another problem associated with this single account usage is that a user who rents all the movies may not be the same person who actually watches them. For example, parents may rent movies according to the behavior of their children.

3.3.6 Movie Critic

3.3.6.1 Technological Features

- *Recommendation Technique:* Collaborative filtering
- *Knowledge Acquisition Technique:* Explicit user feedback
- *Profile Representation:* Vector model

- *Knowledge Source:* Internal database of items

3.3.6.2 Overview

Movie Critic [60] is a collaborative filtering recommender. It has a simple and fast interface, and it was also used on Cinemax's website. Movie Critic generated preconfigured recommendation lists, as well as a prediction for almost any movie (along with a confidence score). It has common features of a collaborative filtering recommender. The discriminative features of Movie Critic include telling the user where his prediction stood in relation to all users' average ratings, that is, it tells whether the user will like a movie better than most people. In addition to this, Movie Critic lets user do a sanity check before it shut down. This is useful when there is a large range of possible scores to give to an item. In this case, what Movie Critic does is generating a list of 13 movies, one for each possible score, and user could then correct them if necessary.

3.3.6.3 Discussion

Since Movie Critic uses collaborative filtering technology, it faces the minuses of this approach. In addition, a different problem occurred at MovieCritic is that, it offers detailed information about recommendation results to users but users had trouble finding it, due to poor navigation design. From the feedback of the Movie Critic users, it can be observed that MovieCritic was rated negatively on layout and navigation. This affected ease of use and subjective usefulness ratings.

3.4 Conclusions

After this review of recommendation systems, a clear picture of what is the "state of the art" with respect to recommendation systems and specifically film recommenders is acquired. In this picture we can identify that most of the systems use collaborative filtering as their recommendation technique. However, even with non-commercial systems, there seems to have been little innovation in the conducted method.

So far, the most common approach used for movie recommendation is the collaborative filtering approach. However such systems require a sufficiently large number of ratings in order to achieve an appropriate recommendation whereas if content-based approach is conducted, user models will be constructed that takes into account individual user preferences for dimensions and possible features of those dimensions without the need for other users' preferences. Collaborative filtering technology is successful in many cases; however it requires a large dataset in order to make appropriate recommendations. In other words, systems using collaborative filtering are difficult to use for recommending newly released films or unknown films before sufficient ratings are assembled.

In addition, explanations of the reasons behind produced suggestions are proven to be a very important facility considering trust in recommendation systems. It also plays an important correction mechanism for handling errors that comes with a recommendation. However, most of the systems lack this facility. Transparency of user profiles and handling context preferences of users are also other desirable features of the recommenders, however, as it can be seen from the descriptions of the example recommenders, most of the systems do not use these features.

When movie domain is considered, the major approaches reported so far use collaborative filtering and content-based filtering techniques. The well-known film recommender, MovieLens [5] is provided by the GroupLens research project. It is based on collaborative filtering, which requires a sufficiently large number of ratings in order to achieve an appropriate recommendation.

In recommender systems, many works related to hybrid recommendation techniques tried to integrate multiple approaches in the prediction generation process [25]. Hybrid recommenders usually combine two or more recommendation techniques, but they are not concerned with the conversion of user models between different techniques. In [51], the authors extract content-based user models from collaborative user models and use both of these models for generating predictions. However, our approach focuses on generation of pure content-based predictions, based solely on the user models that are converted from collaborative user models with the efficient usage of domain knowledge. A content based user model generation is proposed in [61], which converts collaborative user models to content-based user models. However, they kept movie items as a set of features where all features' weights are the same, 0 if not exists, 1 otherwise (in item profiles). In addition, we have used three main fine-tuning mechanisms, which results in higher precision than standard content based user models where movie item profiles are kept in binary.

CHAPTER 4

OPENMORE: A CONTENT BASED MOVIE RECOMMENDER

This chapter presents our movie recommender, OPENMORE. First, an overview of the system is presented. Then the architecture of the system is explained, and the data representation scheme is described. Finally, each system component is discussed in detail, before proceeding to the evaluation of the proposed recommender.

4.1 System Overview

The task of OPENMORE is to recommend movies to a user considering the feedback taken from him. The basic features of OPENMORE include:

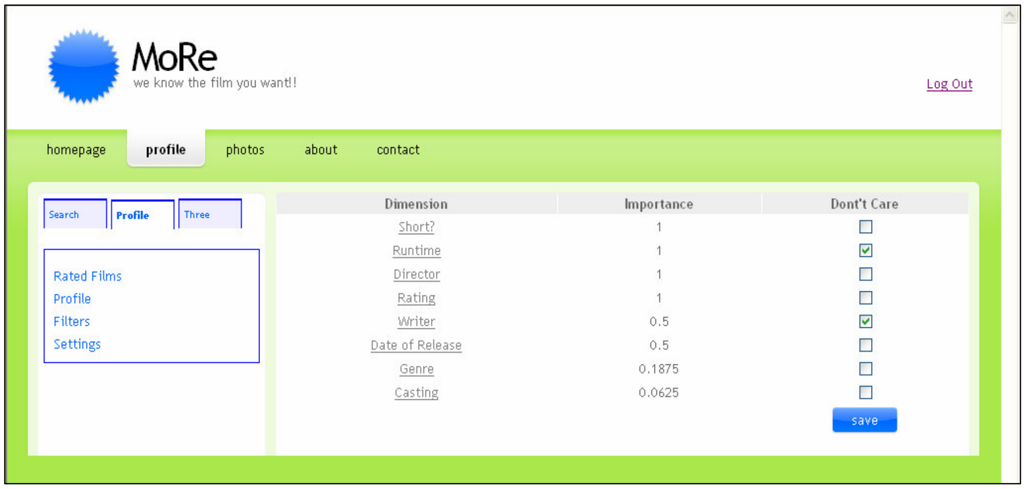
- Automatic creation and update of user profiles
- Open user profiles
- Controlling recommendation process through "don't care"s
- Context filtering
- Explanation

Initially, user profile is formed from the ratings that the user has entered to the system. Every time he submits new ratings, the profile data is updated accordingly. System is designed such that, users can view their profile. They will observe which features are more important than others. In addition to open user profiles, users are enabled to supply feedback to their profile data in terms of "*don't care*"s. If the user thinks a feature or a dimension is not important for him, he will set that feature/dimension as "don't care" in his profile. For instance, consider the scenario, where the user rated five movies with high ratings. Accidentally, actor *A* took role in all of these movies, however, the user does not appreciate that actor, that is, whether the actor *A* took part in these movies actually has no effect on his ratings. He does not want OPENMORE to consider actor *A* while producing recommendations to him. In that case, he will set the "*don't care*" field of that feature to *yes*.

In addition to setting specific features to "*don't care*", users can set a whole dimension to "*don't care*". For instance, if for a user, the runtime of a movie has no importance while he is choosing a movie to watch, he will set "*don't care*" field of runtime dimension to *yes*, which will result in setting all the features of that dimension to "*don't care*".

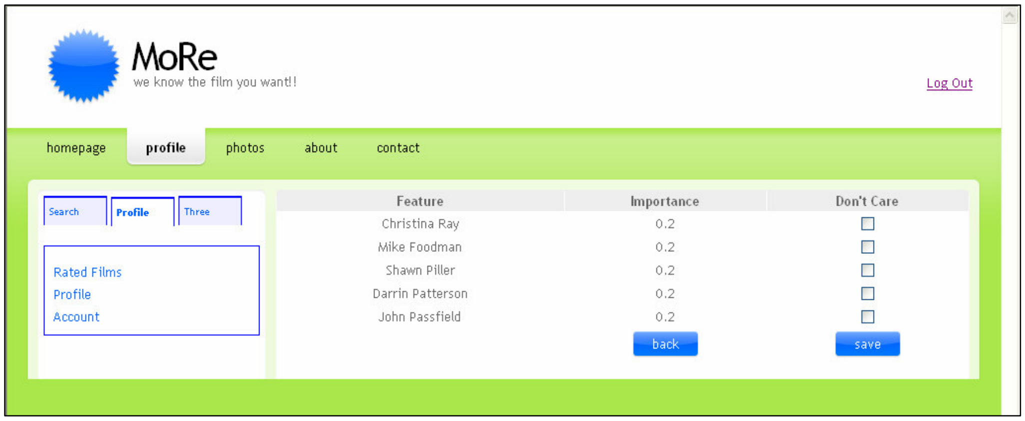
Two screens are designed for displaying user profile and allowing users to update their profile data. One of them is for displaying and updating dimensions, which is shown in Figure 2. In that screen, when the name of the dimension is clicked, all the features of that dimension, which have effect on the user profile, are displayed with their relative scores in the second screen (Figure 3). However, the size of the set of the features for each dimension is important while displaying these features. For instance, as more movies are rated, the number of the actors in the user profile will increase a lot. However, a paging mechanism is implemented, and features are displayed according to their scores, with highest scored feature at top, so users will not get frustrated with long lists.

Users can create context information for different context preferences. OPENMORE will use this data while it builds recommendations for the user. In addition, for every recommended movie, an explanation is provided, which explains the reasons the movie is recommended to him with the provided rating. Explanations are displayed in natural language together with feature scores that have effect in predicted rating. An example of explanation screen for the movie "Analog Days" is shown in Figure 4.



Dimension	Importance	Don't Care
Short?	1	☐
Runtime	1	✓
Director	1	☐
Rating	1	☐
Writer	0.5	✓
Date of Release	0.5	☐
Genre	0.1875	☐
Casting	0.0625	☐

Figure 2 – User Profile Screen for Dimension



Feature	Importance	Don't Care
Christina Ray	0.2	☐
Mike Foodman	0.2	☐
Shawn Piller	0.2	☐
Darrin Patterson	0.2	☐
John Passfield	0.2	☐

Figure 3 – User Profile Screen for features of "Director" dimension

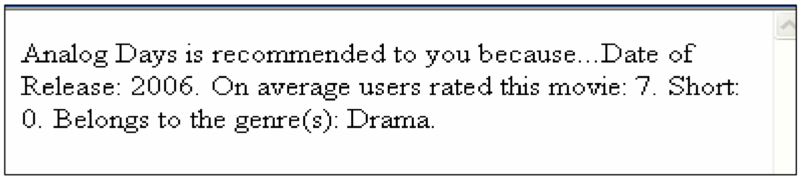

Analog Days is recommended to you because...Date of Release: 2006. On average users rated this movie: 7. Short: 0. Belongs to the genre(s): Drama.

Figure 4 – Explanation Screen

4.2 System Architecture

4.2.1 System Components

System functionality is achieved by five main components.

1. **Web Crawler:** Takes movie data from IMDb and inserts information about movies into database.

2. **Item Profile Constructor and Updater:** Forms and updates item profiles

3. **User Profile Constructor and Updater:** Forms and updates user profile data from the information provided by the user and the movie content data that already exists in the database.

4. **Recommender:** Calculates the appropriate recommendations for the user using the movie content and the user profile information.

5. **Explanator:** Constructs an explanation by giving details of why a movie is recommended with the calculated score.

4.3 Design Issues

4.3.1 General Description of the Proposed Approach

OPENMORE uses content-based recommendation technique for producing movie recommendations. Movie domain can be seen as a set of dimensions where each dimension has a set of features. For instance, one dimension may describe the genre of a movie, and contain features like horror, musical, action, etc.

The proposed approach can be seen as a combination of three distinct parts; *item profile construction and update*, *user profile construction and update*, and *recommendation and explanation generation*. The system's functionality is enhanced by providing users to view their profile and update them in a limited way through providing feedback in terms of “don’t care” values. Both item and user profiles are kept in terms of movie dimensions and features. Therefore, user profiles are presented in terms of the dimension and feature values. In addition, users can create multiple context profiles of their own. Each context profile is a set of positive and negative feedback for the features of different dimensions. Users can direct OPENMORE to use any of these contexts while producing recommendations.

