CLUSTER BASED COLLABORATIVE FILTERING WITH INVERTED INDEXING

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Özlem Nurcan Subakan

August, 2005

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. İbrahim Körpeoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. H. Murat Karamüftüoğlu

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

CLUSTER BASED COLLABORATIVE FILTERING WITH INVERTED INDEXING

Özlem Nurcan Subakan
M.S. in Computer Engineering
Supervisor: Prof. Dr. Özgür Ulusoy
August, 2005

Collectively, a population contains vast amounts of knowledge and modern communication technologies that increase the ease of communication. However, it is not feasible for a single person to aggregate the knowledge of thousands or millions of data and extract useful information from it. Collaborative information systems are attempts to harness the knowledge of a population and to present it in a simple, fast and fair manner. Collaborative filtering has been successfully used in domains where the information content is not easily parse-able and traditional information filtering techniques are difficult to apply. Collaborative filtering works over a database of ratings for the items which are rated by users. The computational complexity of these methods grows linearly with the number of customers which can reach to several millions in typical commercial applications. To address the scalability concern, we have developed an efficient collaborative filtering technique by applying user clustering and using a specific inverted index structure (so called cluster-skipping inverted index structure) that is tailored for clustered environments. We show that the predictive accuracy of the system is comparable with the collaborative filtering algorithms without clustering, whereas the efficiency is far more improved.

Keywords: Collaborative filtering, recommender systems, clustering, inverted files, performance evaluation.

ÖZET

EVİRİLMİŞ DİZİN YAPILI VE TOPAKLAMA TEMELLİ İMECELİ SÜZGEÇLEME

Özlem Nurcan Subakan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Özgür Ulusoy
Ağustos, 2005

Çağımız toplumları, iletişimi kolaylaştıran büyük ölçekli bilgi ve çağdaş teknolojilere sahiptirler. Ancak, bir şahsın çok büyük miktarlardaki verileri tek başına kümeleyip bu verilerden yararlı bilgiler elde etmesi olanaklı değildir. İmeceli bilgi sistemleri bir toplumun bilgilerini basit, hızlı ve adil bir şekilde bir araya toplama çabalarının bütünüdür. İmeceli süzgeçleme, bilgi kaynağının kolayca ayrıştırılamadığı ve geleneksel bilgi süzgeçleme tekniklerinin uygulanmasında zorluklarla karşılaşıldığı alanlarda başarıyla uygulanmaktadır. İmeceli süzgeçleme, kullanıcılar tarafından oylanan bir madde değerlendirme veri tabanı üzerinde çalışmaktadır. Bu yöntemlerin işlemsel karmaşıklıkları tipik ticari uygulamalarda milyonları bulabilecek kullanıcı sayısına doğrusal orantılı olarak artmaktadır. Bu tür ölçeklenirlik kaygılarını ortadan kaldırmak için, kullanıcı topaklaması uygulayan ve topaklanmış ortamlara uydurulmuş belirli evirilmiş dizin yapısında olan (topak atlamalı evirilmiş dizin yapısı da denebilir) verimli imeceli süzgeçleme tekniğini geliştirdik. Bu sistemin öngörücü doğruluğunun topaklama uygulanmayan imeceli süzgeçleme algoritmalarıyla aynı ölçekte olmasına rağmen verimliliğinin çok daha iyileştirilmiş olduğunu gösterdik.

Anahtar sözcükler: İmeceli süzgeçleme, tavsiye sistemleri, topaklama, evirilmiş sistemler, başarım analizi.

Acknowledgement

First and foremost I would like to thank to my supervisor Prof. Dr. Özgür Ulusoy, for giving me the invaluable chance of working with him. He has always encouraged me genuinely and believed me right from the beginning. They are absolutely his exceptional guidance, most pleasant personality and traits as a researcher and advisor what made this graduate study enjoyable and what made others jeaolus.

A gigantic amount of thanks goes to İ. Sengör Altıngövde. Without his motivation, assistance, discussions, "meetings at anywhere and anytime", and all his support, this thesis would not be possible. I owe him a lot.

I am grateful to all of my friends for their motivating and caring friendships, just-in-time buggings and debuggings, and for the significance they brought in my life.

Last, but not least, I want to thank to my family who have been very supportive and encouraging through my long trek, and always have given me the strength when I needed it, they were never afraid to push me and give their opinions even on things they did not understand.

To my mother,

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent years have brought about the exponential growth of the volume of everyday things with the developments in science and technology. The number of products, news, movies, music, books, and the flow of papers is incredibly huge. We are truly in an "information age". People are overwhelmed when browsing through today's information ocean and could not possibly filter through the items in order to select the ones they actually want and need. Which papers should I read to learn about that area? Which movies should I watch? Which books should I buy? People handle this information overload through their own effort. They take into account the recommendation of other people for movies, CDs, books, music, etc. by the word of mouth, by reviews, by surveys, etc. However with the explosive growth of the information, this way of filtering becomes less and less a factor. As the world becomes more digital and interactive, more and more options will become available. We then will need much more time and effort than we can dedicate in order to face this information process.

The field of information filtering, in general terms, attempts to automate this process by supporting people by recommending the items they would really want and need and eliminating the things that they do not want to be bothered with.

Currently, the application domain of recommendation is also very wide: news [6], research papers [29], music [36], radio [20], movies [30], Web pages [41], jokes [19], etc.

1.1 Common Methods to Handle Information Overload

Currently, there are three different techniques commonly used to tackle the information overload challenges: *Information Retrieval (IR), Content Based Filtering (CBF) and Collaborative Filtering (CF).*

1.1.1 Information Retrieval

> "Information retrieval is the art and science of searching for information in documents, searching for documents themselves, searching for metadata which describes documents, or searching within databases, whether relational stand alone databases or hypertext networked databases such as the Internet or intranets, for text, sound, images or data." [1]

[1] http://en.wikipedia.org/wiki/Information_retrieval

In that sense, IR focuses on allowing users to express queries, and then select the items that match a topic of interest. Internet search engines such as Google [18] and Yahoo! [42] are popular information retrieval systems. With the exponential expansion of the Internet, such search engines may return millions of pages for one keyword search. Each year, journals, conferences and magazines report on thousands of researches. Intrinsically, retrieving the most relevant information from the Web is still difficult.

1.1.2 Content Based Filtering

This is the common and obvious technique used in this domain and also called *information filtering*. The filter selects the items for the user's consumption based upon the correlations between the syntactic and semantic content of the items and the user's preferences. In content based filtering, the items must be in machine parse-able form, or features must have been assigned to the items manually. In that sense, application domain is restricted. System Lira which recommends Web pages is an example of content based filter [4].

1.1.3 Collaborative Filtering

Collaborative filtering is a popular technique for information overload which has been developed over the past decade. It works over a database of ratings for items by users. It is based on the collaboration among the users, ideally like-minded users. Therefore, collaborative filtering is also called *social filtering*. There are generally two cases: It either provides prediction for some item or outputs a recommendation (suggestion) list for some user based on the similarities between

user rating histories. The logic behind collaborative filtering based systems is that each user belongs to a community of like-minded people; hence the items favored by these users can be used to form predictions or suggestions. A collaborative filtering based system maintains a user profile database, which records each user's interests (positive and negative) in specific items. Then it compares the active user's profile to all the other profiles and weighs each profile for its degree of similarity with the active user's profile. Finally, the system considers a set of the most similar profiles and uses information contained in them to recommend items to the users. Collaborative filtering has been successfully used in domains where the information content is not easily parse-able and traditional information filtering techniques are difficult to apply [32].

1.2 Motivation

As we have discussed above, the main task in collaborative filtering systems is defining a peer group and predicting the votes for the active user effectively and efficiently. When a collaborative filtering system just starts, there does not exist enough ranking data and it is hard to find the peer groups, so the accuracy of the recommendation will not be good. After some time, when the database becomes bigger, the efficiency of the system is influenced since the collaborative filters need more time to scan the database to find the like-minded people. Therefore, the major challenges for collaborative filtering are the effectiveness and the efficiency.

This thesis presents cluster based collaborative filtering with inverted indexing and a hybrid approach that combines collaborative filtering with content based filtering by a two-stage clustering based on first the attributes of the items to

be recommended and then the users' profiles. Our approach is better than the current social filters in terms of efficiency and it shows comparable accuracy.

The contributions of this thesis are as follows:

- We improve the efficiency of collaborative filtering by applying user clustering and using a specific inverted index structure (so called cluster-skipping inverted index structure), that is tailored for clustered environments. We show that the predictive accuracy of the system is comparable with the collaborative filtering algorithms without clustering, whereas the efficiency is far more improved.

- We present a hybrid filter which combines content based filtering with collaborative filtering by a two-stage clustering, i.e., first clustering the items and then imposing user clusters on top of these item clusters. Although clustering has been applied for users and items in previous works [39], and inverted index structure for collaborative filtering has been adopted in a nonclustered environment in a very recent work [14] separately, we are not aware of any other studies that combine user clustering with an inverted index structure.

- We choose our application domain as movie recommendation systems. Experimental results demonstrate the capabilities of the proposed technique and its potential for immediate application. A prototype of the system, namely MoRec, is currently in use.

1.3 Overview of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2, we discuss previous research in this area. In Chapter 3, we describe cluster based collaborative filtering with inverted indexing and hybrid filtering. In Chapter 4, we provide the experimental settings, and present the results for the proposed approaches. In the last chapter, we detail possibilities for future work for the system and for recommendation systems in general, and present our conclusions.

Chapter 2

Background and Related Work

The concept of collaborative filtering descends from the work in the area of information filtering. The term collaborative filtering was introduced by Goldberg et al. [17] who published about collaborative filtering techniques in the filtering of information. They developed a system called Tapestry for filtering emails. Tapestry accepted the ratings or annotations of the users for the items, in this case electronic documents such as emails and Netnews. As users read documents they attach annotations to the documents. The filters that search the annotations were however constructed by the user, using complex queries. The collaborative filtering provided by Tapestry was not automated and required users to make complex queries. The query may involve keywords, subject, authors, etc. and the annotations given by the people. In this system, you still had to know who the people are with tastes like yourself.

First automated collaborative system was introduced by GroupLens which provided personalized predictions for UseNet news articles using a neighborhood

based algorithm. It used Pearson correlations to weigh user similarity, used all correlated neighbors and presented a weighted average of deviations from the neighbor's mean as the final prediction [32]. [1]

In 1996, interest in collaborative filtering led to a workshop on the topic at the University of California, Berkeley. The results of this workshop led to Communications of the ACM special issue on *recommender systems*, the term was introduced by Resnick and Varian [33]. Resnick and Varian define a recommender as a system which accepts user models as input, aggregates them, and returns recommendations to users. Two early collaborative filtering recommender systems were Firefly and LikeMinds. Firefly evolved from Ringo [36] and HOMR (Helpful OnlineMusic Recommendation Service) and allows a website to make intelligent book, movie, or music recommendations [2]. Firefly's underlying algorithm is now used to power the recommendation engines of sites such as BarnesandNoble.com. LikeMinds was acquired by MacroMedia. Various commercial sites make use of collaborative based IF, including CDNow.com, reel.com, and Amazon.com. Another example for collaborative filtering based systems is the FAB system, which is a Web page recommendation system that computes the similarity between user profiles to identify the advisors for active users [3]. In FAB system, Web pages that are highly rated by the advisors are recommended to the active user. WebWatcher is another example of collaborative filters [27]. After been fed some information, WebWatcher can accompany the users about which links to take from a Web page on a website based on a short description of the user's interest which is learned from the user when entering the site. By tagging each page the interests of all users who went to this page, the system is

[1] GroupLens technology is commercially available through Net Perceptions.
[2] FireFly was initially marketed by Agents Inc., which was later acquired by Microsoft.

able to compare a user's interest with this community interest and recommend whether to follow the link going to that page.

Collaborative approaches constitute the main thrust of current recommender systems research. Once users are modeled, the process of collaborative filtering can be viewed operationally as a function which accepts a representation of users and items as input and returns a recommended subset of those items as output. Since people's likes and dislikes are naturally not orthogonal it can be claimed that collaborative filtering for recommendation is an effective technique.

Although there seems to be an increasingly strong demand for collaborative filtering techniques, only a few different algorithms have been proposed in the literature thus far, and there have been limited published results on the relative performance of various algorithms used in collaborative filtering systems. Currently, the algorithms can be classified into memory-based and model-based algorithms [8]. Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings whereas memory-based algorithms utilize the entire user-item database to generate a prediction. Model-based algorithms require more time to train but can provide predictions in shorter time in comparison to memory-based algorithms. Memory-based algorithms repeatedly scan the preference (or profile) database to locate the peer groups for the active users. A prediction is then computed by weighing the votes of the users in the peer groups. The people in the peer groups are identified based on their similarity in preferences to the active user in-memory. Then, different algorithms are employed to provide a prediction or a top-N recommendation list for the active user. Consequently, these algorithms can be equivalently called

correlation-based or *nearest-neighbor collaborative filters*. This structure is dynamic and immediately reacts to changes in the user database. Every new rating added to the user database is included in the neighborhood calculation, since similarities between users are calculated in memory when needed. Model-based systems use a probabilistic approach and compute the expected value of a user prediction, given his/her ratings on other items. In terms of accuracy, model based algorithms performs as well as memory based ones [8]. These algorithms do not suffer from memory bottlenecks found in memory based predictions. However, the inherent static structure of these techniques makes it difficult to update the model without rebuilding it. The model building process is performed by different *machine learning* algorithms such as *Bayesian network*, *clustering*, and *rule-based* approaches. The Bayesian network model [8] formulates a probabilistic model for collaborative filtering problem. The rule-based approach applies association rule discovery algorithms to find association between co-purchased items and then generates item recommendation based on the strength of the association between items. The clustering model treats collaborative filtering as a classification problem [39], and works by clustering similar users in the same class and estimating the probability that a particular user is in a particular class, and computes the conditional probability of ratings.

In general terms, clustering is the unsupervised division of data into group of similar objects. In our case, either people or items or both are grouped into clusters based on their similarity of interest. Some systems cluster items based on user rating data [31]. Some others cluster users based on a model; generally compute conditional probabilities of votes for certain items for each cluster, estimate probability distributions for the active user for being in each cluster, and return weighted sum of the votes. Also there exist models which cluster both users and

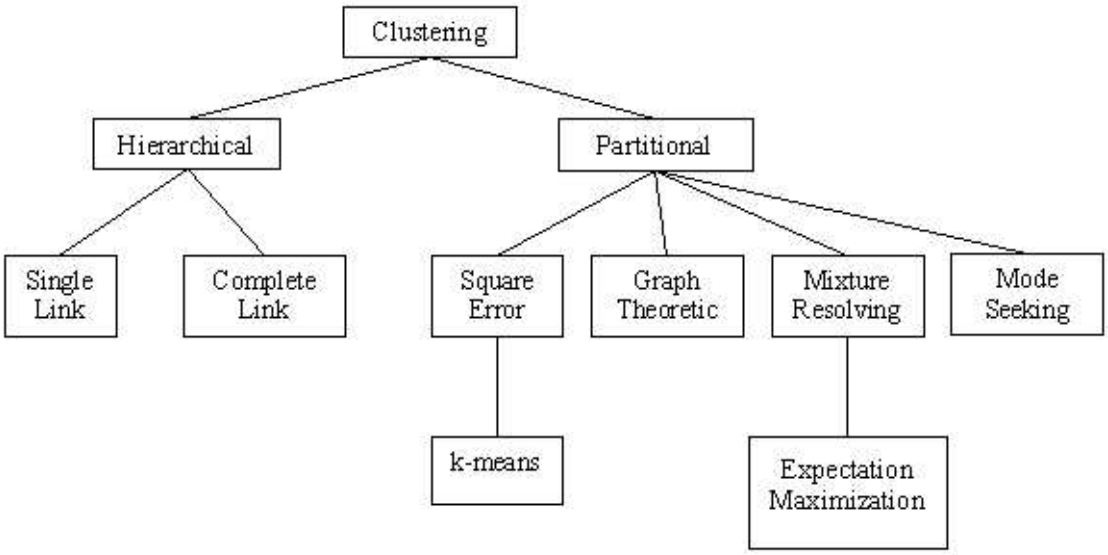

Figure 2.1: Clustering methods [25]

items together simultaneously without assuming that each user and item should only belong to a single cluster [37]. Besides, some approaches tend to cluster the items returned as a result of a user query in an information retrieval system, which may be called *post-retrieval clustering* [38].

There are many different ways to express the clustering problem. For instance, the clusters that are identified may be exclusive, so that every object belongs to only one group. Or, they may be overlapping, so that one object may fall into several clusters, or they may be probabilistic, where an instance belongs to each group according to a probability, or they may be hierarchical, such that there is a rough division of the objects into clusters at a high level [25]. Figure 2.1 classifies several clustering methods.

In hierarchical clustering the data are not partitioned into a particular cluster in a single step. Instead, a series of partitions takes place, which may run from a

single cluster containing all objects to n clusters each containing a single object. The end result of the algorithm is a tree of clusters called a *dendrogram*, which shows how the clusters are related. By cutting the dendrogram at a desired level, a clustering of the data items into disjoint groups is obtained. Partitional clustering, on the other hand, attempts to directly decompose the data set into a set of disjoint clusters. They obtain a single partition of the data instead of a clustering structure, such as the dendrogram produced by a hierarchical technique. Partitional methods have advantages in applications involving large data sets for which the construction of a dendrogram is computationally expensive.

The popular clustering algorithms used in this domain are *k-nearest neighbor* and *k-means*. The K-means algorithm is based on a very simple idea: Given a set of initial clusters, assign each point to one of them, and then each cluster center is replaced by the mean point on the respective cluster. These two simple steps are repeated until convergence. A point is assigned to the cluster which is close in Euclidean distance to the point. It is easy to implement but has two drawbacks. First, it can be slow since in each step the distance between each point to each cluster has to be calculated, which can be expensive in the presence of a large dataset. Second, this method is sensitive to the provided initial clusters. The goal of k-nearest neighbor clustering method is to simply separate the data based on the assumed similarities between various classes. It finds the k observations in the learning set that are closest to some particular item and predicts the class of that item by majority vote, i.e., chooses the cluster that is most common among these k neighbors.

Another clustering method is *Cover-Coefficient Based Clustering*, shortly C^3M. This is a single-pass partitioning algorithm. It creates clusters of items

that belong to only one cluster each. It chooses a set of seed items that are later used as centroids, and assigns each non-seed item to a cluster by calculating the similarity of the document with each centroid. The document is then placed into a cluster whose centroid is most similar to it [12].

Cover Coefficient, CC is the base concept of the C^3M clustering algorithm. CC concept serves:

1. Identifying the relationships among documents of a database by means of a matrix, the so-called CC matrix.
2. Determining and calculating the number of clusters that a document database will have.
3. Selecting cluster seeds using a *cluster seed power*.
4. Forming clusters with respect to C^3M, by applying the concepts 1-3.
5. Correlating the relationships between clustering and indexing.

During clustering and query/user cluster matching, based on the attribute values, the distances between the objects have to be determined. For this purpose, several distance measures, i.e., metrics on the feature space are used to evaluate the similarity of the patterns. Some of these measures are *Euclidean distance*, *Manhattan distance*, *Mahalanobis distance*, *cosine coefficient*, *Dice Coefficients*, *Jaccard Coefficient*, *Hamming distance*, *Pearson Correlation Coefficient* and *vector similarity* as illustrated in Figure 2.2 .

The cosine coefficient, Pearson correlation and the Euclidean distance are the measures that have been commonly used. They work well when the data set has

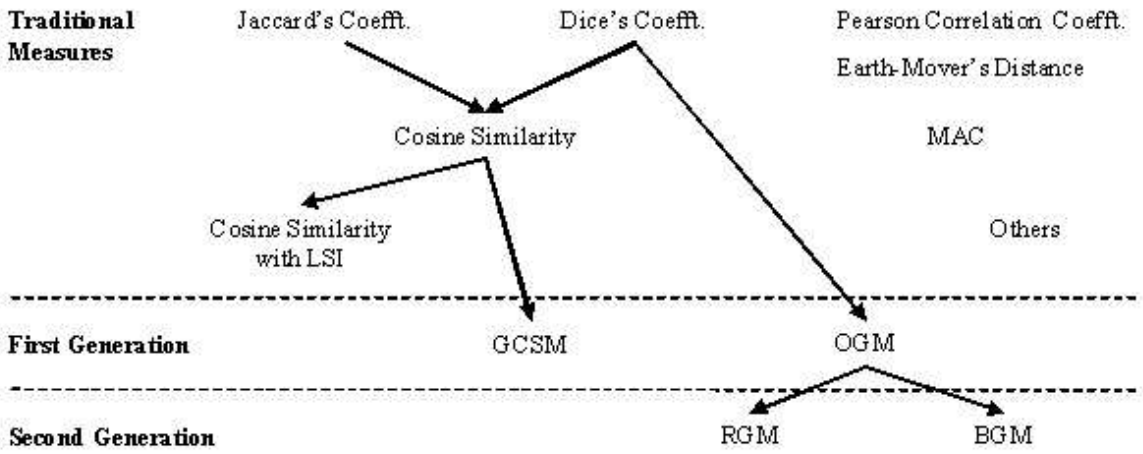

Figure 2.2: Similarity measures [25]

compact or isolated clusters [38]. One way of computing the similarity between two items is to treat each item as a vector in the space of users and use the cosine measure between these vectors as a measure of similarity. Formally, if R is the $n \times m$ user-item matrix, then the similarity between two items x and y is defined as the cosine of the n dimensional vectors corresponding to the xth and yth column of matrix R. The cosine between these vectors is given by

$$sim(x, y) = \cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|_2 \cdot \|\vec{y}\|_2} \tag{2.1}$$

The most common measure for calculating the similarity is the Pearson correlation algorithm. Pearson correlation measures the degree to which a linear relationship exists between two variables. The Pearson correlation between users a and i is defined as in [8]

$$w(a, i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \tag{2.2}$$

where v_a : profile for user a (all votes of user a)

$v_{a,j}$: user a's vote for item j

$\bar{v}_a$: mean value of the votes for user a

j : index that runs over all intersecting items in two profiles, i.e. $j \in \{v_a \cap v_i\}$.