In general, a content-based recommender tries to find best matches between the user profile and the item profiles. In the following subsections, first the user model concept is described in the collaborative filtering and content-based filtering approaches. Data representation schemes and the profile construction mechanisms are described next, which constitutes the base part of the proposed method. Finally, the recommendation algorithm is described, which makes use of these data to assign recommendation scores to movies that confirm with the context preferences.

4.3.2 User Models

Collaborative filtering is one of the most popular recommendation techniques which use cross-user correlations to generate predictions by weighing the opinions of similar users [53]. The input to a standard collaborative filtering system is a matrix of users’ ratings on a set of items, where each row represents ratings of a single user and each column represents ratings on a single item. Thus, collaborative filtering user models are represented as a vector of pair of ratings $i_k : r_k$, which corresponds to a real rating r_k provided by the user on an item i_k.

Content-based filtering [62] builds personalized recommendations by taking the features of items that have been rated by the user, and the set *C* of available items, not yet rated by the user as input. The output of the system will be a subset of *C*, containing the items whose features match the features of the items that are liked by the user. Content-based recommenders generate predictions based on the set of features weighed according to a predefined scale. Therefore, the resulting user models can be represented as a vector of pair of ratings $f_k : w_k$ where f_k denotes one of the domain features and w_k *is* the level of the user's preference regarding this feature.

In OPENMORE, item profiles are also kept in the system as a vector of pairs of $f_i : w_i$, where f_i denotes one of the features in a movie and w_i denotes its corresponding weight. The weights w_i is equal for all the features f_i regardless of individual movies. In the following subsections, the details of the construction of these vectors are described in detail.

4.3.3 Domain Description

We have selected the movie domain for the application of a content-based recommendation system. Movies are appropriate items to be used in content-based recommendation systems, since much content information about them can be easily accessed through IMDb [2] and other resources available on the web.

For each movie item, a set of features is kept in the database of OPENMORE. These features are extracted from IMDb pages by the web crawler module, which accesses movies through keys that are unique for each movie in IMDb. Details of these features and the extracted knowledge are given in section 4.3.5.

4.3.4 Features and Dimensions

Dimension and *feature* concepts are needed to be described in detail for the movie domain. Each movie has a set of features where each feature belongs to a dimension. Dimensions of movie domain that are used in the recommendation process in OPENMORE are; *date of release, rating, color, country, language, runtime, genre, casting, directors, and writers*. Each of these dimensions can have a set of possible values, where for some of them, the values are determined prior to data extraction from the web, and for some of them, values are determined at runtime, and can have new elements whenever new movie information is added to the database. In addition, for some dimensions, a movie can have a set of features whereas for some others, it can have a single value for that dimension. For instance, for the *casting* dimension, a movie can have a set of actors and actresses that have took part in the movie. In addition, the set of all actresses and actors is not stable, since day by day, new actors and actresses appear. However, every movie can have a single *date of release*, and single *runtime* value.

Runtime and *rating* dimensions are the only dimensions whose features are determined prior to data extraction process. This is due to the reason that, these values are rounded up and kept as intervals in the system. For instance *rating* values between *6* and *6.4* are fall in to the feature *category 6*, whereas *rating* values between *7.5* and *8* fall into the feature category of *8*. Similarly, every movie of *runtime* smaller than *30 minutes* fall into the feature *category 30 minutes* for *runtime dimension* and every movie of *runtime* greater than or equal to *180 minutes* fall into the feature *category 180 minutes* for this dimension.

Details of features of dimensions and each dimension's properties are shown in Table 5.

Table 5 – Features and dimensions of movies (*Dimension: name of dimension, Possible Values: possible values of dimension Predetermined: whether values of dimension are stable and determined prior to data extraction Single Value?: if a movie can have a single feature of that dimension, then value of this column is YES, otherwise NO.*)

Dimension	Predetermined	Possible Values	Single Value?
Date of release	NO	1913,1914,…..2008,..etc.	YES
Rating	YES	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	YES
Color	NO	Color (Technicolor), Black and White (Sepiatone), Color (Kodak), …etc.	YES
Country	NO	USA, Canada, UK, …etc.	NO
Language	NO	English, French, Japanese, …etc.	NO
Runtime	YES	30, 60, 90, 120, 150, 180	YES
Genre	NO	Action, Crime, Drama, Horror, Thriller, Comedy, Mystery,…etc.	NO
Casting	NO	Antony Hopkins, Julia Roberts, Jim Carrey, …etc	NO
Director	NO	Andrzej Sekula, Eric Bress, Alexandra Lipsitz, …etc.	NO
Writer	NO	Roman Polanski, Charles MacArthur, Mike Jittlov, …etc.	NO

4.3.5 Data Representation

Information used in the recommendation process consists of content knowledge of the movies collected from the web. The system includes a web crawler module, which connects to IMDb for collecting the movie data. A set of features is extracted from the web pages of the movies and these features are stored in the system. The features extracted from the web for each of the movies are shown in dimension column of Table 5.

A database is created to store all the information needed for the recommendation process. Data extracted from the web for each movie is processed and inserted into the OPENMORE database. More details about the system tables are shown in Table 6.

Table 6 – OPENMORE Database

Table	Columns	Explanation
movie	id, url, title	Keeps movie information, extracted from the web
featurebase	id, name, description, dimension, stableScoreOfFeature	Keeps information for all features of each dimension
moviefeature	id, movie, featureBaseRef	Keeps features of movies
dimension	id, dimension, numberOfFeaturesInDimension	Keeps dimension information
filter	id, userStatePreference, featureBaseRef, negative	Keeps filters defined by users
userstatepreference	id, user, nameOfState	Keeps context preferences of users
ratedmovie	id, movie, rating, userRef	Keeps rated movies of users
user	id, name, password	Keeps user data
userdimension	id, userRef, dimension, negative, dimensionScore	Keeps user profile regarding each dimension
userdimensionfeature	id, featureScoreOfUser, userRef, negative, featureBaseRef, ishigh, negdegree, posdegree, negscore, posscore, totalcount, negcount, poscount	Keeps user profile regarding each feature

Database tables kept in the system can be divided into two categories; movie tables that are related to the movie information and user tables that keep information about the user data and the user's movie interests. *movie, featurebase, moviefeature, dimension* tables fall into the first category whereas the tables *filter, userstatepreference, ratedmovie, user, userdimension*, and *userdimensionfeature* fall into the second category. Details of these tables and how data is stored in them is further explained in the following sections.

4.4 System Components

4.4.1 WebCrawler

As mentioned in the previous paragraph, a web crawler module is used to extract the movie information from the web. The web crawler connects to IMDb for each movie, and extracts the data contained in the html page of the movie into a buffer. Then the data extractor module extracts data from this buffer. Details of this extraction scheme are described in the following paragraphs.

Data extraction process runs in two modes; namely, the *initial mode* and the *successor mode*. The "*Initial mode*" refers to the initial run of the web crawler module before OPENMORE starts serving its users. The "*Successor mode*" of the web crawler module works for every movie that will be added to the OPENMORE database, after the system starts its recommendation process. For the time being, all the movie information for the testing process of OPENMORE is stored in the database. Whenever new movie data is added to the database through *successor mode* of the web crawler module, all effected tables are updated accordingly.

All the dimensions' values and the number of features of each dimension are kept in the *dimension* table. This table is formed prior to the data extraction process. The "*numberOfFeaturesInDimension*" field is updated after the *initial run* of the web crawler module, since the number of the features in dimensions where *PREDETERMINED* column is set to *NO* in Table 5 is not known until all the features are inserted into the database. Whenever new features are added for each dimension, the related dimension's "*numberOfFeaturesInDimension*" field is updated accordingly. The number of the rows of the *dimension* table is static for the time being; however, adding new dimensions does not need any modification for the whole recommendation process.

For each dimension, the possible set of values is kept in *featurebase* table. Web crawler module runs for each movie one by one. It first inserts the basic movie data regarding its title and web address in the database. After that, it inserts every feature of all the dimensions into the database one by one. A feature is inserted into the database if it does not already exist. Therefore not for all the features of a movie, a record in the *featurebase* table is created. But for all features of a movie, a record in the *moviefeature* table is created, together with the id of the related *featurebase* and *movie* record.

The described database tables are the ones that keep movie information extracted from the web by the web crawler module. The other tables keep the user related data. One important thing to mention is that, the *stableScoreOfFeature* field of the *featurebase* table is related to the item profile creation process, and therefore filled and updated by the item profile constructor and updater module.

Both the user and item profiles are kept in terms of the movie dimensions and features. Profiles include the features of the dimensions shown in Table 5. In the following subsections, details of item and user profile construction processes are described.

4.4.2 Item Profile Constructor and Updater

Item profiles are formed prior to the start of the recommendation process. OPENMORE gathers movie data from IMDb [2]. There are two phases in the item profile construction; the *initial phase*, and the *successor phase*. In the initial phase, all the intended movie data are collected before the system starts serving its users. New movie data is added in the successor phase afterwards, in which this new data is inserted and the other effected fields are updated accordingly.

We consider each movie as a combination of dimensions where each dimension has a set of features. We base our item profile construction algorithm on the idea that, the importance of each of the features in the domain should not be the same when we consider the degree of how well one discriminates a movie from the others; therefore we consider the following values for each feature in order to find out their discrimination degree;

1. the ratio of the number of the movies that has feature f_i to the total number of the movies in the database

2. the number of the possible features in the dimension that feature f_i belongs to.

We propose that, *a feature f_i will be more discriminative if fewer movies in the whole movie domain have it*. For instance, think about the following scenario; we have 10 ratings for a user, where in 5 of them actor *a1* acts, and the remaining 5 have country *c2*. Moreover, the user ratings for these movies are all the same. In addition, we have totally 3500 movies in the domain and in 20 of them actor *a1* acted. Furthermore, there are totally 600 films that have the country *c2*. In this scenario, *a1* gives more clues about the user model although both *a1* and *c2* exists in the same number of movies that has same ratings provided by the user. Although *a1* is not a very common feature in the domain, our user has rated movies that *a1* exists. However, nearly 17% of all the movies in the database have the feature *c2* and it is more probable that the user may not have any consciousness or preference about *c2* since it is a very common feature in the whole movie domain.

Our second hypothesis for the item profile construction process is; *a feature will be more discriminative if the size of its dimension set is large*. For instance, actually almost all the movies have values from the genre and the casting dimensions (except possibly animation movies). Therefore every movie has a set of features from each of these dimensions. If the size of the dimension set of a feature f_i is smaller than another feature's dimension size (which is a different dimension); then the probability that f_i exists in any movie is higher. Therefore f_i should be counted as a less descriptive feature. In our current database that we have built for evaluation, there are totally 23 genres and 8409 actors/actresses. Therefore if an actor/actress acted in most of the movies that the user rated highly, this information is more valuable for us than knowing that a genre exists in most of the movies that the user rated highly.

In order to model our hypotheses, we have used Inverse Document Frequency [48] theory. We calculate the item feature weight (IFW) of each feature f_i with the following formula (4):

$$IFW(f_i) = \log\left(\frac{|M|}{|M_{f_i}|}\right) \times \log\left(|D_j|\right) \tag{4}$$

Here, $|M|$ is the size of the set of all the movies in the domain, $|M_{f_i}|$ is the total number of the movies that has feature f_i, $|D_j|$ is the size of the dimension that f_i belongs to. As it can be observed from the formula, the first multiplicand represents our first hypotheses, and the second multiplicand represents our second hypotheses.