Once clusters are created, recommendations are produced, which can be of two types, given a set of ratings for that user and other users in the system. *Prediction* is a numerical value expressing the predicted score of the item for the active user. *Recommendation list* is a list of N products that the system believes the user will like the most. Simply, a prediction system can be extended to provide recommendations by predicting the user's ratings for all items that have not yet been rated and returning the top-rated items. There are several prediction algorithms in the literature. *SWAMI*, a research conducted for collaborative filtering algorithm development at University of California, Berkeley, exploits three prediction algorithms, namely a *Pearson correlation-based method*, the *support vector method*, and a *scalable Pearson correlation-based method* that uses clustering to improve scalability and accuracy [16]. Correlation-based prediction is one of the mostly used methods in collaborative filtering applications. The underlying idea in correlation-based method is to compute a user's predicted rating of an item as a weighted average of the ratings given to that item by other users. Rather than sum over all of the users in the system to generate the prediction, the algorithm [32] only considers the neighborhood of users who are well correlated with the current users. This is more efficient, since the average is computed over a much smaller set of values, and more accurate, since the votes of potentially

large numbers of poorly correlated users do not affect the current user [22]. Support vector method views the prediction problem as a classification task. It uses existing users to identify voting classes, and then uses these classes as a basis for prediction. In order to overcome the scalability problems with Simple Pearson method, the users are clustered according to their correlation level. Hence, the clusters can be used to find out the neighbors of a specified user. As clustering algorithm, a method similar to k-means algorithm is used. K-means algorithm requires a distance metric between points, however it is not possible to provide a metric value for correlations between users. Therefore, in each iteration the center is determined by finding the user that has the best overall correlation with all the other users in the cluster. The main advantage of Clustered Pearson is improved scalability. However, on the other hand, the method requires a long off-line training time to improve scalability. And also, using k-means algorithm causes instability due to randomized initial cluster selection and due to undefined k value, the number of clusters [16].

For faster calculation of predictions, some approaches adopt disk based inverted file structures [14]. Coster et al. have two reasons for this. Firstly, matching user profiles in a collaborative filtering system can be very expensive. Secondly, if all user preferences stored can be accessed directly from disk it is possible to maintain a much larger set of users and titles. Actually the reason for using inverted file search in this domain is that a user query contains a small percentage of the total number of the items in the document collection. Here a user's votes for items can be seen as a document. Accumulators are stored in main memory during the scan of each inverted list, for holding partial sums of votes. For calculating the similarities between the active user and all other users, several correlation methods are employed. One of them is Simple Pearson

algorithm which is also used in [16]. Another algorithm is extended Pearson algorithm, which uses *Inverse User Frequency* (IUF). The more the number of votes that a title has, the lower weights it is assigned. This is due to the assumption that the titles that are rated by many users are less useful in capturing the similarity between two users. In addition to these, *Default Voting* (DEF) is used in experiments. The idea is to assign default votes for the titles that have not been rated by both users. The value of the default vote depends on the system. Hence, the similarity is calculated by using all the titles. As a fourth method, Default Voting is extended by Inverse User Frequency (DEFIUF). For speeding up the algorithm, some early termination heuristics, such as *Quit* and *Continue*, are also used.

Collaborative filtering recommendation systems are evaluated using different metrics with different data sets. Breese et al. evaluate their algorithm for three different data sets: *i) MS Web*, the data set capturing the individual visits to various areas of the Microsoft corporate web site. *ii) Television* data set using Neilsen network television viewing data for people. *iii) EachMovie* explicit voting system for movie database. For prediction of some specific item x, in order to judge the effectiveness they look at the average absolute deviation of the predicted vote to the actual vote on items on which the users in the test set have actually voted. These scores are then averaged over all the users in the test set of users as in the GroupLens project. For top-N recommendation list, they estimate the expected utility of a particular ranked list to a user. The expected utility of a list is the probability of viewing a recommended item times its utility. In this analysis, they set the utility of an item as the difference between the vote and the default vote in the domain.

SWAMI uses EachMovie data set for experimentation. To examine the efficiency and effectiveness, SWAMI has three metrics: The first is the *mean absolute error*, already in common use in the literature, where the error is the absolute value of the differences between the actual vote and the predicted vote. This measures the accuracy of the prediction algorithm. The second metric is the *variance of the mean absolute error*, which measures how reliable the prediction algorithm is. The third metric, *weighted mean*, aims at measuring how well the prediction algorithm does on the "harder" movies to predict, movies with high vote variance. The weight given to a movie is $\mid true\_user\_vote - mean\_user\_vote \mid$. The weight is higher for movies that are far from the user's mean. Thus, the formula for the weighted mean is $\mid (true\_user\_vote - mean\_user\_vote) * (true\_user\_vote - predicted\_user\_vote) \mid$.

Ungar et al. experiment their system on both synthetic data and real data from *Purchase CDNow*. They test their system on CDNow's customers by sending email recommendations of new artists. They claim that the automated system resulted in doubling of the purchase rate.

Herlocker et al. use *MovieLens* data set for experimentation. They consider three metrics for evaluation. *Coverage* is a measure of the percentage of the items for which a recommendation system can provide predictions. They compute the coverage as the percentage of the items over all users for which a prediction was requested and the system was able to provide a prediction. To assess the *accuracy*, they compute both mean absolute error and root mean squared error as statistical accuracy metric, and *ROC (receiver operator characteristic)* sensitivity as the decision support accuracy metric. ROC measures the *sensitivity* and *specificity* of the test. Sensitivity is the probability of a randomly selected recommendable

(good) item being accepted by the filter, whereas specificity is the probability of a randomly selected bad item being rejected by the filter.

Although collaborative filtering based systems have been successful in several domains, they still possess some problems such as:

Sparsity: One of the biggest problems is the extreme sparsity of the data. Consider that there may be thousands of users, several million items. However, each user may rate only a dozen of items, giving a highly sparse data. So, there will be a real problem in obtaining a large amount of data about any item or user with such little data.

Cold start: The sparsity problem can be difficult to overcome after users have made a large number of recommendations; however it is even harder to overcome when the system has just been started and there are no user recommendations at all.

New item: Similarly, a new item that has not had many ratings also cannot be easily recommended. New item problem can be handled by content information gathered by inferring the similarities between existing items and the new item [35]. Naïve Bayes text classifier applied to person/actor data is used. For each person a separate naïve Bayes classifier is trained so no collaborative information is used. The model is trained with Laplace smoothing. All movies in the data set should be rated. Such hybrid recommendation systems which combine content-based and collaborative filtering can help the new item problems [35].

Scalability: The other major problem affecting most of the recommender systems is their ability to scale up to large systems. With millions of items and

users a typical recommender system will suffer serious scalability problems.

Gray sheep: In a small or medium community of users, there are individuals whose opinions do not agree or disagree with a group of people. These users will rarely receive accurate predictions even after initial start up phase for the user and the system.

On the other hand, in content-based techniques, the user model includes information about the content of items of interest, e.g., whether these are web pages, movies, music, or anything else. The items of interest are defined by their associated features. Using these items as a basis, the technique identifies similar items that are returned as recommendations. A content-based recommender learns a profile of the user's interests based on the features present in objects the user has rated. Schafer et al. call this *item-to-item correlation* [34]. The type of 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.

In content based systems, the first step is to gather content data about the items. For example book title, author, description etc. for the books or the director, cast etc. for the movies are some of the common content information. Most systems use information extraction techniques to extract these data, and information retrieval techniques to retrieve the relevant information [3]. Web crawlers collecting data on the web are common tools in this step. Secondly, user ratings are collected. Then using the using ratings together with the content information, user profiles are compiled. Finally, unrated items' contents are compared with the user profiles and scores are assigned according to the degree of similarity.

Content based techniques might prove highly suitable for users who have specific interests and who are looking for related recommendations. Many machine learning techniques have been applied to this problem. Some researchers working on these, such as Chen and Norcio [13] and Jennings and Higuchi [26] have modeled users with the application of neural network methodology. NewsWeeder [28] , NewsDude [7] also belong to the category of content-based recommender systems.

However, content based filtering has limitations too:

- Either the items must be of machine parse-able form or attributes must be assigned to the items by hand.
- This technique recommends more of what the user has already seen before, i.e. it has no inherent method to generate serendipitous recommendations.
- Content-based techniques also have a start-up problem in that they must accumulate enough ratings to build a reliable classifier.

In light of these issues, we aim an efficient and effective solution for information overload which overcomes as many problems stated as possible. We present collaborative filtering with inverted indexing using either C^3M or K-Means clustering algorithms and enhance the filtering process by adding the content information in a two-stage clustering strategy. Experiments and results show the achievements.

Chapter 3

Methodologies

The approaches we use for handling information overload consist mainly of two dominant research paradigms: content based filtering and collaborative filtering. A pure content based filter recommends items based solely on a profile built up by analyzing the content of the items that a user has rated [3]. Content based filters are less affected by the mentioned problems of the pure collaborative filters because they use techniques that apply across all documents. For example, a filter that predicts high rating for movies with the word "Jedi" in their plot summaries or with the genre "Action" can give the prediction before anyone has watched the movie. Despite these strengths, content based filters alone can prove ineffective. These techniques can have difficulty in deciding between high quality and low quality information on the same theme. Also, as the number of items increases, the number of items in the same content based category increases too, further decreasing the effectiveness of these filters. Collaborative filtering exploits the speed of computers with the intelligence of the people. Collaborative filters correlate the ratings of a user with those of other users to determine how to make

future predictions for the rater. As well, these filters share the ratings with other users so they can use them in giving their own votes.

In light of these, we first introduce a cluster based collaborative filtering technique with a specific inverted index structure to improve the efficiency of the collaborative filtering. We then describe a hybrid filter approach exploiting collaborative filtering added with content information with this inverted indexing, cluster-skipping inverted index structure (CS-IIS), for further improvement in performance. We evaluate the performance of our collaborative and hybrid filtering approaches with a previously proposed technique, collaborative filtering with inverted indexing [14].

3.1 Cluster Based Collaborative Filtering with Cluster-Skipping Inverted Index Structure

Collaborative filtering is probably the most familiar, most widely implemented and most mature of the technologies in recommendation systems. Collaborative filters aggregate ratings of items, recognize the cohesion between users on the basis of their votes for items, and generate new ratings (predictions) based on inter-user similarities.

The greatest strength of collaborative filters is their applicability to all domains of rating data being completely independent of any machine parse-able representations of the items to be recommended. However, it is still challenging to improve the efficiency and the accuracy of the collaborative filters. On this

basis, we try to improve the performance of the collaborative filtering methodology by combining it with clustering algorithms. For comparative purposes, we work on both *K-means* clustering algorithm, which is one of the mostly used algorithms in this area, and *Cover-Coefficient Based Clustering*, shortly C^3M, due to its comparable efficiency [12]. Furthermore, we use a cluster-skipping inverted index structure (CS-IIS) [11], a recently proposed file structure for clustered data sets, for improving efficiency by reducing in-memory computation costs. We improve the scalability of the collaborative filtering technique using these approaches altogether.

Our approach involves a clustering phase of the users according to their correlation in the ratings for the items. Clustering is done by either C^3M or *K-means*. In the search strategy, i.e., cluster-based retrieval (CBR), a two-stage query processing is performed. The queries are first compared with the clusters, or more accurately, with cluster representatives called centroids. Detailed query-by-document comparison is performed only within selected clusters. So, the first stage is selecting the best correlated clusters of a user profile, and the second stage is selecting the best correlated neighbors from these best correlated clusters. Best matching clusters are found by using the previously computed centroids. An inverted index file is also created for the centroids. Then a cluster-skipping inverted index structure is exploited to find the best matching users among the best matching clusters to the given query profile. During this stage, two alternatives can be applied, either all users of the best matching clusters are searched, or only the centroids of the clusters are considered to be the best correlated "virtual" users. We use Simple Pearson correlation (see Equation 2.2) to decide on the degree of similarity in both stages. A comprehensive example for illustrating our filtering methodology is also provided later. After the formation

of the neighborhood, the final step in the methodology is providing the prediction considering the degree of correlation of the neighbors and the ratings given for the items by these neighbors, specifically movies. We use a correlation based method which is the most popular prediction technique in collaborative filtering applications. It was originally used in the GroupLens project [32]. The basic idea is to compute a user's predicted rating of an item as a weighted average of the ratings given to that item by other users. Specifically, prediction $p_{u,x}$ for user u on item x is given by [16]:

$$p_{u,x} = \mu_u + \kappa \sum_{i \neq u} w_{u,x} v_{i,x} - \mu_u \tag{3.1}$$

where $w_{u,i}$: similarity between user u and i

$v_{i,x}$: user i's vote for item x

μ_u : user u's mean vote

$\bar{v}_a$: mean value of the votes for user a

κ : appropriate normalization factor

i : index that runs over selected neighbors in the system

3.1.1 Clustering Techniques

3.1.1.1 *K*-Means

K-means clustering algorithm is the simplest and the most popular clustering algorithm by far. This main-memory algorithm is based on a very simple idea. Given a set of initial clusters, assign each point to one of them, then each cluster center is replaced by the mean point of the respective cluster. These two simple steps are repeated until convergence. A point is assigned to the cluster which is the closest - according to a distance measure - to the point [25].

For a simple example, suppose we have five points as in Figure 3.1 and want to have k=2 clusters. Suppose we assign the points 1, 2, 3, 4 and 5 in that order for k=2. Then the points 1 and 2 are assigned to the cluster 1 and cluster 2 respectively, and become their centroids initially.

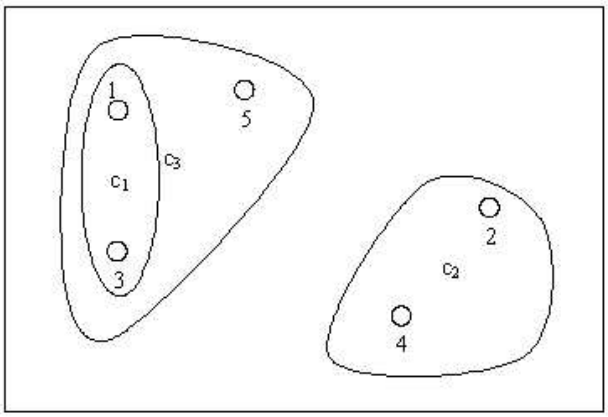

Figure 3.1: Example of a k-means clustering

Now we consider the points. Suppose point 3 is chosen and it is closer to point 1 than to point 2, so 3 goes to cluster 1. The centroid of that cluster changes to c_1. Now suppose we are to assign 4, and 4 is closer to point 2 than to c_1. So,

it is joined to the cluster 2 and the centroid of that cluster moves to c_2. The remaining point 5 is closer to c_1 than to c_2, therefore goes to cluster of c_1. The final situation is clusters $\{1, 3, 5\}$ and $\{2, 4\}$ with centroids c_3 and c_2, respectively.

The reasons for the popularity of this algorithm are [25]:

- It is easy to implement.
- Its time complexity is $O(nkm)$, where n is the number of points/items, k is the number of clusters and m is the number of iterations taken by the algorithm to converge. Typically, k and m are fixed in advance and so the algorithm has linear time complexity in the size of data set.
- Its space complexity is $O(n + k)$.
- It is order independent, i.e. for a given initial seed set of clusters, it generates the same partitioning of the items independent of the order in which the points/items are presented to the algorithm.

However, in the presence of a large data set, it can be slow since in each iteration the distance between each point to each cluster has to be calculated. Moreover, this algorithm is sensitive to the selection of the initial seed set. The number of clusters, k, should be given to the algorithm.

Several variants of the k-means algorithm have been described in the literature [2]. Some of these methods attempt to select a good initial partition so that the algorithm is more likely to find the global minimum value. Another variation is to permit splitting and merging of the resulting clusters. Typically, a cluster is split when its variance is above a specified threshold, and two clusters are merged

when the distance between their centroids is below another specified threshold. Using this variant, it is possible to obtain the optimal partition starting from any arbitrary initial partition, provided that proper threshold values are specified [5].

Our k-means implementation uses a correlation based method. In the simple k-means algorithm, a metric is required to calculate the distance between points, however it is not possible to provide a metric value for correlations between users in collaborative filtering. Therefore, we use a Pearson correlation method formulated in Equation 2.2 for calculating neighbors.

3.1.1.2 Cover Coefficient based Clustering

In our methodology, as an alternative to popular k-means algorithm, we use C^3M which was shown to have many desirable properties for a clustering algorithm with respect to other algorithms in the literature. This clustering method can be used in dynamic environments incrementally for cluster maintenance [10]. This method models the document collection in a vector space. Here a document collection is represented by a document matrix D, of size mxn, where m is the number of documents and n is the number of terms. An example D matrix is given in Figure 3.2.

In the example, the terms t_2, t_4 and t_6 appear in document d_1. In our domain, documents correspond to the users and terms correspond to the items, specifically movies. The number of clusters is determined by using the cover coefficient concept, CC [12]. For an m by n document matrix, this number, n_c and the

$$D = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{matrix}$$

Figure 3.2: Example of a D matrix

average cluster size d_c are specified as follows:

$$1 \leq n_c \leq min(m, n) \; ; \; max(1, m/n) \leq d_c \leq m. \tag{3.2}$$

C^{3}M maps the document matrix D to a cover coefficient matrix C of size m by m by using a two-stage probability calculation. This C matrix represents the relationships among the documents in the database. To illustrate this concept we summarize the calculation of c_{12}. According to the D matrix, d_1 contains three terms $\{t_2, t_4, t_6\}$. According to the two-stage probability model, to calculate c_{l2} we must first randomly select each one of the terms of d_1, and then try to select d_2, from the outcome (term) of the first stage. At the first stage, if we select t_2 or t_6, then d_2 has a chance of $\frac{1}{2}$. However, if t_4 is selected at the first stage, then the probability of selecting d_2 at the second stage is $\frac{1}{4}$. This is because t_2 and t_6 appear in d_1 and d_2. On the other hand, t_4 appears in d_1, d_2, $d_3$and d_5 . At the first stage, the probability of selecting each element of $\{t_2, t_4, t_6\}$ from d_1 is $\frac{1}{3}$, and for the rest the probability is 0 (i.e., no term in $\{t_1, t_3, t_5\}$ appears in d_1). If we denote the probability of selecting document d_i from term t_k at the first stage with p_{ik} and similarly denote the probability of selecting document d_j from term t_k at the second stage with p'_{jk}, then c_{12} can be given as follows 3.3:

$$c_{12} = \sum_{k=1}^{6} p_{1k} \times p'_{2k} = \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times \frac{1}{4} + 0 \times 0 + 0 \times \frac{1}{2} + \frac{1}{3} \times \frac{1}{2} + 0 \times 0 = 0.42 \quad (3.3)$$

Some of the documents are chosen as cluster seeds and other documents are assigned to the clusters initiated by the seed documents. For clustering process, C^{3}M does not need the entire C matrix. The diagonal entries of C matrix are used to find the number of clusters, n_c, and the cluster seeds. In order to assign a non-seed document d_i to a cluster, the relationship between d_i and the seed document d_j is determined by calculating the c_{ij} entry of C, which shows the extent with which d_i is covered by d_j. Hence, only $(m + (m - n_c) * n_c)$ entries of the total m^2 entries of C matrix are required. This is actually a small value compared to m^2, since $n_c \ll m$. A brief description of the algorithm is given in Figure 3.3. A thorough discussion and complexity analysis of C^{3}M are available in [12].

Algorithm 1: C^{3}M

1: Determine the cluster seeds of the database.
2: $i = 1$
3: **repeat**; /* construction of clusters */
4: **if** d_i is not a cluster seed **then**
5: Find the cluster seed (if any) that maximally covers d_i if there is more than one cluster seed that meets this condition, assign d_i to the cluster whose seed power value is the greatest among the candidates.
6: $i = i + 1$
7: **until** $i \geq m$
8: If there remain unclustered documents, group them into a ragbag cluster (some nonseed documents may not have any covering seed document).

Figure 3.3: C^{3}M [12]

The algorithm has been shown to satisfy some important characteristics. Clusters produced are stable, i.e., small errors in the description of the documents lead to small changes in the clustering. The algorithm is independent of the order of the documents and so generates a unique classification. Extra data structures needed for the implementation of the algorithms require a very small memory space. The algorithm distributes the documents evenly among the clusters, i.e., it does not generate a few fat clusters and many singletons. Also, it does not require n_c to be pre-specified, but obtains it inherently.

3.1.2 File Structures

3.1.2.1 Inverted File Search Algorithm

In the basic inverted indexing, for each query term, corresponding inverted list is scanned [40]. During this scan, accumulators are stored in main memory for holding partial sums of ratings for items. In our case, partial sums will be the partial similarities between the active user and the other users. After processing all inverted lists, weight arrays such as containing normalization factors are combined with the complete accumulators to produce the final score for a user. To apply inverted search to correlation, we use the partial accumulators method proposed by Coster et al. [14] and keep three different accumulator structures in-memory for Equation 2.2: one for the sum in the nominator, and two for the sums in the denominator. Figure 3.4 elucidates the algorithm of Coster et al. for neighborhood calculation of collaborative filtering with inverted indexing.

Algorithm 2: Inverted correlation neighborhood search

Input is the user profile v_a, max number of neighbors K and array of means $MEANS$. Output is array of nearest neighbors.