In order to create user models, we use the collaborative user models and the domain specific knowledge for the movies that we gather from the IMDb movie database [2]. IMDB provided information in several dimensions; however for the sake of the simplicity, we use 10 feature categories which we believe the most important ones for the user preferences in our work: *genre, casting, language, year, country, rating, director, writer, runtime and color.* After this data is collected, we calculated IFW values of all the features in the domain and store each movie as pairs of $f_i : w_i$ for each of the features f_i in each movie, where w_i is the IFW value of feature f_i.

4.4.2.1 Item Profile Update Mechanism

Whenever a new movie is added to the database in successor mode, IFW values are reevaluated. This is due to the fact that, all the factors that affect the weights of the features in movies are changed when a new movie item is added to the database.

4.4.3 User Profile Constructor and Updater

We form content-based user models by using the ratings provided by the user and the item profiles. We keep user profiles as a vector of weights of the features that exist in the item profiles of the rated movies.

We keep three weights for each feature f_i: neg_weight, pos_weight, and total_weight.

neg_weight corresponds to the weight of f_i that is calculated from the negatively rated movies and pos_weight corresponds to the weight of f_i that is calculated from the positively rated movies. (The reason behind keeping these weights separately is described in the Rating Estimation section 4.4.4.2). The total weight of a feature in the user model is kept in total_weight which is the sum of neg_weight and pos_weight.

For each movie rating provided by the user, we have a list of a movie's features and their corresponding *IFW* values. We form the weights of each f_i in the user model by using the ratings of the movie that has f_i in its profile and its corresponding *IFW* value.

Ratings used in the OPENMORE are in [1-5] scale. The reason behind using this scale is that, the evaluation of the proposed content-based recommendation algorithm is done using the MovieLens dataset [52], which has already conducted [1-5] scale. Therefore, in order to be compatible with the evaluation dataset, [1-5] scale is used in the system.

Ratings greater than 3 are taken as positive, and less than 4 as negative in almost all of the studies reported so far that use the MovieLens dataset [63]. In order to highlight the negativity of the scores, we subtract 3 from each of the ratings. However, to stress the small negativity in rating 3, we do not take rating 3 as 3-3=0; we give -0.1 to highlight this effect and differentiate the features that occurred only in the movies that were rated with rating 3 from the ones, that do not exist in the user model (does not exist in any of the rated movies).

The weights of the features in the user model are created and updated according to the rating of the movies, and the IFW values. In other words, the result of the multiplication of the rating of the movie with the corresponding *IFW* value was added to the positive weights of all the movie genres, actors and directors involved in the movie, and similarly for all the remaining dimensions if the rating for that movie is greater than 3. If the rating is smaller than 4, then the negative weights of the features are inserted/updated in the same manner. In addition to this, the numbers of the occurrences for each feature in negatively and positively rated movies having that feature are stored in the system (which will be used in the Rating Estimation).

For example, consider a user who has the rating 4 for the movie *"The Usual Suspects"*. According to the IMDb, *"Stephan Baldwin" and "Kevin Spacey"* took role in this movie. The user rated this movie positively; therefore the pos_weight and total_weight of these features are increased by the result of the multiplication of 1 (4-3 = 1) with *IFW* values of these features. The weights of other features that exist in this movie are updated accordingly. In addition to this, the number of occurrences in positively rated movies for these features is increased by 1.

The steps of the content-based user model generation process are given below:

USER MODEL GENERATION PROCESS

- User models are kept as a vector of f_i *:(* neg_weight*,* pos_weight*,* total_weight*)* values, where each *fi* exist in one of the rated movies of the user.

- neg_weight of f_i is calculated by the following formula, where $\text{c\_neg}(\text{f}_\text{i})$ is the number of occurrences of f_i in negatively rated movies:

$$\text{neg\_weight}(\text{f}_\text{i}) = \text{IFW}(\text{f}_\text{i}) \times c\_neg(f_i) \qquad \textbf{(5)}$$

- pos_weight of f_i is calculated by the following formula, where $\text{c\_pos}(\text{f}_\text{i})$ is the number of occurrences of f_i in positively rated movies:

$$\text{pos\_weight}(\text{f}_\text{i}) = \text{IFW}(\text{f}_\text{i}) \times c\_pos(f_i) \qquad \textbf{(6)}$$

- total_weight of f_i the sum of $\text{neg\_weight}(\text{f}_\text{i})$ and $\text{pos\_weight}(\text{f}_\text{i})$.

$$\text{total\_weight}(\text{f}_\text{i}) = pos\_weight(f_i) + neg\_weight(f_i) \qquad \textbf{(7)}$$

- For each f_i, we keep the number of occurrences of f_i in positively rated movies and negatively rated movies, that is, $\text{c\_neg}(\text{f}_\text{i})$ and $\text{c\_pos}(\text{f}_\text{i})$ values are kept in the system, which will be used to optimize the constructed user models.

- The calculated feature weights and count of occurrences are kept in the *userdimensionfeature* table's columns, which are shown in Table 7.

Table 7 – Userdimensionfeature Table

Field	Calculated Weight/Count
negscore	$\text{neg\_weight}(f_i)$
posscore	$\text{pos\_weight}(f_i)$
negcount	$\text{c\_neg}(f_i)$
poscount	$\text{c\_pos}(f_i)$
totalcount	$\text{c\_neg}(f_i) + \text{c\_pos}(f_i)$

Actually, user profiles are stored in "*userdimension*", "*userdimensionFeature*", "*filter*" and "*userstatepreference*" tables. "*filter*" and "*userStatePreference*" tables correspond to context filtering feature, which is described in section 4.4.5. "*userdimensionfeature*" table stores calculated feature scores for each user together with the information that tells whether that feature is a "don't care" feature for the user (and some other information about the features which will be described in the next section). For every dimension and for each user, there is a record in "*userdimension*" table, which keeps the average of the total_weight of every feature in the user model and in the corresponding dimension.

4.4.3.1 Optimizing Content-Based User Models

Considering the movie domain, we use some mechanisms in order to fine_tune the constructed content-based user models, which are described in the following subsections.

4.4.3.1.1 "Don't Care" Features

Content-based user models typically store features which the user is indifferent. For instance, consider the following scenario; we have a user who sees only American movies, that is, he never prefers watching movies that were produced by a different country. Almost all of the movies he has rated has feature "*American*" for the country dimension. This shows us that, feature "*American*" has no effect on the preferences of the movies for that user, and the reason behind liking/not liking these movies that have "*American*" feature are the other features except this feature; therefore this feature is identified as a "*don't care*" feature for the target user.

"Don't care" features are identified by the number of occurrences of that feature in the rated movies and the total number of movies that user has been rated. Identifying these features and removing them from the prediction generation step increases the accuracy of the results, because the recommender will base its decisions only on the differentiating features.

To filter these "*don't care*" features, a threshold is defined: "*TH_DONT_CARE*". Features that have the value $\frac{\mathrm{c\_pos}(\mathrm{f_i}) + \mathrm{c\_neg}(\mathrm{f_i})}{total\_rated\_movies}$ greater than "*TH_DONT_CARE*" is set to "*don't care*" and the prediction mechanism is designed and implemented such that only the features that are not "*don't care*" are used in the generation of the recommendation scores. In addition, this optimization is applied to the user models of the users who have rated at least *50* movies.

4.4.3.1.2 "Highly Positive" Features

When movie domain is considered, there can be some features that the user prefers, and likes (has rated with high scores) all the movies that have that feature, regardless of other properties of those movies. For instance, consider a user who is a fan of "*Stephen Spielberg*" and likes all the movies directed by "*Stephen Spielberg*", regardless of other features in those movies. Therefore, if there is a movie directed by "*Stephen Spielberg*" and the user has not seen yet, there is a high probability that the user will like that movie, regardless of the features in the casting, genre or other dimensions. As a result of this, we can identify these features as "highly-positive" features and promote them in the prediction score generation. After such features are identified, the prediction generation step uses the *pos_weight*s of the features that exist in any movie together with a "*highly positive*" feature, in order to be sure to come up with a positive score for such movies. However, if there are two or more movies with a "*highly positive*" feature, the one with the highest cumulated *pos_weights* of its features beats the others.

To identify "*highly positive*" features, two thresholds are defined: "*TH_HIGH_RATIO*", "*TH_HIGH_TOP_COUNT*". First the candidate "*highly positive*" features are found out, which are the ones that have $\frac{c\_pos(f_i)}{(c\_pos(f_i) + c\_neg(f_i))}$ value greater than "*TH_HIGH_RATIO*". This filtering aims to find out the features that have occurred in positively rated movies significantly more than the negatively rated movies.

One more filtering is done in order to select the possible "*highly positive*" features. For every dimension, we set "*TH_HIGH_TOP_COUNT*" threshold in order to find out whether a candidate "*highly positive*" feature, which has been selected after the first filtering, have occurred in a necessary number of rated movies in order to really become a "*highly positive*" feature. This filtering is done for the following scenario; if a user has rated n movies positively and in all those movies actor a_1 played, and there is no other rated movie that has a_1 feature (a_1 occurred only in the positively rated movies), then the $\frac{c\_pos(f_i)}{(c\_pos(f_i) + c\_neg(f_i))}$ value will be 1 for a_1. This will make a_1 a candidate "*highly positive*" feature, because it will for sure pass the first filtering. This is due to the fact that, the highest value of "*TH_HIGH_RATIO*" can be 1. However, if n is so small, like for instance 2, this does not show that a_1 is a "*highly positive*" feature, because n is so small to make a_1 "*highly positive*". Conversely, if n is big enough, this will show that a_1 is really "*highly positive*".

In order to identify these cases, we perform a second filtering. We promote features with "*highly positive*" value, only if they have been included in a sufficient number of the profiles of the rated movies. However, this threshold should differ from dimension to dimension, according to the possible number of occurrences of the features in the movies. For instance, for genres or languages, the number of possible occurrences of the features in these dimensions is high. As a result, the "*TH_HIGH_TOP_COUNT*" threshold is relatively high. For the second, such as the casting, number of possible occurrences of the actors/actresses in movies is very low compared to genres or languages, and the "*TH_HIGH_TOP_COUNT*" threshold is low. One more thing necessary to mention is that, this optimization is applied to the user models of the users who have rated at least *50* movies like "don't care" features.

4.4.3.1.3 "Highly Negative" Features

The idea behind "*highly negative*" features is the same as the idea behind "*highly positive*" features. With "*highly negative*" features, we try to identify which features result in always bad scores and use this valuable knowledge to fine-tune the constructed user models.

For instance, a user may dislike "*horror*" movies and never likes them regardless of the other features that exist in a candidate movie that has genre "*horror*". We identify "*highly negative*" features by using the same thresholds with the "*highly positive*" case. However, for "*highly negative*" case, first elimination takes only the features that have $\frac{c\_neg(f_i)}{(c\_pos(f_i) + c\_neg(f_i))}$ value greater than "*TH_HIGH_RATIO*". Then these features are further filtered according to their number of occurrences in the rated movies, that is, only the features that has $\mathrm{c\_neg}(\mathrm{f}_i)$ value greater than *TH_HIGH_TOP_COUNT* are set to "*highly negative*".

Table 8 – Thresholds Used To Fine-Tune Content-Based User Models

Threshold	Property	Formula
TH_DONT_CARE	"Don't Care" Features	$\frac{c\_pos(f_i) + c\_neg(f_i)}{total\_rated\_movies} > TH\_DONT\_CARE$
TH_HIGH_RATIO	"Highly Positive" Features	$\frac{c\_pos(f_i)}{(c\_pos(f_i) + c\_neg(f_i))} > TH\_HIGH\_RATIO$
TH_HIGH_RATIO	"Highly Negative" Features	$\frac{c\_neg(f_i)}{(c\_pos(f_i) + c\_neg(f_i))} > TH\_HIGH\_RATIO$
TH_HIGH_TOP_COUNT	"Highly Positive" Features	$c\_pos(f_i) > TH\_HIGH\_TOP\_COUNT$
TH_HIGH_TOP_COUNT	"Highly Negative" Features	$c\_neg(f_i) > TH\_HIGH\_TOP\_COUNT$

All the thresholds used for fine-tuning content-based models are summarized in Table 8. During the evaluation phase, the optimal values of these thresholds are determined, as it will be described in the evaluation section.