1: Allocate accumulators SAI, SAA and SII of size N.
2: Allocate array TOP of size K for nearest neighbors.
3: **for** all $j \in v_a$ **do**
4: locate inverted list L for title j
5: **for** each user u and vote v in L **do**
6: $SAI[u] = SAI[u] + (v_{a,j} - \bar{v_a}) * (v - MEANS[u])$
7: $SAA[u] = SAA[u] + (v_{a,j} - \bar{v_a})^2$
8: $SII[u] = SII[u] + (v - MEANS[u])^2$
9: **for** all $u \in SAI$, such that $SAI[u] \neq 0$ **do**
10: $corr = SAI[u]/\sqrt{(SAA[u] * SII[u])}$
11: **if** $corr \geq TOP[K-1]$ **then**
12: add $(u, corr)$ to TOP
13: restore TOP to sorted order
14: return TOP

Figure 3.4: Neighhborhood computation for collaborative filtering with inverted indexing [14]

Notice that, theoretically Coster et al. defines 3 different accumulators instead of a single one as in a typical information retrieval, to store required values for computing Pearson coefficient. The mean values of votes for each user are stored in the static array *MEANS*. N denotes the number of users in the database. The algorithm computes the top K neighbours to user a. The sorting step is performed by using min-heaps.

3.1.2.2 Cluster-Skipping Inverted Index Structure

One of our contributions to efficient collaborative filtering is the adaptation of cluster-skipping inverted index structure (CS-IIS) [11]. This approach provides

efficient and effective cluster based retrieval for large databases without considerable additional storage overhead, and improves scalability which is one of the important problems in collaborative filtering approaches. It significantly reduces the cost of similarity calculations. CS-IIS provides efficient access to the clustered documents based on the common terms with the queries.

In cluster based retrieval strategy, there are two components whose file structures crucially affect the efficiency of the system. These are selection of n_s number of best matching clusters using centroids, and selection of d_s number of best matching documents of the selected best matching clusters. One of the possible structures that can be used is inverted index for both centroids and documents, i.e., IC inverted index of centroids and IIS inverted index of all documents. In CS-IIS, Can et al. keep IC in its usual structure; however in IIS component they store not only posting list information but also cluster membership information. Posting list information associated with the members of a cluster is stored next to each other, and it is followed by the members of the next cluster. Pointers are used from the beginning of one cluster's posting list to the next one for skipping the clusters which are not selected as best matching clusters. An example is illustrated in Figure 3.5. In this example, the given D matrix is clustered using C^3M.

In Figure 3.5 each posting list header contains the associated term, the number of posting list elements associated with that term, and the posting list pointer (disk address). The posting list elements are of two types, "cluster number - position of the next cluster" and "document number - term frequency" for the documents of the corresponding clusters. If we assume that the user query contains the terms t_3, t_5 and the best-matching clusters for this query are $C1, C3$;

$$D = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 3 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 1 \\ 0 & 0 & 1 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 1 & 4 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{matrix} \end{array}$$

$C1 = \{d_1, d_2\}$
$C2 = \{d_3, d_4\}$
$C3 = \{d_5, d_6, d_7\}$

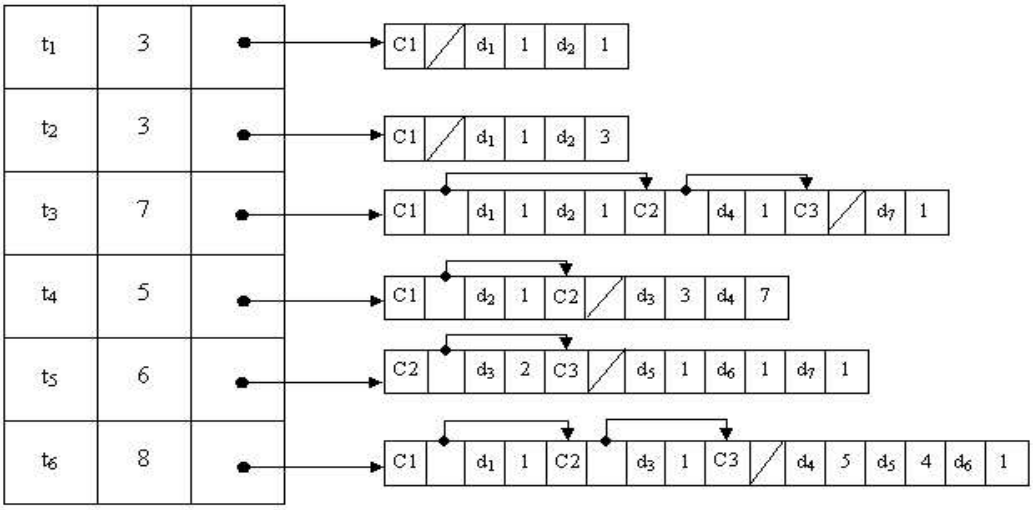

Figure 3.5: Inverted file structure with skips [11]

using the cluster-skipping IIS approach during query processing after selecting the best-matching clusters we only consider the posting lists associated with t_3 and t_5. While processing the posting list of t_3 we skip the portion corresponding to C2 (since it is not a best-matching cluster). Similarly, while processing the posting list of t_5; we again skip the unnecessary C2 portion of the posting list and only consider the part corresponding to C3. In other words, by using the skip approach we only handle the documents that we really need to match with the query.

The presented cluster based retrieval strategy has been shown to improve the efficiency of query processing via in-memory similarity calculations. For large databases, this strategy can achieve a time efficiency and effectiveness comparable with full search. This characteristic helps us significantly in improving the scalability of collaborative filters.

Example: We illustrate our methodology with an example. Suppose we have movie set as $M : \{M1, M2, M3, M4, M5\}$ and user set as $U : \{U1, U2, U3, U4, U5\}$. Our document matrix which contains the ratings of these users for these movies is as in Figure 3.6:

$$D = \begin{array}{c} \begin{matrix} M1 & M2 & M3 & M4 & M5 \end{matrix} \\ \left[\begin{array}{ccccc} 3 & 0 & 1 & 2 & 0 \\ 3 & 0 & 1 & 2 & 0 \\ 1 & 3 & 2 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 3 & 4 \end{array}\right] \end{array} \begin{matrix} U1 \\ U2 \\ U3 \\ U4 \\ U5 \end{matrix}$$

Figure 3.6: Document matrix of the example

Now suppose that these users are clustered and the clusters are as follows:

$C1 = \{U1, U2\}$, $C2 = \{U3, U4\}$, and $C3 = \{U5\}$. According to the profiles given in D matrix, we compute the centroids of the clusters by averaging the votes for each movie item by the corresponding users of each cluster, i.e.,

$$\begin{array}{lllllll} \text{Centroid-C1}: & \langle M1: & 3 & M3: & 1 & M4: & 2\rangle \\ \text{Centroid-C2}: & \langle M1: & 1.5 & M2: & 2 & M3: & 1.5\rangle \\ \text{Centroid-C3}: & \langle M2: & 1 & M4: & 3 & M5: & 4\rangle \end{array}$$

The corresponding inverted index structure for centroids can be given as in Figure 3.7. So, for a given user query $U_q : \{M1 : 4, M4 : 2\}$, assume that we set the number of best correlated clusters and the number of best correlated users to 1, i.e., we are to select the most correlated cluster considering the centroids, and then to select the best matching user from this best cluster. Applying Pearson correlation method as in Figure 3.4, we find the best cluster as C1.

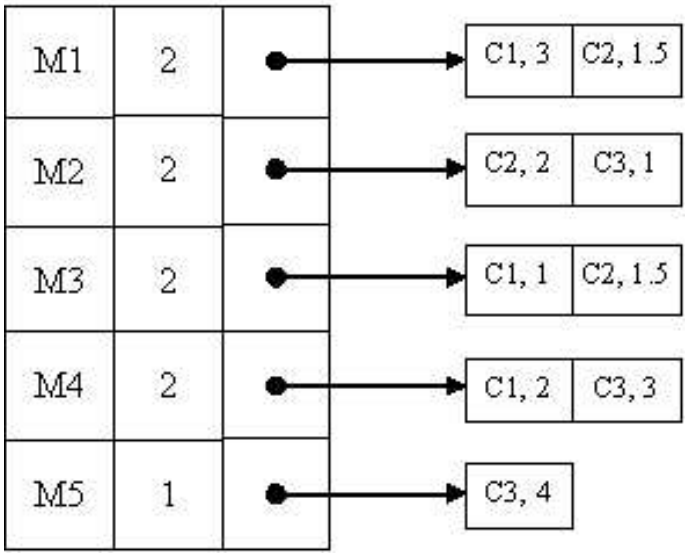

Figure 3.7: IC- Inverted centroids structure of the example

Now for selecting best matching users, two alternatives, using centroids or using all users in the best matching clusters exist. In order to find the best

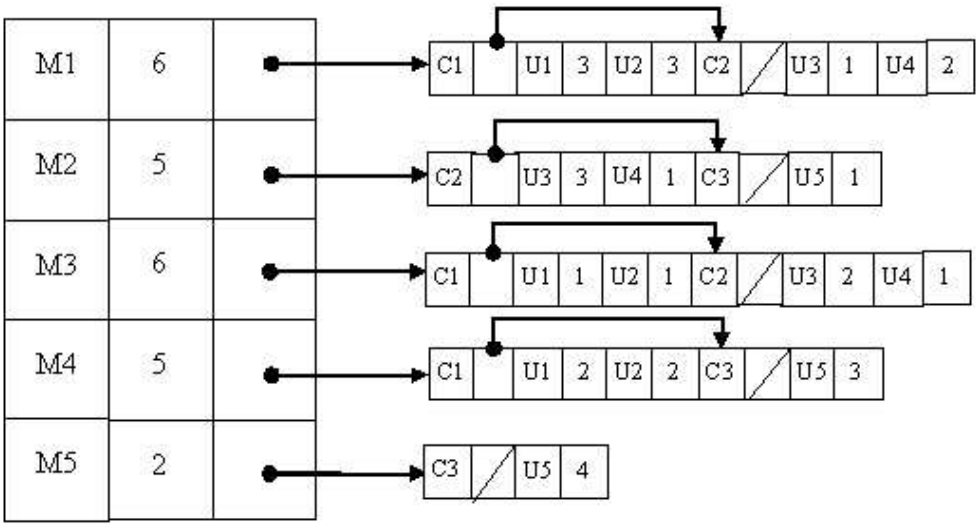

Figure 3.8: Cluster-skipping inverted index structure of the example

matching users considering all users in the best matching clusters, we use the cluster-skipping inverted index structure given in Figure 3.8 and again by applying Pearson correlation, we find the 1 best matching user to be U1. Then we can exploit the prediction method explained in Section 3.1 to give the predicted ratings.

3.2 Hybrid Filtering with Cluster-Skipping Inverted Index Structure

Hybrid recommender systems combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual recommender technique. Most commonly, collaborative filtering is combined with some other technique in an attempt to avoid some of the associated problems such as cold start[1]. One of the most common hybrid recommendation techniques is the filtering scheme which combines content based and collaborative filters together [9].

Content based filters only require one user, but have the weakness of not being able to predict interest on information that is significantly different from anything seen before. Collaborative filters require multiple users; they can handle new and unseen information items, but only as long as some other user has seen and rated the item. Thus, the goal of hybrid filters is to take the best features of each technique and minimize the impact of their weaknesses with the goal of outperforming each individually.

One of our approaches is to facilitate collaborative filtering without throwing away the content information available so that alleviating some of the problems each approach could possess. To accomplish this, we perform a repeated clustering technique [39]. Firstly, we generate the clusters according to the *genre* attribute of movies. In this stage, first the movies are partitioned into clusters according to their *genre* attribute. This produces clusters such as *Action*, *Drama*, *Romance*, etc. These clusters are overlapping, i.e., a movie may fall into several

[1] described in Chapter 2

clusters, such as both *Drama* and *Romance*. Then the users are clustered according to *genre* of the movies they watched. These clusters are non-overlapping, since a user falls into the cluster from which s/he has watched the largest number of movies. Then, the users of each cluster obtained in this stage are subject to another clustering using either C^3M or K-Means. For example, suppose we have two clusters, e.g. *Action* and *Drama*, obtained in the first partitioning stage. The users who have watched movies from *Action* cluster are clustered among themselves and similarly the users who have watched movies from *Drama* cluster are clustered among themselves. Then these clusters are presented to the rest of the methodology, i.e., either collaborative filtering with inverted indexing or collaborative filtering with cluster-skipping inverted index structure is applied. The experimental results given in Chapter 4 verify our expectations.

Chapter 4

Experiments and Results

Throughout the previous chapters, we have described existing techniques for handling information overload, discussed the problems associated with them and provided our own approaches for improvement. In this chapter, we try to evaluate the improvement gained via our approaches in comparison with the existing algorithms by conducting a large set of experiments. We also describe the developed prototype movie recommender system, MoRec, which is currently in public use[1] [24].

Advancing in two ways, separate hardware and software configurations have been employed for the experimental evaluation and for the development of MoRec. The hardware needed to configure a test environment for our research includes a machine, which operates on LINUX, with a memory of minimum 256 Mbytes RAM and a 2.4 GHz Intel Pentium-IV processor. The proposed and existing approaches were implemented in C programming language on a Linux platform.

[1] http://pcvideo.cs.bilkent.edu.tr/MoRec/index.aspx.

C code was compiled by *gcc*, the GNU Compiler Collection,[2] and for debugging purposes, *kdebugger*, KDbg a graphical user interface to the GNU debugger,[3] was utilized. The hardware needed to configure MoRec consists of a machine with at least 512 Mbytes RAM and a processor of at least 2 GHz since this machine is both a Web and a database server. It also needs an Internet connection so that the Web page requests can be satisfied. The software configuration of MoRec includes Windows XP Professional as the operating system, Internet Information Services (IIS) as the Web server and Microsoft Visual Studio .NET as the implementation platform. Storing, retrieving and interacting with the data are handled by Microsoft SQL Server 2000 Personal Edition. As a result, the user interface package of Microsoft Visual Studio .NET and the database management system Microsoft SQL Server 2000 Personal Edition are used as off-the-shelf components.

We use the EachMovie dataset as both the test-bed of our approaches and as the database of MoRec, collected by the Digital Equipment Corporation (DEC) Systems Research Center from 1995 through 1997. This dataset comprises a total of 2811983 ratings of 1628 movies by 60087 users. Each rating is on a scale of 0 to 5 [4] [15]. We have used a Web crawler, WebSPHINX (Website-Specific Processors for HTML INformation eXtraction) developed by Rob Miller at Carnegie Mellon University, in order to gather the content data of movies from The Internet Movie Database (IMDb) [23] for hybrid filtering approaches. Out of 1628 movies present in EachMovie dataset, the content information –including synopsis, genre, key actors and actresses– of 1482 movies could be downloaded. Out of these 1482 movies, 1224 movies have synopsis.

[2] http://gcc.gnu.org/

[3] KDbg is authored by Johannes Sixt.

[4] For more information see *http://www.research.compaq.com/SRC/eachmovie/*.

To evaluate our approaches, we have conducted a set of experiments. We have used a popular statistical accuracy metric, *mean absolue error* (MAE) to evaluate the accuracy and the elapsed time for neighborhood formations to evaluate the efficiency of our approaches. MAE calculates the absolute difference between the actual vote and the prediction, averaged over all predictions.

4.1 Implementation Details

The underlying testing system consists of 7 modules responsible for different stages of information filtering process:

1. *Preprocessing*: In this module, the data files provided by EachMovie dataset, for movies, votes and users, are modified to form two document vectors by eliminating the unnecessary parts in these files. One of these document vector files includes the train user data used to train the predictor. The other document vector file is for keeping the test user data. While generating these files, the usual matrix representation is not used, instead a sparse matrix representation of vectors is exploited. A simple example is given in Figure 4.1. The actual document vector contains 15 movies in a line instead of 6 as in the example.

2. *Inverted Index Structure - IIS*: In this module, the inverted index structure of train user data is generated and kept as a binary file, so that it can be accessed randomly, i.e., the votes of any user can be accessed without reading the file sequentially.

3. *Collaborative Filtering with IIS*: This module performs the prediction. We

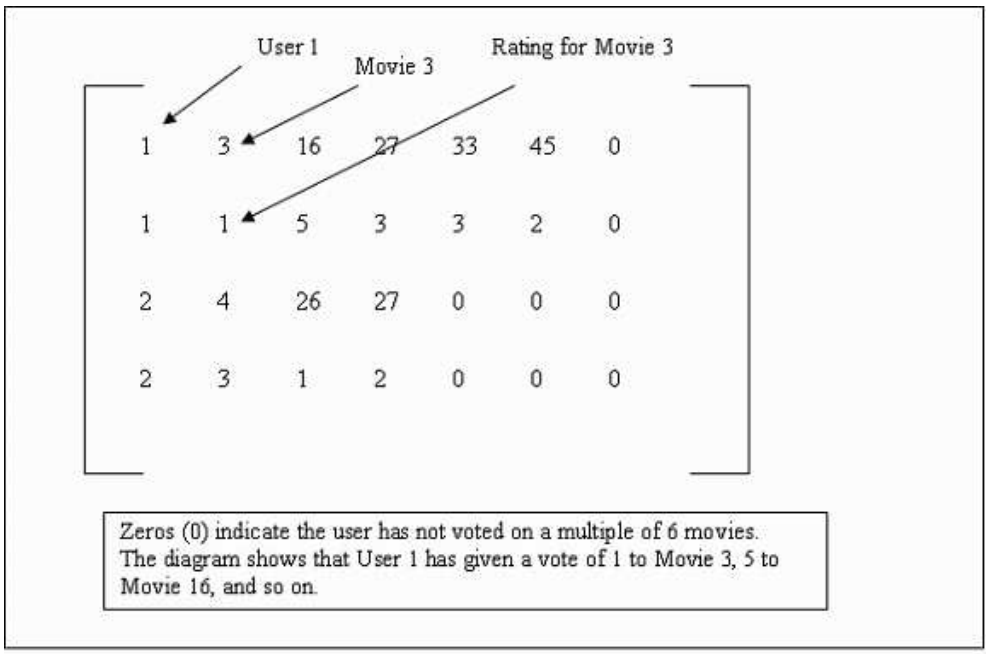

Figure 4.1: Document vector format

read in train and test user binary files. Then we generate a pre-defined number of predictions, usually 5, for each user and calculate the absolute difference between the prediction generated by the system and the actual vote given by the user. At last, we calculate the average of the absolute differences of each prediction to form Mean Absolute Error. We output the mean absolute error together with the time statistics.

4. *Clustering*: This module deals with the cluster formation of train users. It groups similar users into one cluster. There are two algorithms as explained in Chapter 3: C^3M and K-Means. Both of these algorithms take train user data and create two files, one that shows the users of each cluster and another that shows the cluster of each user.

5. *Centroid Generation*: In this module, we create the centroids, virtual users representing each cluster using the train user data and clustering files. We

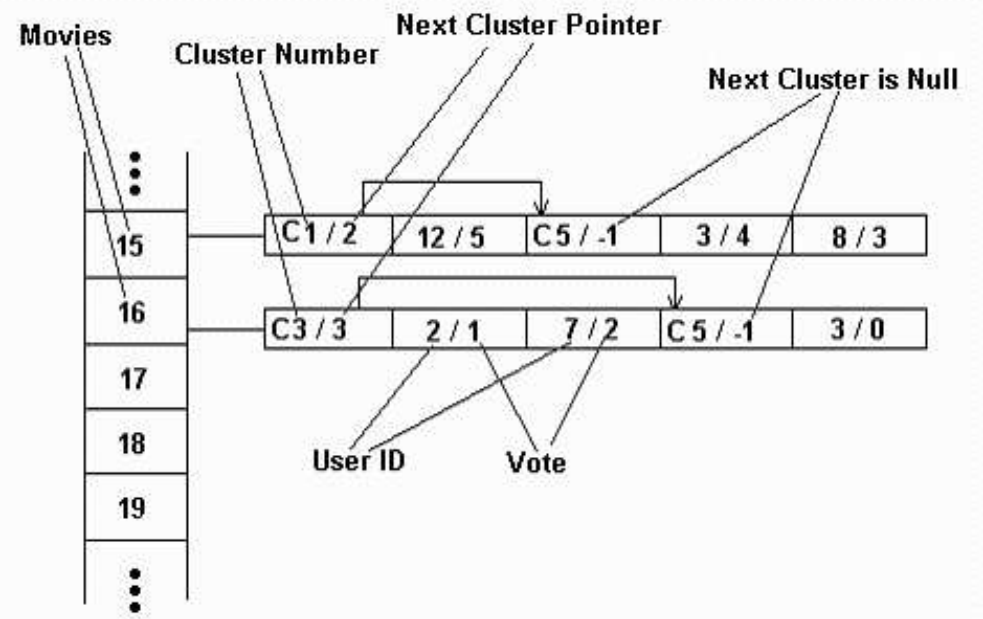

Figure 4.2: Inverted Index with skipping data format

generate a file, which holds the mean votes of these centroids. Another file that is created is a binary file, which contains the movies with the votes of each centroid as they are virtual users with votes on movies. The last output of this module is another file which shows a list of movies with the number of centroids that have votes on them. This file is used to determine the size of the data structures used in the last module, Cluster Based Collaborative Filtering with CS-IIS.

6. *Cluster-Skipping Inverted Index Structure*: This module reads in the inverted list of votes before clustering, as well as the clustering information. It outputs the inverted index with skipping data structure as exemplified in Figure 4.2.

7. *Cluster Based Collaborative Filtering with CS-IIS*: This module is a more developed version of Collaborative Filtering with IIS. Again, it reads each user's votes from a binary file when needed; however, unlike Collaborative

Filtering with IIS, it first selects a number of clusters, and then generates a prediction from train users of these clusters.

In Figure 4.3, an overview of these modules is given according to components' input and output relations.

4.2 Experimental Setting

We have used the largest available public dataset, EachMovie, for testing our algorithms. We compare our approaches –cluster based collaborative filtering with cluster-skipping inverted index structure and the hybrid approach– against collaborative filtering with inverted indexing proposed by [14].