We only permit the features of *genre*, *casting* and *director* dimensions to become a "*highly*" feature; since we believe these are the most discriminating dimensions when the movie domain is considered. After identification of "*highly*" features is completed, we have processed the features that occur with "*highly*" features. We assume that, if a feature *fi* exists in negatively rated movies, and all those negatively rated movies have a "*highly negative*" feature, than this feature has no effect on those negative ratings, since we propose that, the reason behind that negative ratings are "highly negative" features. Therefore, we reset the negative weight of feature *fi.* To do this, we set the $neg\_weight(f_i)$ of *fi* to 0, and we increase the total_weight accordingly (total_weight is set to $pos\_weight(f_i)$ and $c\_neg(f_i)$ is set to 0). The idea behind this optimization is; we assume that, user did not like those negatively rated movies because of the "*highly negative*" features, not because of feature *fi*. Same process is done for features that exist in positively rated movies, and all those positively rated movies have a "*highly positive*" feature. This time, $\text{pos\_weight}(f_i)$ and $\text{c\_pos}(f_i)$ is set to 0, and total_weight is set to $\text{neg\_weight}(f_i)$ of *fi*. If this optimization results in features that have *total_weight* equal to 0, these features are removed from the user-model.

During optimization process, we first identify the "*don't care*" features. Then the "*highly positive*" and the "*highly negative*" features are identified from the ones that have not already set to "*don't care*". Therefore, a feature can never be set to both "*don't care*" and "*highly positive*" or "*highly negative*".

4.4.3.2 User Profile Update Mechanism

The update of user profiles is accomplished by two different processes; the *automated user profile update process*, which works automatically whenever user provides ratings for the unrated movies and the *explicit feedback processor* which updates user profiles from the feedback provided by the user by the help of the *open user profile* facility.

Whenever user rates new movies, pos_weight_fi, neg_weight_fi, and total_weight_f_i values of all the features of the newly rated movies are added / updated according to the ideas presented in section 4.4.3.

User profiles are presented in the form of importance values for the features and dimensions, as shown in Figures 2 and 3. For every dimension, a score is calculated by averaging the total_weight_f_i scores of the features of each dimension. User can also observe each score that was assigned to the features.

Open user profile facility increases user's trust to the recommender and it provides a mechanism for handling possible errors. From the user profile views, users can set the importance of a whole dimension, or specific features of each dimension to “don’t care”. If a dimension is set to “don’t care”, this value is propagated to every feature of that dimension. As a result, recommender does not take into account these “don’t care” features in its rating estimation process. For instance, for a user, the runtime of a movie may have no effect on his movie preferences; therefore he may set that dimension to “don’t care” as a whole. For another user, whether the director of a movie is someone other than “*Steven Spielberg”* may not be important, that is, he may like “*Steven Spielberg”* however; he may not know other directors well, so he may not want the recommender to make any calculations considering these other directors. In that case, he may set all those other directors’ values to “don’t care”(an easy way is provided for the user for this process, first he checks “don’t care” field to “yes” for director dimension and unchecks *“Steven Spielberg”s* “don’t care” value).

4.4.4 Recommender

4.4.4.1 Content Based Recommendation

OPENMORE uses content-based recommendation approach. In general, a content-based recommender tries to find best matches between user profile and item profiles. Details of recommendation construction process are described in section 4.4.4.2.

4.4.4.2 Rating Estimation

The prediction generation process takes a content-based user model *um* and a set of candidate movies *MC*. It generates a list of recommended movies that are sorted according to their calculated recommendation scores.

The prediction generation process produces scores for the three distinct subsets of the *MC* where each one is ordered in itself according to the produced score:

- *List_Pos:* For the candidate movies in *MC* which have a feature that exists in *um* and is “*highly positive*”.

- *List_N:* For the candidate movies in *MC* which have no feature that exists in *um* and is “*highly positive*” or “*highly negative*”.

- *List_Neg:* For the candidate movies in *MC* which have a feature that exists in *um* and is "*highly negative*".

The resulting recommendation list has the candidates in *List_Pos* at the top. Then the candidates in *List_N*, and finally, the candidates in *List_Neg* exist in the recommendation list. The pseudo-codes of the algorithms for the generation of these lists are given below in Figures 5, 6, 7.

```
Gen_List_Pos(User-Model um, candidate_movie_set MC )
list_pos={}
rec_score=0
high_feature_exists=false
for each mc∈ MC
{
  fSet(mc)=get feature set of mc
  for each f ∈ fSet(mc)
  {
    if ( f ∈ fSet(um) and is_dont_care(f,um)=false)
    {
       if(is_highly-pos(f,um)=true)
              high_feature_exists=true
       rec_score=rec_score+pos_weight(f)
    }
  }
  if (high_feature_exists=true)
       add mc to list_pos together with rec_score
  rec_score=0
}
```

Figure 5 – List-Pos Generation Algorithm

```
Gen_List_Neg(User-Model um, candidate_movie_set MC )
list_neg={}
rec_score=0
high_feature_exists=false
for each mc∈ MC
{
  fSet(mc)=get feature set of mc
  for each f ∈ fSet(mc)
  {
     if ( f ∈ fSet(um) and is_dont_care(f,um)=false)
      {
         if(is_highly-neg(f,um)=true)
                 high_feature_exists=true
         rec_score=rec_score+neg_weight(f)
      }
  }
  if (high_feature_exists=true)
       add mc to list_neg together with rec_score
  rec_score=0
}
```

Figure 6 – List-Neg Generation Algorithm

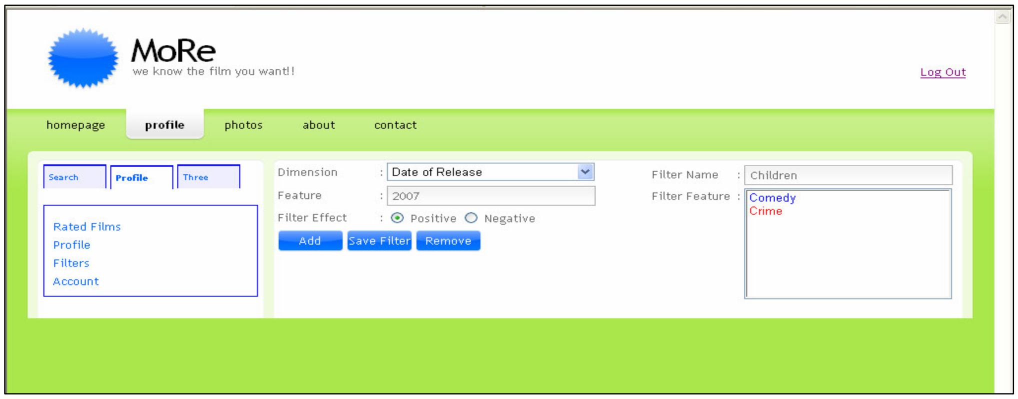
```
Gen_List_N(User-Model um, candidate_movie_set MC )
list_N={}
rec_score=0
high_feature_exists=false
for each mc∈ MC
{
  fSet(mc)=get feature set of mc
  for each f ∈ fSet(mc)
  {
      if ( f ∈ fSet(um) and
         is_dont_care(f,um)=false and
         is_highly-neg(f,um)=false and
         is_highly-pos(f,um)=false)
      {
              rec_score=rec_score+total_weight(f)
      }
      else
      {
              high_feature_exists=true
              exit for LOOP
      }
  }
  if (high_feature_exists=false)
      add mc to list_N together with rec_score
   rec_score=0
}
```

Figure 7 – List-N Generation Algorithm

4.4.5 *Context Filtering*

The idea behind context filtering is to increase user satisfaction and propose more suitable recommendations considering user mood. Suppose a user establishes a rating profile that contains possible ratings for the movies *Shakespeare* and *Die Hard*. Whether he prefers *Shakespeare* or *Die Hard* may heavily depend on his context, that is, for instance, whether or not he is taking his girl friend to the movie. In order to handle this, system allows users to create multiple “contexts” to handle different moods. A context is defined as a set of filters, which includes information for eliminating or emphasing the features of each dimension. The Context filterer works on the feature level and it takes positive and negative feedback for the features. For instance, user can create a context for the cases when he will want to watch a movie with his children. He may create a context named “Family” and wants the comedy films and the movies released at year 2007 to be recommended to him, whereas no crime movies are advised to him. This case is illustrated in Figure 8, where there exist 3 features in context “Family”, two of which are positive, namely, feature “Comedy“ for genre dimension and feature “2007” for “year” dimension; and one of which is negative, namely, feature “Crime” for “genre” dimension.

While user is taking recommendations, he can choose a context, and this will force the recommender to produce the recommendations that satisfy this context’s specifications. User may also create a default context for himself if he is sure about some features, which he wants to be counted as positive or negative all the time. If no context is chosen or set as default, OPENMORE proceeds as if there is no context preference.



Dimension : Date of Release
Feature : 2007
Filter Effect : [✓ Positive] Negative
Filter Name : Children
Filter Feature : Comedy
Crime

Figure 8 – Context Screen

4.4.6 *Explanation Module*

Explanations provide a mechanism for handling errors in a recommendation. For every movie recommended to the user, he is provided with an explanation considering why that movie is predicted with that rating to him, and which dimensions and features are effective on the results. Explanations are formed for each movie, from the user models. This facility enables users to observe the reasons behind predictions, and update their profile data and context filters accordingly to get better recommendations.

4.5 Implementation Details

4.5.1 Implementation Environment

The implementation environment includes a server running on the Windows XP operating system. Apache-Tomcat Application Server v6.0.10 [35] is used together with the open source database MySQL [46]. They provide a secure and reliable platform for our system, and are also easy to set up locally in order to preview and test code as it is being developed.

MySQL is the most popular open source database which is frequently used together with Apache web server. The reasons of wide use of MySQL are its performance, reliability and easy administration. Furthermore, it is available for all major operating systems.

OPENMORE was implemented in JDeveloper 10 [32] a free development environment by Oracle. JDeveloper was not released as open source but is provided at no charge and is being continuously developed.

Java with its Servlet and JSP technologies is used in the implementation. The key factors supporting this decision are Java's platform independence, multi-threading and higher security facilities. In addition, it has better performance than interpreted scripting languages, and more suitable for the development of large projects. Additively, the Apache Tomcat server and related technologies give Java developers a rich set of tools to quickly build more sophisticated Web applications. Tomcat version 5 supports the latest JSP and Servlet specifications, JSP 2.0, and Servlets 2.4.

All communication between the client and the server application is done using HTTP where the client application initiates a request by establishing a TCP connection to a particular port on the server host. The server application is listening on that port and waits for the client application to send a request. Upon receiving a request, the server application accepts the connection and takes action according to the request before sending a response to the client application. HTTP is chosen as application layer protocol because it suits our needs and works well with today's Internet infrastructure.

CHAPTER 5

EVALUATION OF THE SYSTEM

This chapter presents the experimental methodology used to test the proposed approach. First, the properties of the dataset and the metrics used during the experiments are described. Then, the results of the experiments are reported and discussed.