4.3 Efficiency and Effectiveness

Several evaluation metrics for evaluating the accuracy of collaborative filtering approaches have appeared in the literature (see Chapter 2). The most common metric is the *Mean Absolute Error* (MAE). MAE is the average absolute deviation of the predicted ratings from the actual ratings on items the test users have voted. The lower the mean absolute error, the more accurate the scheme. We choose MAE in our experiments for two reasons: a) It is the most commonly used metric and allows us to compare our results with a larger set of previous works. b) There is a vast research literature on performing statistical significance testing and computing confidence intervals for MAE. Furthermore, Herlocker et

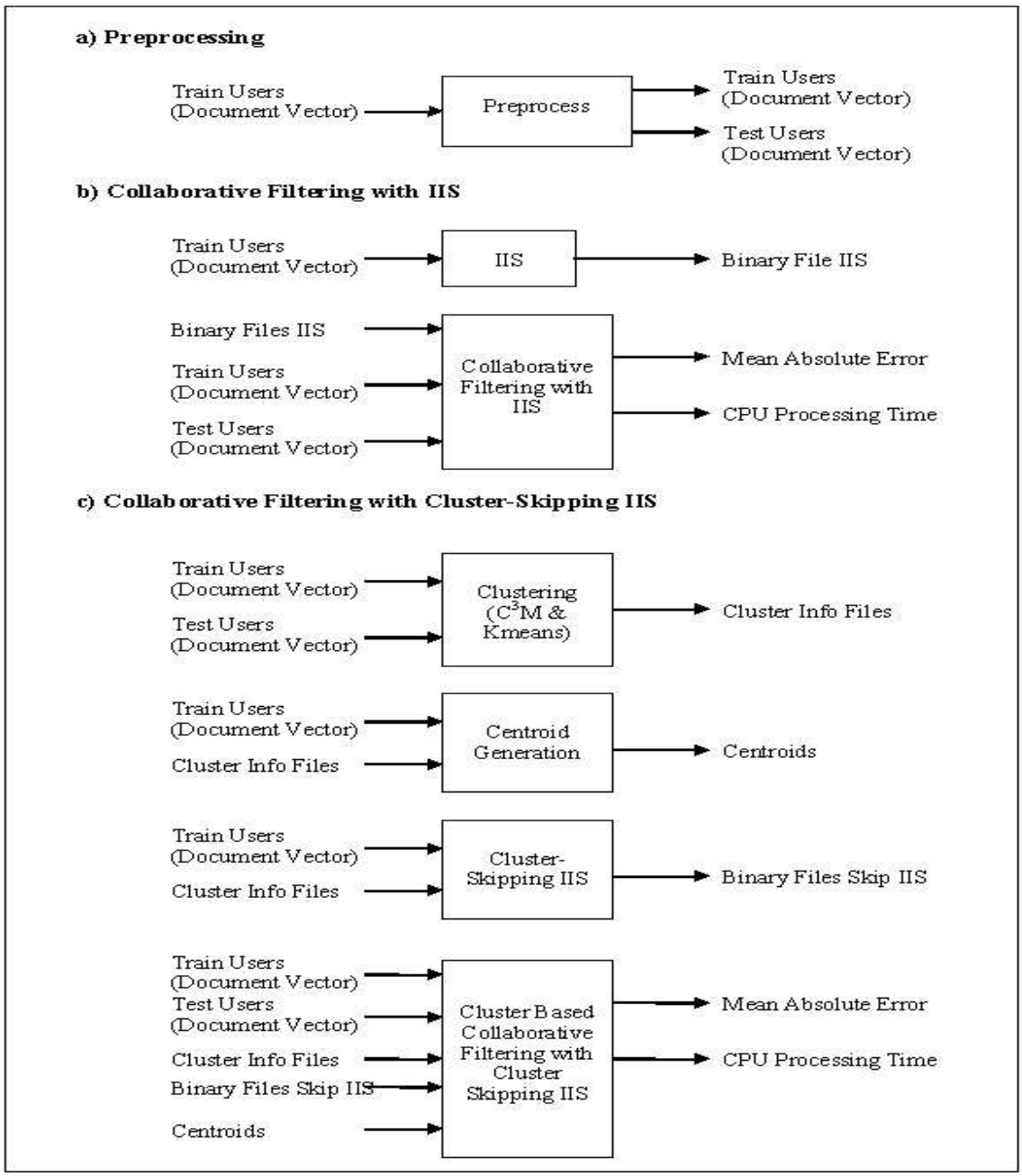

Figure 4.3: Modules of the System

al. [22] have also suggested the use of MAE for prediction evaluation. For evaluating the efficiency, we measure the neighborhood formation times for comparative purposes.

The MAE is calculated by summing the absolute errors of the corresponding rating-prediction pairs and then computing the average as given in Equation 4.1 [22].

$$MAE = \sum_{u=1}^{N} \frac{|P_{u,i} - R_{u,i}|}{N} \tag{4.1}$$

where $P_{u,i}$ means the prediction for user u on item i; $R_{u,i}$ means the actual rating of the user u on item i in the test data; N is the number of rating-prediction pairs between the test data and the prediction result.

We evaluated the algorithms on the *AllBut1* protocol, meaning that for each user we held out a single vote that should be predicted on the basis of all the other votes in the profile as Coster et al. [14]. We experimented with a training set size of 90% of all users, i.e., 55000 users. This choice is due to comparison with Coster et al. who used the inverted file search algorithms for collaborative filtering (see Chapter 2). We used a neighborhood size k of 30. We set the number of clusters that are used as the best correlated clusters from which k nearest neighbor is selected, to 10% of the total number of clusters. We set the number of movies for which a prediction is calculated for each test user, to 5. Different cluster sizes were experimented, 24, 200, and 1300, for both C^3M and K-means algorithms. 24 is due to C^3M, since the algorithm itself determines the cluster size if it is not predetermined by the user. Since we let C^3M select the number of clusters in each partial cluster for hybrid filtering approach, this stage generates a total of nearly 200 clusters for all first-stage clusters. Therefore, in order to compare the hybrid filtering approach with the collaborative filtering

one, we set the number of clusters to 200. For the last setting, we determined the number of clusters for each partial cluster to be 100 and this makes a total of 1300 clusters for hybrid filtering, since the first stage (see Chapter 3) in hybrid filtering produces 13 clusters of users according to the content information of movies. Again for comparative purposes we set the cluster size to 1300. Each experiment was performed 5 times. There are two strategies for determining the best matching neighbors: by using centroids of the best matching clusters and by considering all users of these clusters. For two filtering approaches, i.e., collaborative filtering and hybrid filtering, this makes a total of 105 experiments, including the experimentations of collaborative filtering with inverted indexing for comparative purposes with Coster et al. [14].

We measure the total elapsed time for neighborhood formations for 5 queries of all test users in seconds, and the mean nighborhood formation time (referred as Mean NF Time in Tables from 4.1 to 4.4) for one user query in milliseconds. The accuracy metric MAE is on a scale of 0 to 5. In tables, CF with IIS refers to collaborative filtering with inverted indexing proposed by Coster et al [14], CF with Cluster-Skipping IIS refers to cluster based collaborative filtering with cluster-skipping inverted index structure. C^3M and K-Means refer to the algorithms employed for clustering.

1. *Effectiveness*: We compare our results with the results of Coster et al. As discussed in Chapter 2, Coster et al. proposed using a disk based inverted file structure for collaborative filtering and used some early termination heuristics [14]. They compare their results with typical in-memory vector search (see Chapter 2) and show that their approach yields equivalent accuracy. Table 4.1 shows the results of the cluster based collaborative filtering with cluster-skipping

Collaborative filtering							
Algorithm	**CF with IIS**	**CF with Cluster Skipping-IIS**					
		C^{3}M			***K*-Means**		
Cluster size		**24**	**200**	**1300**	**24**	**200**	**1300**
Accuracy	0.91	0.92	0.91	0.90	0.92	0.91	0.90
NF Time	1324.57	651.01	593.09	815.15	531.22	883.02	1027.05
Mean NF Time	49.86	24.15	22.00	30.24	19.71	32.76	41.52

Table 4.1: CBR - Users : Collaborative filtering

Collaborative filtering							
Algorithm	**CF with IIS**	**CF with Cluster Skipping-IIS**					
		C^{3}M			***K*-Means**		
Cluster size		**24**	**200**	**1300**	**24**	**200**	**1300**
Accuracy	0.91	1.18	1.25	1.29	1.22	1.30	1.32
NF Time	1324.57	81.27	72.06	111.74	70.68	73.79	84.78
Mean NF Time	49.86	3.02	2.67	4.15	2.62	2.74	2.92

Table 4.2: CBR - Centroids : Collaborative filtering

Hybrid filtering					
Algorithm	**CF with IIS**	**CF with Cluster Skipping-IIS**			
		C^{3}M		***K*-Means**	
Cluster size		184	1300	184	1300
Accuracy	0.91	0.90	0.90	0.94	0.94
NF Time	1324.57	623.82	902.78	829.22	1747.47
Mean NF Time	49.86	23.33	33.53	30.80	64.90

Table 4.3: CBR - Users : Hybrid filtering

Hybrid filtering					
Algorithm	**CF with IIS**	**CF with Cluster Skipping-IIS**			
		C^{3}M		***K*-Means**	
Cluster size		184	1300	184	1300
Accuracy	0.91	1.18	1.29	1.32	1.31
NF Time	1324.57	73.13	74.58	73.81	84.52
Mean NF Time	49.86	2.74	2.77	2.74	2.91

Table 4.4: CBR - Centroids : Hybrid filtering

IIS for the case that considers all users of the best correlated clusters in determining the best correlated users. Table 4.2 shows the results of the alternative which considers only cluster representatives for selecting the best correlated users of a query user, i.e., CBR - centroids. Both tables prove that our approaches are as effective as CF with IIS.

Tables 4.3 and 4.4 show our results for hybrid filtering approach for both users and centroids cases using C^{3}M and K-Means clustering algorithms. Interestingly, the results are slightly better than CF with Cluster-Skipping IIS, while they are still as well as CF with IIS, in terms of predictive accuracy.

Experiments reveal that the C^{3}M produces comparable effectiveness with the K-Means algorithm. Notice that in Table 4.4, the MAE for C^{3}M with a cluster size of 184, is 1.18 whereas it is 1.32 for K-Means while the neighborhood formation time is still less. This difference is more significant in Table 4.3, i.e., CBR - users case of hybrid filtering approach. Besides, for larger cluster sizes, the accuracy is better.

1. *Efficiency*: The overall time needed for a prediction includes the disk access times and the in memory computation time. We evaluate the efficiency of our

approaches considering these two measurements.

We simulated a disk based environment since it is very hard to obtain reliable measurements of disk access times due to enhanced caching capabilities of modern operating systems. Values of the simulation parameters were determined considering the characteristics of a Seagate Cheetah ST37405LC disk (in March 2002) for which the average seek time and latency add up to 8.6 ms and the transfer time per sector of 512 bytes is 0.014 ms [21].