5.1 Data Set

Our experiments use the MovieLens million rating dataset [5], which is a collaborative filtering dataset storing ratings by 6040 users of 3952 movies. Ratings are provided on a scale of 1 to 5, 5 being excellent and 1 being terrible. Based on the rating guidelines presented to the users of MovieLens, we identified ratings of 4 and 5 to signify "good" movies [63]. These are the movies that would make good recommendations.

As it is mentioned in the *readme* file of the dataset, some movieIDs do not correspond to a movie due to some accidental duplicate entries or inconsistencies. As we are doing content-based filtering, we processed all the movies used in the dataset (in order to collect information from IMDb database) and these inconsistent movie entries and their corresponding rating data are removed from the OPENMORE database. Therefore, our resulting database has 3881 movie items, 6040 users and 1000187 ratings.

We applied 5-fold cross-validation to the dataset of ratings by splitting the set of ratings using 20%-80% ratio and doing such split 5 times for each user. For each of these 5 splits, we designated the 20% part of the initial dataset as *evaluation dataset* and the remaining 80% of the dataset was designated as *training dataset*. We repeat our experiments with every training sets and test sets for each of the users selected for evaluation and after this phase completed, we average their results. We generated each training set and testing sets by taking a random sample [51] of the data as follows:

- For every user, we separate and group his movie/rating pairs into intervals defined by the ratings. Therefore, for every user, there are at most 5 groups of ratings, one group for rating value 1, one group for rating value 2, and so on.

- From each group of ratings for each user, we take the same number of movie/rating pairs to each training and testing datasets (when possible, since sometimes number of items is not divided evenly by 5), and we control to have 20% distribution of movie/rating pairs each time from each rating group to each training and testing datasets.

As a result of this procedure, each of the testing sets will be more representative of the distribution of ratings for the entire dataset than it would have been if we have used a simple random sampling.

In order to give an idea about the rating dataset, we provided Table 1 which shows the distribution of the number of rated movies among the users in the dataset. As it can be observed, *2909* users have rating data for more than *100* movies.

Table 9 – Distribution of the Ratings among the Users in the Data set

Number of Rated Movies	Number of Users
0 to 25	492
26 to 50	1301
51 to 75	785
76 to 100	553
101 to 125	480
126 to 150	345
151 to 175	306
176 to 200	200
201 to 225	207
226 to 250	148
251 to 300	268
301 to 400	354
401 to 500	205
over 500	396

5.2 Evaluation Metrics

There are several performance metrics that are traditionally used to evaluate performance of recommender systems, such as mean absolute error (MAE), mean squared error (MSE), correlation between predictions and actual ratings, precision, recall, F-measure, and the Receiver Operating Characteristic (ROC) [43]. Moreover, these metrics are classified into *statistical accuracy* and *decision-support accuracy metrics.* Statistical accuracy metrics compare the predicted ratings against the actual user ratings on the test data. For instance, the MAE measure [37] is a representative example of a statistical accuracy measure. On the other hand, the decision-support accuracy metrics measure how well a recommender system can predict which of the unknown items will be highly rated. The F-measure is a representative example of the decision-support accuracy metric [8] which will be described in the following paragraphs. Moreover, although both types of measures are important, it has been argued in the literature [8] that the decision-support metrics are better suited for recommender systems because they focus on recommending high-quality items, which is the primary target of recommender systems. Therefore, in the evaluation phase, we have used the following decision-support metrics: *accuracy*, *precision*, *recall*, and *F-measure*. In order to describe these metrics, we will use Table 10.

Accuracy is defined as the fraction of correct recommendations to the total possible recommendations (Formula 8). So it is the ratio of recommendations that are correctly classified as possible or negative when the total number of possible recommendations is considered.

$$Accuracy = \frac{(N_{rs} + N_{in})}{N} \qquad \textbf{(8)}$$

Precision and *recall* are the most popular metrics for evaluating information retrieval systems. For the evaluation of recommender systems, they have been used by Billsus and Pazzani [54], Basu et al. [51], and Sarwar et al. [39].

Precision is defined as the ratio of relevant items selected correctly to the number of items selected, or in other words, the fraction of positive examples that were correctly classified as positive, as shown in Equation 9. Precision represents the probability that a selected item is really relevant for the user.

$$\Pr ecision = \frac{N_{rs}}{N_s} \qquad \textbf{(9)}$$

Recall, shown in Equation 10, is defined as the ratio of relevant items selected to the total number of relevant items available. Therefore, *recall* represents the probability that a relevant item will be selected. It is calculated by taking the ratio of the items that are correctly classified as positive to the total number of the positively rated items.

$$\mathrm{Re}\,call = \frac{N_{rs}}{N_r} \tag{10}$$

Several approaches have been taken to combine precision and recall into a single metric. One approach is the *F-measure* which combines precision and recall into a single number by the formula presented in equation 11.

$$F - measure = \frac{2 \times \mathrm{Pr}\,ecision \times \mathrm{Re}\,call}{\mathrm{Pr}\,ecision + \mathrm{Re}\,call} \tag{11}$$

Table 10 – Table Showing the Categorization of Items in the Data Set with Respect to a Given Information Need

	Predicted as Positive	**Predicted as Not Positive**	**Total**
Actual Positive	N_{rs}	N_{rn}	N_r
Actual Negative	N_{is}	N_{in}	N_i
Total	N_s	N_n	N

The movie data set and the produced recommendation results must be separated into two classes; "relevant" or "not relevant" in order to make metric calculations. That is, we need to transform all the ratings in the dataset into a binary scale in order to compute *precision*, *recall* and *accuracy* values. The MovieLens dataset has a rating scale of 1–5 and is commonly transformed into a binary scale by converting every rating of 4 or 5 to "relevant" and all ratings of 1–3 to "not relevant" [58]. In addition, for the items whose recommendation scores are positive are counted as positive and others as negative in the produced recommendation lists.

For each user, *precision*, *recall*, *F-measure*, and *accuracy* values are computed for every testing set (there are 5 testing sets for every user) and these values are averaged for each of the 5 testing sets for each user to come up with one *precision*, *recall*, *F-measure* and *accuracy* value for every user. We averaged the results of every user to find out the general *precision*, *recall*, *F-measure* and *accuracy* values of the proposed system.

5.3 Evaluation Details

Experiments are accomplished using the MovieLens collaborative filtering dataset. The collaborative user models in the dataset are transformed to content-based user models and two groups of experiments were performed. The conducted experiments are shown in Table 11. In the first group, three sets of experiments are conducted and in the second experiment group, two experiments were done. For the second set of the first group, three sub sets of experiments are performed. For each of the tests, the reasons behind the experiments are displayed in the last column of Table 11. In the following sections, we refer to experiments with their experiment numbers given in the first column of Table 11.

5.3.1 First Group of Experiments

The first group of experiments is conducted to fine-tune the prediction generation mechanism by selecting the most appropriate values for the *TH_DONT_CARE*, *TH_HIGH_TOP_COUNT* and *TH_HIGH_RATIO* thresholds. To accomplish this, we have performed five experiments for finding out the best value of each of these three thresholds; Exp1.1, Exp1.2.1, Exp1.2.2, Exp1.2.3 and Exp1.3.

Exp1.1 is conducted to find the best value for *TH_DONT_CARE* threshold. We tested the pure content-based recommender with only “don’t care” features, that is, without using any other optimization (without “highly positive” and “highly negative" features). Exp1.2.1, Exp1.2.2, and Exp1.2.3 aim to find out the best *TH_HIGH_TOP_COUNT* values for *genre*, *casting* and *director* dimensions. In these experiments, we set *TH_HIGH_RATIO* to *0.8.* We conducted totally three subsets of experiments in this second set in order find out the best *TH_HIGH_TOP_COUNT* values.

When we completed the second set of experiments, we set the best *TH_HIGH_TOP_COUNT* values for each of the dimensions that we have found out in the second set of experiments and conducted the third experiment, Exp1.3, to find out the best value for *TH_HIGH_RATIO.* "don't care" features were excluded from both the second and third set of experiments of the first group; therefore these two tests were conducted with pure content-based recommender with only "highly positive" and "highly negative" features.

The tests in the first group were completed using the rating data of the randomly selected *200* users. As the selections have been done randomly, the statistical properties of the original dataset have been preserved. Results of the predictions were evaluated and the best values for the thresholds were chosen using the *precision* metric mentioned in section 5.2. Experimental results of the first group of experiments are given in the following subsection.

Table 11 – Table Showing the Conducted Experiments

Experiment	Exp. Group	Exp. Set	Exp. Sub Set	Testing What?
Exp1.1	1	1	-	*TH_DONT_CARE*
Exp1.2.1	1	2	1	*TH_HIGH_TOP_COUNT for casting dimension*
Exp1.2.2	1	2	2	*TH_HIGH_TOP_COUNT for genre dimension*
Exp1.2.3	1	2	3	*TH_HIGH_TOP_COUNT for director dimension*
Exp1.3	1	3	-	*TH_HIGH_RATIO*
Exp2.1	2	1	-	*Pure Content-Based Rec.*
Exp2.2	2	2	-	*Pure Content-Based Rec. with Optimizations*

5.3.1.1 Experimental Results

To find the most appropriate value of *TH_DONT_CARE*, experiments were conducted on the pure content-based predictor with only the “don’t care” features’ optimization. During the experiments, the values of *TH_DONT_CARE* were increased from 0.50 to 0.90 by 0.5. Results of this experiment are illustrated in Figure 9. The horizontal axis shows the values of *TH_DONT_CARE* threshold and the vertical the precision value. As it can be seen, precision values initially increase with the *TH_DONT_CARE*, and then decrease. This is explained by the influence of “don’t care” features. If the *TH_DONT_CARE* threshold is low, and so many features are assigned to “don’t care”, this results in eliminating discriminative features from the prediction score generation. In other words, features are assigned to “don’t care” incorrectly which are in fact not “don’t care”. Therefore, precision is low with low values of TH_DONT_CARE, due to filtering out more features through setting them to “don’t care”, which are possibly the discriminative features for the users. When the *TH_DONT_CARE* threshold is too high, precision is low due to not assigning neutral features to "don't care" which are in fact "don't care" features for the user. Thus *TH_DONT_CARE=0.60* is chosen as an optimal value where the precision is highest (over 0.65).

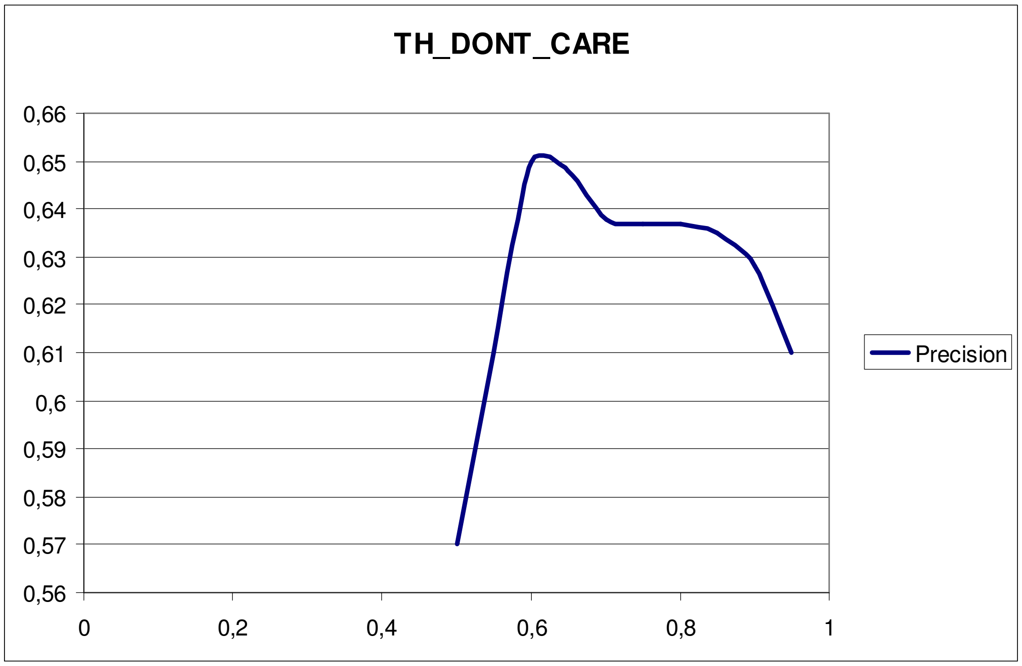


Figure 9 – Precision vs. *TH_DONT_CARE* Threshold

After determining the value of *TH_DONT_CARE* threshold in Exp1.1, second set of experiments started to find out the optimal values for *TH_HIGH_TOP_COUNT* for genre, casting and director dimensions. The dimensions in the testing procedure for *TH_HIGH_TOP_COUNT* were separated, and the same methodology was used to determine the optimal *TH_HIGH_TOP_COUNT* value for each of these three dimensions. We set *TH_HIGH_RATIO* to *0.80* for all the tests in this second set of experiments (The optimal value for *TH_HIGH_RATIO* is found at the third set). All the tests in this second group were completed distinctively, in order to find out the optimal *TH_HIGH_TOP_COUNT* values without the effect of any other optimization. Therefore, we have completed tests for different dimensions distinctively, that is, we first made tests to determine optimal

TH_HIGH_TOP_COUNT for casting category, and we only permit the features of this category to become "highly positive" or "highly negative", that is, we only used this "highly" values' optimization for casting dimension. Similarly, in Exp1.2.2, we make tests to determine optimal *TH_HIGH_TOP_COUNT* for genre, and only used this optimization for content-based user model generation. So for each genre, casting and director dimension, a separate experiment was conducted and the precision values were computed as a function of the *TH_HIGH_TOP_COUNT* threshold.

Considering *TH_HIGH_TOP_CO*UNT, we observed two different situations corresponding to three types of categories. For the first one, such as casting or directors, the number of possible features is very high, and the *TH_HIGH_TOP_COUNT* threshold is low. For the second, such as genre, the number of possible features is low. As a result, the *TH_HIGH_TOP_COUNT* threshold is relatively high.

The experiment was conducted with the same *200* users, used in the previous experiment. Figures 10, 11, 12 illustrate the results of the experiments for *TH_HIGH_TOP_COUNT* for casting, genre and director categories respectively. In all these experiments, the horizontal axis shows the *TH_HIGH_TOP_COUNT* values and the vertical – the precision values. The results show that for the all the three dimensions, precision increases with *TH_HIGH_TOP_COUNT* and after a maximum precision value, it decreases and stabilizes. Thus, setting too many features of these dimensions through low *TH_HIGH_TOP_COUNT* value lowers the precision of the generated predictions. Conversely, high *TH_HIGH_TOP_COUNT* values disable the system from catching features that are in fact "highly". Similar behaviors observed for all these categories. For genre category with a small number of possible features compared to casting and directors, the optimal *TH_HIGH_TOP_COUNT* threshold is *TH_HIGH_TOP_COUNT = 50*, which is much higher compared to the other two categories with a large number of features. The optimal *TH_HIGH_TOP_COUNT* threshold is *TH_HIGH_TOP_COUNT = 6* for casting *TH_HIGH_TOP_COUNT = 4* for director dimensions.

These observed values also confirm with the statistics we collected from the data set. For instance, for the director dimension, we compute the maximum occurrence value of each of the directors in each user profile (considering all 6040 users). These values vary from 1 to 8. We averaged these maximum occurrence values (not including value 1) for all the users, which results in an average value of value *6* for directors. We also compute the average occurrence of every feature in the director dimension for each user (not including 1) and we average all these values for all the users, which we found as 2. Therefore, we tried *TH_HIGH_TOP_COUNT* values for director dimension starting from *2* at least up to *7*, until we get a nearly stable decrease in precision. We completed this procedure for genre and casting dimensions also, and we determine which values to use for trying to find optimal values for *TH_HIGH_TOP_COUNT* in this way. The resulting values are meaningful considering these statistics.

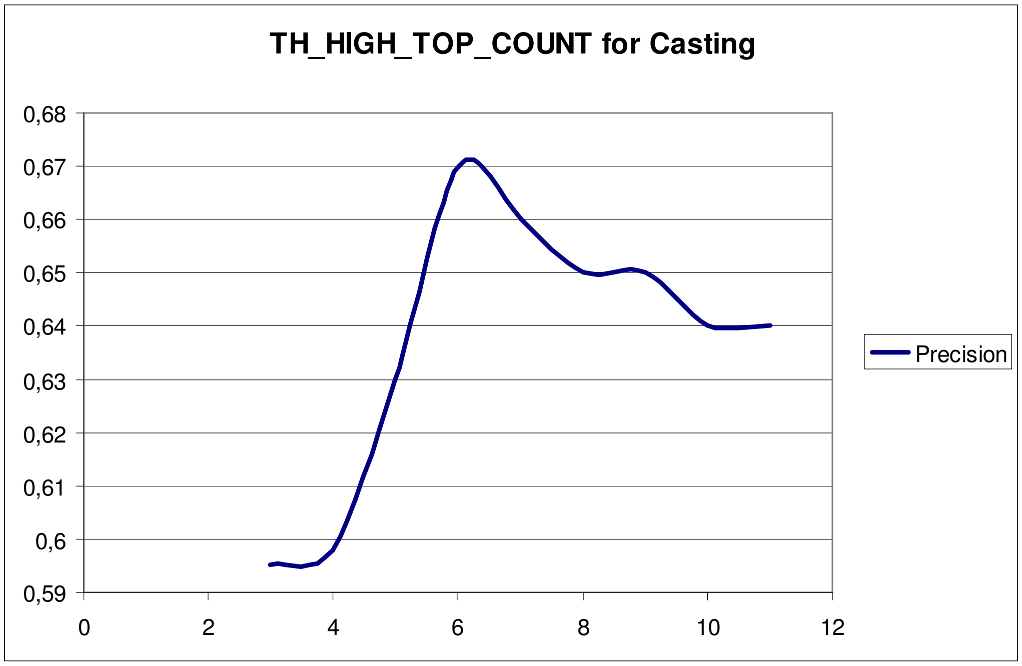


Figure 10 –Precision vs. *TH_HIGH_TOP_COUNT* Threshold for *Casting* Dimension

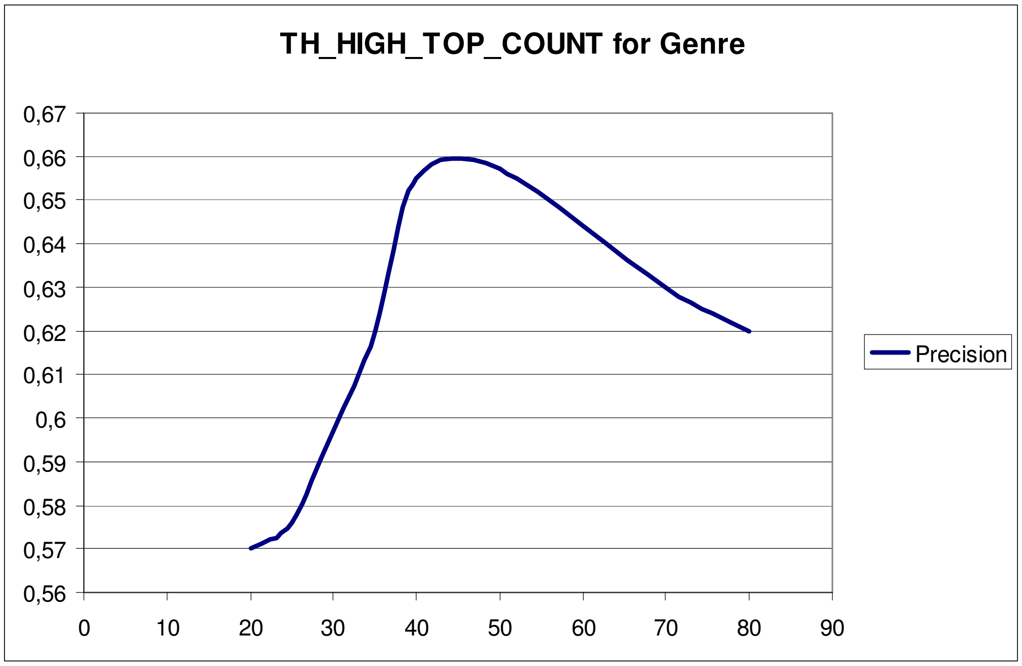


Figure 11 – Precision vs. *TH_HIGH_TOP_COUNT* Threshold for *Genre* Dimension

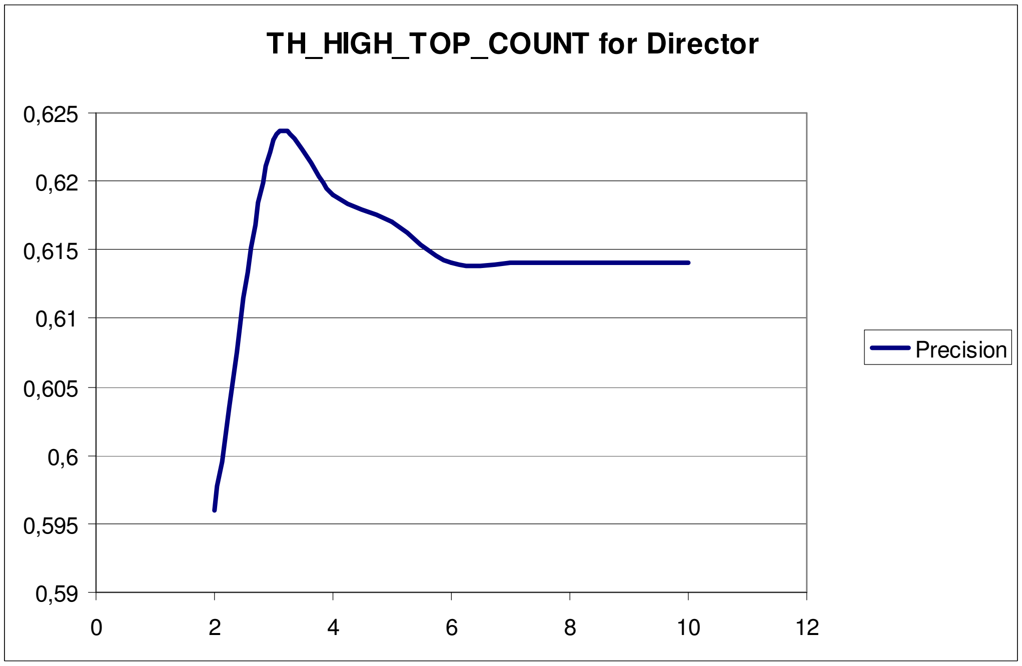


Figure 12 – Precision vs. *TH_HIGH_TOP_COUNT* Threshold for *Director Dimension*

After we have completed tests with *TH_HIGH_TOP_COUNT*, we have tested *TH_HIGH_RATIO* for values *{0.60, 0.70, 0.75, 0.80, 0.85, and 0.90}*. During this experiment, we have set *TH_HIGH_TOP_COUNT* for each of the three dimensions to their optimal values, and include the "highly positive" and "highly negative" features for all these three dimensions to be taken into account during the prediction generation process, since we tried to find the best value for *TH_HIGH_RATIO* considering all these three dimensions. The results of this experiment are shown in Figure 13. As it the case for other previous experiments, the horizontal axis shows the *TH_HIGH_RATIO* values and the vertical – the precision values. The figure is similar to other observations, as expected. Low values of *TH_HIGH_RATIO* leads to incorrect assignments of features of the three dimensions to "highly positive" and "highly negative", and this decreases precision. However, higher values than *TH_HIGH_RATIO = 0.75* prevents the system to assign features to "highly positive" and "highly negative" which again decreases precision. After *TH_HIGH_RATIO = 0.90*, the precision value stabilizes since no features can be assigned to "highly positive" or "highly negative" at all.

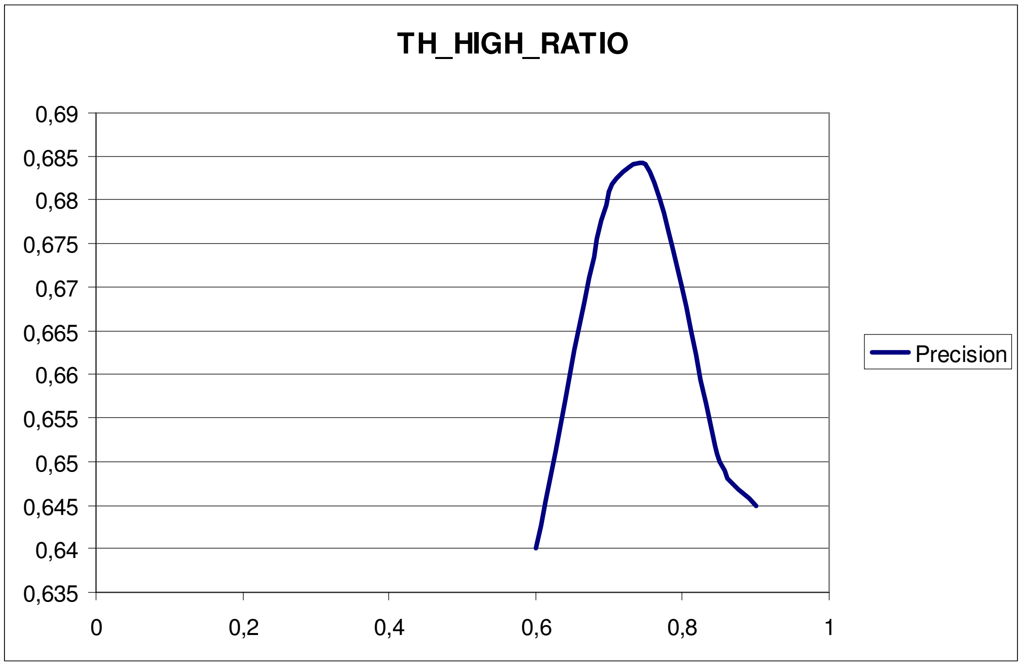


Figure 13 – Precision vs. *TH_HIGH_RATIO* Threshold

5.3.2 Second Set of Experiments

The determined *TH_DONT_CARE*, *TH_HIGH_TOP_COUNT* and *TH_HIGH_RATIO* thresholds were applied in the second set of experiments, which are designed to compare the performance of the pure content-based predictor with the content-based predictor that uses optimizations. In Exp2.1 we get results for the pure content-based recommender, and in Exp2.2 we evaluate the effect of the "don't care" features and "highly" features in order to get the effect of the optimizations on the predictions of our recommender. For these experiments, we used 5-fold cross validation as described in section 5.1. In addition, we conducted tests using all the users, that is, 6040 users that exist in the dataset.

5.3.2.1 Experimental Results

The results of the experiments are shown in Table 12, in terms of the *precision*, *recall*, *F-measure* and *accuracy*.

Table 12 – Results of Second Group of Experiments

Experiment	Precision (%)	Recall (%)	F-Measure (%)	Accuracy (%)
Exp2.1	*60.09*	*91.8*	*71.27*	*64.58*
Exp2.2	*62.02*	*91.7*	*72.84*	*66.04*

As it can be seen from the results, optimizations increase the precision, F-measure and accuracy values; however, there is a very small decrease in the recall value, which can be explained by the effect of assigning features to "don't care", therefore cannot be producing scores for some of the movies. The combined system performs 3.21% better on the precision metric. When F1-measure is considered, there is 2.20% increase and accuracy increased by 2.26%. Recall rate is significantly high and this is an important benefit of the content-based recommender compared to collaborative recommenders. This is due to the fact that, collaborative recommenders can not make predictions unless a movie is rated by a sufficient number of users. However, our content-based recommender can generate a recommendation score for almost every movie and the high recall rate shows that the system can capture most of the relevant movies of the users.

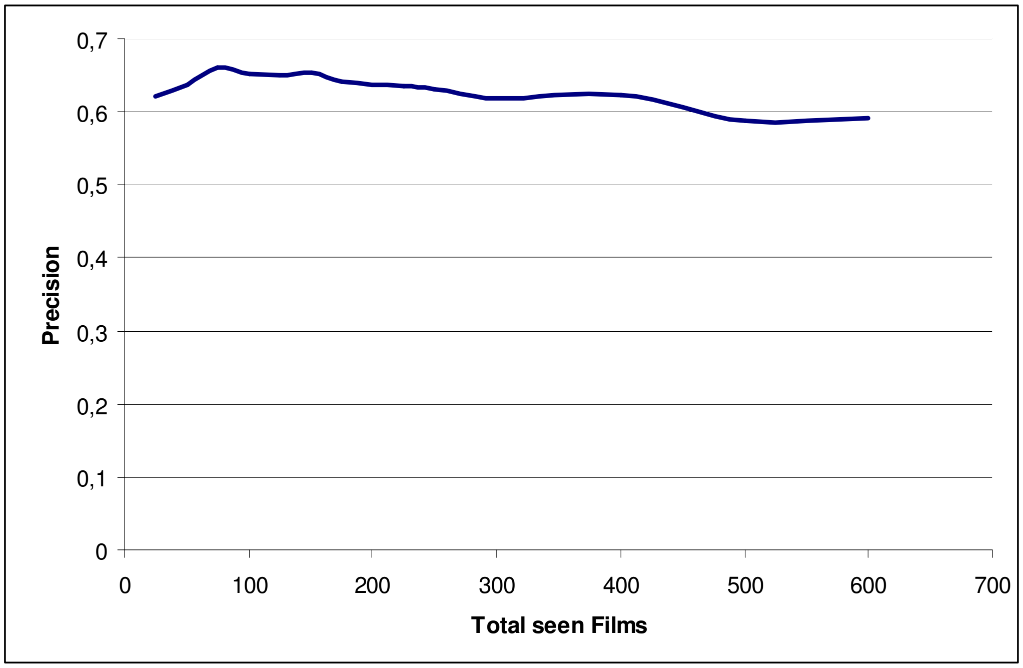


Figure 14 – Precision vs. Total Seen Films

In Figure 14, we have provided the precision of the system as a function of the total number of the films seen. There is not a very significant variance in the precision when the total number of the rated movies is considered. However, the chart shows that, the precision falls as the user has evaluated movies more than 400, and this can be explained by the effect of the neutral features which can not be identified as "don't care" features by the recommender. More specifically, as the user rates more movies, the number of features that exist in his profile increases a lot, which results in many neutral features. These features cannot be identified as "don't care" features since they do not contained in the sufficient number of rated movies to pass the *TH_DONT_CARE* threshold. For instance, there can be many features from the casting dimension that have been involved in only three or four rated movies, which cannot be identified as "don't care", however, which are in fact neutral. When the user has rated a large number of movies, these neutral features' side effect exists more. Therefore this large number of neutral features hampers the accuracy of the generated predictions. However, one more thing to consider is that, only for below 400 rated movies, the performance is below the average and according to Table 9, only 601 users have rated more than 400 movies, so systems performance is below the average only for 9,95% of the users.

In addition, the performance of the system is high even with the small number of rated movies, which is an advantage over the collaborative filtering systems; since collaborative systems' performance is relatively low when the total number of rated movies is low. This is due to the fact that, it becomes difficult for the collaborative recommenders to find nearest peers in that case [61].

5.3.2.2 Comparative Performance Results

Many of the movie recommenders use collaborative filtering recommendation technique, and it is hard to find movie recommenders that conducted content-based filtering, and used the same metrics and dataset that were used to evaluate OPENMORE. In [64], the performance results for different recommenders are given in terms of precision metric; however, information regarding the dataset used in the evaluation and the evaluation procedures were not mentioned exactly. The results from [64] are displayed in Table 13 in order to compare the systems' performance with the existing studies. MovieLens, a collaborative filtering recommender is mentioned in section 3.3.2. MovieMagician [64] is a hybrid recommender system that provides a rating prediction when requested.

Table 13 – Comparative Performance Results

Methodology	Precision (%)	Recall (%)
MovieLens	*66*	*74*
MovieMagician Feature-Based	*61*	*75*
MovieMagician Clique-Based	*74*	*73*
MovieMagician Hybrid	*73*	*56*
OPENMORE	*62.02*	*91.7*

As it can be seen from the results, our system's precision values are lower than MovieLens' and *MovieMagician'* except Feature-Based version. However, our system outperforms the systems mentioned in Table 13 on the recall metric. This is the main advantage of the content-based predictors, since most of the collaborative-based methods, like MovieLens, cannot produce predictions for the movies that were not rated by any peer users which results in bad performance in terms of recall metric.

In addition, the performance of OPENMORE will be different if the evaluation is done by using another dataset and if the randomly selected users from the dataset were used. In most of the studies, only a randomly selected users' data is used in the evaluation, not the whole dataset. However, we have used the whole dataset without any randomness in the evaluation of our proposed system. In addition, the system's performance is best for the users who rated at most 70 movies. It is mentioned in [61] that, *85.92%* of the users in the commonly used EachMovie dataset (*http://research.compaq.com/SRC/eachmovie/*) rated up to *75* movies. Therefore, if that dataset were conducted, the performance results would be better, since OPENMORE may produce better results for these users who have rated a small number of movies and who comprises more than half of the whole dataset.

CHAPTER 6

CONCLUSIONS

6.1 Summing Up

In this thesis work, a content-based approach to movie recommendation is presented. The translation from collaborative to content-based user models through allowing a content-based recommender to generate recommendations for a new user, whose user model was imported from a collaborative recommender, is discussed. The proposed system, OPENMORE, first constructs item profiles from the information it collects from IMDb database. It uses this information to build movie profiles through assigning feature weights to each feature in the movie domain. Two hypotheses were discussed, which determine the discrimination degrees of the features and which are used in the calculation of feature weights. Then these item profiles are used together with collaborative user models, that is, the rating data of the users in order to build content-based user models. The content-based user models are fine-tuned through optimizations of “don't care” and “highly” features. In addition to these, the system is designed and implemented such that, it presents the user models to the users and it enables the users to edit them in a limited way through assigning the features to “don’t care”. Explanations of the recommendations are provided to the users and users can create multiple content preferences which enable them to create filters of their own. The editability of the open user models, positive and negative feedback facilities allow users to change the constructed user profiles and provide missing information or to correct errors, which increase the quality of the produced suggestions. This effectively provides a mechanism for the user model to be examined and edited and thus to tune the adaptation process.

The experimental study described in Chapter 5 first focused on determining the thresholds which are used in the optimization of the constructed user models. First the pure content-based recommender is evaluated and then the thresholds were applied and the accuracy of the generated content-based predictions of the recommender that uses the optimizations was measured, so as to find out the degree of improvement gained by the optimization strategies. In all these experiments, we have used the publicly available movie rating dataset MovieLens. In the first group of experiments that we have tried to find the optimal values of the thresholds, we used *200* randomly selected users and applied precision metric to evaluate the results. In the second group we included all *6040* users without selecting randomly and obtained the results.

6.2 Discussion and Outlook

The experiments showed that for more than 90 % percent of the users, the accuracy of the system is above the average. In addition, for users who have rated up to nearly 60 movies, the accuracy is highest. This is an advantage of the content based recommender over the collaborative recommenders, since collaborative recommenders do not perform well on such user models. The discussed prediction mechanism can be enhanced in the future so that combined features can be taken into account as one more optimization that will increase the accuracy. For instance, when one feature occurs with some other, this will result in bad recommendation and if such cases can be captured, this will increase the prediction accuracy of the recommender.

In addition, IMDb have plot keywords available for the movies, which give important clues about the movie synopsis, and therefore content. This information can be integrated into OPENMORE as a new dimension, which will not require a major change in the current version. Trust in content-based recommendation systems can be further studied and how the system can be enhanced to gather more trust from the users will be considered attentively. In addition, the system proposed in this thesis study mainly uses a content-based approach; however, a collaborative recommendation engine can be integrated into this content-based recommendation system as a future line of work, in order to provide more accurate recommendations. This engine would be under the control of the users, so if the users want other peoples' ideas to affect their recommendations, collaborative engine would come to work.

REFERENCES

[1] WIKIPEDIA, Recommendation Systems, http://en.wikipedia.org/wiki/Recommendation _ system, Last accessed on January 2008.

[2] The Internet Movie Database (IMDb), http://www.imdb.com/, Last accessed on January 2008.

[3] Amazon.com, www.amazon.com, Last accessed on January 2008.

[4] E! Online, www.moviefinder.com, Last accessed on January 2008.

[5] MovieLens, www.movielens.umn.edu, Last accessed on January 2008.

[6] PANDORA, www.pandora.com, Last accessed on January 2008.

[7] last.fm, www.last.fm, Last accessed on January 2008.

[8] J. L. Herlocker, J. A. Konstan, A. Borchers, J. Riedl, "An algorithmic framework for performing collaborative filtering. ", In Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99), 230–237, 1999.

[9] E. Rich, "User Modeling via Stereotypes", Cognitive Science, vol. 3, no. 4, pp. 329-354, 1979.

[10] M.J.D. Powell, “Approximation Theory and Methods”, CambridgeUniv. Press, 1981.

[11] G. Salton, “Automatic Text Processing”, Addison-Wesley, 1989.

[12] J.S. Armstrong, “Principles of Forecasting-A Handbook for Researchers and Practitioners”, Kluwer Academic, 2001.

[13] B.P.S. Murthi, S. Sarkar, “The Role of the Management Sciences in Research on Personalization”, Management Science, vol. 49, no. 10, pp. 1344-1362, 2003.

[14] G.L. Lilien, P. Kotler, K.S. Moorthy, “Marketing Models”, Prentice Hall, 1992.

[15] Adomavicius G., Tuzhilin A. “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions”, IEEE Transactions on Knowledge and Data Engineering 17, 734–749, June 2005.

[16] W. Hill, L. Stead, M. Rosenstein, G. Furnas, “Recommending and Evaluating Choices in a Virtual Community of Use”, Proc. Conf. Human Factors in Computing Systems, 1995.

[17] P. Resnick, N. Iakovou, M. Sushak, P. Bergstrom, J. Riedl, “GroupLens: An Open Architecture for Collaborative Filtering of Netnews”, Proc. 1994 Computer Supported Cooperative Work Conf., 1994.

[18] U. Shardanand, P. Maes, “Social Information Filtering: Algorithms for Automating ‘Word of Mouth’”, Proc. Conf. Human Factors in Computing Systems, 1995.

[19] Balabanovic M., Shoham Y., “Fab: Content-based, collaborative recommendation”, Communications of the ACM, 66–72, 1997.

[20] Resnick P., and Varian H. R., “Recommender systems”, Communications of the ACM 40, 56–58, March 1997.

[21] Riedl J., Konstan J., “Word of Mouth: The Marketing Power of Collaborative Filtering”, New York, NY: Warner Books, 2002.

[22] Belkin N. J., Croft W. B., “Information filtering and information retrieval: Two sides of the same coin?”, Communications of the ACM 35, 29–39, December 1992.

[23] Web Information Systems and Technologies, http://www.webist.org/., Last accessed on January 2008.

[24] Schafer J. B., Konstan J., Riedl J., “Recommender Systems in E-Commerce”, In EC ’99: Proceedings of the First ACM Conference on Electronic Commerce, Denver, CO, pp. 158-166, 1999.

[25] Burke R., “Hybrid Recommender Systems: Survey and Experiments”, User Modeling and User-Adapted Interaction. 12(4), pages 331-370, 2002.

[26] Billsus D., Pazzani M., “User Modeling for Adaptive News Access”. User-Modeling and User-Adapted Interaction 10(2-3), 147-180, 2000.

[27] AAAI'07 Workshop on Recommender Systems in e-Commerce, http://forwarding-iwas.uni-klu.ac.at/AAAI07-WS-Recommender-Systems/, Last accessed on January 2008.

[28] ACM Recommender Systems, http://recsys.acm.org/program.html, Last accessed on January 2008.

[29] Hill W., Stead L., Rosenstein M., Furnas G., “Recommending and evaluating choices in a virtual community of use”, In CHI ’95: Conference Proceedings on Human Factors in Computing Systems, Denver, CO,pp. 194-201, 1995.

[30] Breese J. S., Heckerman D., Kadie C., “Empirical analysis of predictive algorithms for collaborative filtering”, In Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence, pp. 43-52, 1998.

[31] e-challenges, http://www.echallenges.org/e2007/default.asp, Last accessed on January 2008.

[32] Oracle JDeveloper Software, http://www.oracle.com/technology/software/products/jdev/index.html, Last accessed on January 2008.

[33] Lang K., “Newsweeder: Learning to filter news”, In Proceedings of the 12th International Conference on Machine Learning, Lake Tahoe, CA, pp. 331-339, 1995.

[34] Guttman R., H. Moukas, A. G., Maes P, “Agent-Mediated Electronic Commerce: A Survey”, KnowledgeEngineering Review, 13 (2), 147-159, 1998.

[35] The Apache Software Foundation, www.apache.org, Last accessed on January 2008.

[36] Brin, S., Page L., “The anatomy of a large-scale hypertextual Web search engine”, Computer Networksand ISDN Systems, 30(1-7), 107-117, 1998.

[37] Thomas Hofmann, “Collaborative Filtering via Gaussian Probabilistic Latent Semantic Analysis”, Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2003), 2003.

[38] Rich, E, “User Modeling via Stereotypes”, Cognitive Science 3, 329-354, 1979.

[39] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, “Analysis of recommendation algorithms fore-commerce”, In Proceedings of ACM E-Commerce, 2000.

[40] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., Sartin, M, “Combining Content-Based and Collaborative Filters in an Online Newspaper”, SIGIR ’99 Workshop on Recommender Systems Algorithms and Evaluation, Berkeley, 1999.

[41] Linden G, Smith B, York J, “Amazon.com recommendations: item-to-item collaborative filtering”, Internet Computing, IEEE, Vol. 7, No. 1, pp. 76-80, 2003.

[42] CDNOW, www.cdnow.com, Last accessed on January 2008.

[43] G. Adomavicius, R. Sankaranarayanan, S. Sen, A. Tuzhilin, Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach.", ACM Transactions on Information Systems, vol. 23, no. 1, January 2005.

[44] WIKIPEDIA, Music Genome Project Attributes, http://en.wikipedia.org/wiki/List_of_Music_Genome_Project_attributes, Last accessed on January 2008.

[45] Mooney, R. J., Roy L, "Content-Based Book Recommending Using Learning for Text Categorization", SIGIR '99 Workshop on Recommender Systems Algorithms and Evaluation. Berkeley, CA, 1999.

[46] MYSQL, www.mysql.com, Last accessed on January 2008.

[47] M. Pazzani, "A Framework for Collaborative, Content-Based, and Demographic Filtering", Artificial Intelligence Rev., pp. 393-408, Dec.1999.

[48] F. Sebastiani, "ML in Aut. TC", ACM Comp. Surv, 2002.

[49] Konstan, J. A. Miller, B. N. Maltz, D. Herlocker, J. L. Gordon, L. R. Riedl, J, "GroupLens: Applying Collaborative Filtering to Usenet News", Communications of the ACM 40(3), 77-87, March 1997.

[50] Terveen, L., Hill, W., Amento, B., McDonald, D., Creter, J., "PHOAKS: A System for Sharing Recommendations", CACM 40(3) pp.59-62, 1997.

[51] C. Basu, H. Hirsh, W. Cohen, "Recommendation as Classification: Using Social and Content-Based Information in Recommendation", Recommender Systems Papers from 1998 Workshop, Technical Report WS-98-08, AAAI Press, 1998.

[52] GroupLens Research, www.grouplens.org, Last accessed on January 2008.

[53] Herlocker, J.L., Konstan, J. A., Riedl, J., "Explaining Collaborative Filtering Recommendations", Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work, 2000.

[54] D. Billsus, M.J.Pazzani, "Learning collaborative information filters", In Proceedings of the 15th National Conference on Artificial Intelligence, 1998.

[55] Miles Efron, Gary Geisler, "Is it all About Connections? Factors Affecting the Performance of a Link-Based Recommender System", Forthcoming in the Proceedings of the SIGIR 2001 Workshop on Recommender Systems, New Orleans, LA, 2001.

[56] Recommendation Explorer Project, http://video.ils.unc.edu/recex/, Last accessed on January 2008.

[57] Reel.com, www.reel.comhttp://video.ils.unc.edu/recex/, Last accessed on January 2008.

[58] B. J.Dahlen, J. A. Konstan, J. L. Herlocker, N.Good, A. Borchers, J. Riedl, "Jumpstarting movielens: User benefits of starting a collaborative filtering system with dead data", TR 98-017., University of Minnesota, 1998.

[59] NetFlix Online Movie Rentals, www.netflix.comhttp://video.ils.unc.edu/recex/, Last accessed on January 2008.

[60] Movie Reviews, http://www.moviecritic.com.au/., Last accessed on January 2008.

[61] S. Berkovsky, T. Kuflik, F. Ricci, "Cross-Technique Mediation of User Models", In proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, (AH), Dublin, Ireland, 2006.

[62] M. Morita, Y. Shinoda, "Information Filtering Based on User Behavior Analysis and Best Match Text Retrieval", In Proceedings of SIGIR Conference, 272-281, 1994.

[63] M.R. McLaughlin, J. L. Herlocker, "A collaborative filtering algorithm and evaluation metric that accurately model the user experience", SIGIR 2004, 329-336, 2004.

[64] C. Christakou, A. Stafylopatis, "A Hybrid Movie Recommender System Based on Neural Networks", In proceedings of the 5th International Conference on Intelligent Systems Design and Applications, IEEE, 2005,

.

APPENDIX

PUBLICATIONS

Oznur KIRMEMIS, Aysenur BIRTURK, A Content-Based User Model Generation and Optimization Approach for Movie Recommendation, In 6th Workshop on Intelligent Techniques for Web Personalization & Recommender Systems at the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08), 13-17 July 2008, Chicago, Illinois.

Oznur KIRMEMIS, Aysenur BIRTURK, OPENMORE: A User Controlled Content Based Movie Recommender with Explanation and Negative Feedback, In 4th International Conference on Web Information Systems and Technologies (WEBIST '08), 4-7 May 2008, Funchal, Portugal.

Oznur KIRMEMIS, Aysenur BIRTURK, A Content Based Movie Recommendation System, in Expanding the Knowledge Economy: Issues, Applications, Case Studies, Paul Cunningham and Miriam Cunningham(Eds), IOS Press, 2007 Amsterdam, ISBN 978-1-58603-801-4; eChallenges 2007, 24-26 October 2007, The Hague, The Netherlands.