In our dataset, the mean number of votes by a user is 45.9. Our strategy requires two random disk accesses per query term, one for the centroid and another for user inverted lists. On the otherhand, collaborative filtering with inverted indexing requires one random disk access, only for the term inverted lists. So, we have an extra cost of disk access for inverted centroid index entries. This brings an overhead of approximately $46 * 8.6 = 395.6$ milliseconds per user query. In order to compensate the time due to extra random disk access per query term, we keep the inverted file for centroids in memory, which is not larger than 4 Mbytes in size. This is not much of a concern regarding today's main memory capacities. Also, this structure can be effectively buffered in main memory considering today's file caching capabilities. Furthermore, a recent work on this area eliminates disk accesses needed for centroid inverted index posting lists by embedding the centroid information to the document posting lists [1]. Hence, this extra I/O cost can be avoided.

The mean number of votes for a movie in our dataset is 1732.4. So, number of processed posting lists elements can be at most 1733*2 in the worst case, if all clusters include only one document, which is impractical. Sequential reads per query term bring an overhead of $N * 2 * 4 * 0.014/512$ milliseconds, where N is

the number of entries ($\langle int, int \rangle$ pairs) in a posting list, 4 is size of an integer in bytes, 0.014 is the block transfer time and 512 is the page size in bytes. For our computation, this causes $1733 * 2 * 4 * 0.014/512 \approx 0.4$ milliseconds, and so less than half millisecond extra I/O cost to our strategy.

Coster et al. show that they obtained a gain of 66% compared to in-memory vector searching. On the other hand, timing results for neighborhood formations in Tables 4.1 through 4.4 show that cluster based collaborative filtering with cluster-skipping IIS, either using C^3M or K-Means algorithms, runs faster than collaborative filtering with inverted indexing, with a gain of at least 50%. This gain is improved with CBR - centroids case, shown in Tables 4.3 and 4.4.

The experimental results can be briefly summarized as follows:

- Experiments show that collaborative filtering with user clustering is as effective as pure collaborative filtering with inverted indexing, and both of the clustering algorithms, namely K-Means and C^3M, yield almost the same accuracy figures.

- In terms of neighborhood computation time in the main memory, collaborative filtering with cluster-skipping IIS takes far more shorter time (in some cases, almost 40%) than collaborative filtering with a typical IIS, since the number of comparisons is significantly reduced by the use of cluster-skipping index structure.

- Collaborative filtering with cluster-skipping IIS makes the same number of disk accesses (given that the inverted list of centroids, which is quite smaller than the cluster-skipping IIS, is kept in the main memory) as collaborative filtering with IIS, and it has only slightly larger sequential access times,

which is easily compensated by very fast sequential read time features of today's hard disks.

- Our results reveal that, computing the neighborhoods by inspecting the actual users that fall into the best clusters yields higher prediction accuracy in comparison to using only the centroids of these best clusters (i.e., as a gigantic "virtual" user) for neighborhood formation. This result also justifies the need for using cluster-skipping IIS file structure for the cluster-based collaborative filtering task.

- Interestingly, the hybrid filtering approach does not improve the prediction accuracy as it might be expected (e.g., see [39]). We attribute this to the fact that in these experiments, the only content attribute used is the "genre" for the initial clustering stage. Thus, employing other attributes such as plot, artists etc. can further improve accuracy. This will be considered in our future works.

4.4 MoRec: MOvie RECommendation System

The web application that rises as an immediate application of the proposed approaches is a web service called Movie Recommendation System, MoRec for short [24]. The underlying software is developed in C# programming language. The required software for MoRec consists of IIS Web Server, Microsoft SQL Server Personal Edition, and Microsoft Visual Studio .NET Framework. The WAP site is also operable on the same platform with the same software components.

MoRec facilitates various functionalities dedicated to movie lovers (Figure 4.4). Users may view all movies in the database (listed according to their initials), view top rated movies (calculated as taking the average scores by the votes of system users), search for movies, vote on movies, view plots and images of movies, comment on movies and read comments of movies. All these operations are provided as a web page. The process of generating predictions is based on the idea of cluster based collaborative filtering with inverted indexing, with the similarity measure algorithm Pearson put into practice. As an extension to the predicting process, the system also recommends a list of movies, specific to each user. The idea behind neighborhood generation is exploited during this procedure. The movies that the neighbor users have voted on are of great importance for this recommendation list to be satisfactory. All voting, prediction and recommendation activities are provided by the server operating behind the web page. The server enables a quick and client-server oriented type of service, satisfactory in terms of performance.

Figure 4.4: Index page of MoRec system

MoRec is a system where users need to create an account and sign in each time they want to use it. MoRec is a web service, so no installation is required. The following figures provide screenshots from the existing system.

Figure 4.5: Sign-up page of MoRec system

To create a new account, the sign-up form displayed in Figure 4.5 should be completed. Having successfully completed the signing up/in process, the system directs you to your homepage such as the one displayed in Figure 4.6.

Figure 4.6: A user homepage in MoRec system

A homepage provides access to current movies. As seen from the figure, the user can vote on the movies in the resulting list, view the synopsis and sometimes a picture of them or request prediction on them.

Figure 4.7: Prediction on a movie in MoRec system

The user can ask for a prediction on a specific movie according to his/her profile and the other users' profile in the system. The prediction scores vary from 0 to 5, each range having an identifying picture and sentence about the prediction result. Figure 4.7 illustrates a prediction for a movie on MoRec. Also the user can provide feedback on the prediction whether the system was accurate enough in its prediction process.

Figure 4.8: Recommendation list for a user in MoRec system

Recommending user a list of movies is another facility provided by MoRec. A list of movies recommended to the sample user is given in Figure 4.8.

Chapter 5

Conclusion

We have investigated the problem of information overload and proposed a new approach, so-called cluster based collaborative filtering technique with cluster-skipping inverted index structure. The application domain we have considered for our work is movie recommender systems. Collaborative recommender systems depend on overlap in ratings across users. These systems work best for a user who fits into a niche with many neighbors of similar taste. Two different clustering algorithms, namely C^3M and K-Means, have been employed in our system. For further improvement in performance, a hybrid approach that involves a two-stage clustering has been developed. The first stage of this approach exploits the content information available in movies, and clusters the users according to the similarities of the genre attribute of the movies rated by these users. The second stage involves another clustering of the generated initial clusters according to the similarity of the user profiles. This two-stage clustering reduces the effect of the new item problem[1] inherent to the collaborative filtering techniques. Our strategy

[1]Described in Chapter 2

also improves the scalability of the collaborative filtering based recommender systems by the adapted cluster-skipping inverted index structure.

We have evaluated our approach on the largest publicly available dataset on movie recommendation, EachMovie [15] and observed that all the algorithms have the same effectiveness as the in memory search technique, whereas skipped inverted index structure based approach improves the efficiency reported by a typical inverted index structure in [14]. Interestingly, the effectiveness of clustering for collaborative filtering and hybrid approaches do not yield very significant differences in the accuracy; whereas they are shown to be efficient for nonclustered cases. Nevertheless, our approach proves itself as a worthwhile technique as it reduces processing times considerably with almost no adverse effect on the predictive accuracy.

Future work will employ a more sophisticated version of cluster-skipping inverted index structure proposed recently, to reduce the extra I/O cost. Besides, to increase the storage efficiency state of the art data compression techniques will be explored. Finally, aiming to provide a more efficient recommendation technique, we will investigate adaptation of update handling methods for cluster-skipping inverted file structures as a future work.

Bibliography

[1] I. S. Altıngövde, F. Can, E. Demir, and Ö. Ulusoy. Incremental cluster-based retrieval with embedded centroids using compressed cluster-skipping inverted files. Submitted to a journal, 2005.

[2] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, Inc., New York, NY, USA, 1973.

[3] M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[4] M. Balabanovic, Y. Shoham, and Y. Yun. An adaptive agent for automated web browsing. Technical Report FCS TN 9752, Stanford Research Institute Technical Report, Stanford, CA, USA, 1997.

[5] G. H. Ball and D. J. Hall. Isodata, a novel method of data analysis and classification. Technical Report NTIS AD 699616, Stanford Research Institute Technical Report, Stanford, CA, USA, 1965.

[6] D. Billsus and M. Pazzani. A personal news agent that talks, learns and explains. In *Proceedings of the Third International Conference on Autonomous Agents*, May 1–5 1999.

[7] D. Billsus and M. J. Pazzani. A hybrid user model for news story classification. In *Proceedings of the seventh international conference on User modeling*, pages 99–108, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.

[8] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithm for collaborative filtering. Technical Report MSR-TR98-12, Microsoft Technical Report, 1998.

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

[10] F. Can. Incremental clustering for dynamic information processing. *ACM Transactions on Information Systems*, 11(2):143–164, 1993.

[11] F. Can, I. S. Altıngövde, and E. Demir. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems*, 29(8):697–717, 2004.

[12] F. Can and E. A. Özkarahan. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Transactions on Database Systems*, 15(4):483–517, 1990.

[13] Q. Chen and A. F. Norcio. Modeling a users domain knowledge with neural networks. *International Journal of Human-Computer Interaction*, 9(1):25–40, 1997.

[14] R. Cöster and M. Svensson. Inverted file search algorithms for collaborative filtering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 246–252, New York, NY, USA, 2002. ACM Press.

[15] EachMovie. Compaq Systems Research Center. `http://www.research.compaq.com`, August 2004.

[16] D. Fisher, K. Hildrum, J. I. Hong, M. Newman, M. Thomas, and R. Vuduc. Swami: A framework for collaborative filtering algorithm development and evaluation. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 366–368, New York, NY, USA, 2000. ACM Press.

[17] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[18] Google. `http://www.google.com`, June 2005.

[19] D. Gupta, M. Digiovanni, H. Narita, and K. Goldberg. Jester 2.0: A new linear-time collaborative filtering algorithm applied to jokes. In *Proceedings of ACM-SIGIR Workshop on Recommender Systems: Algorithms and Evaluation*. ACM Press, 1999.

[20] D. B. Hauver. Flycasting: Using collaborative filtering to generate a play list for online radio. In *Proceedings of International Conference on Web Delivery of Music*, 2001.

[21] J. L. Hennessy and D. A. Patterson. *Computer Architecture : A Quantitative Approach*. Morgan Kaufmann, second edition, 2002.

[22] J. L. Herlocker, J. A. Konstan, A. Borchers, and 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, pages 230–237, New York, NY, USA, 1999. ACM Press.

[23] IMDb. `http://www.imdb.com`, June 2005.

[24] T. Ínce, Ş. Hançerli, and H. H. Arı. MoRec - Movie Recommendation System. Senior Project, Department of Computer Engineering, Bilkent University. `http://pcvideo.cs.bilkent.edu.tr/MoRec/index.aspx`, May 2005.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Survey*, 31(3):264–323, 1999.

[26] A. Jennings and H. Higuchi. A user model neural network for a personal news service. *User Modeling and User Adapted Interaction*, 3(1):1–25, 1993.

[27] T. Joachims, D. Freitag, and T. Mitchel. Webwatcher: A tour guide for the world wide web. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 770–775, August 1997.

[28] K. Lang. Newsweeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339, San Mateo, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[29] S. Mcnee, I. Albert, D. Cosley, P. Gopalkrishnan, A. M. Rashid, J. A. Konstan, and J. Riedl. On the recommending of citations for research papers. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, 2002.

[30] Movielens. `http://movielens.umn.edu`, December 1999.

[31] M. O'Connor and J. Herlocker. Clustering items for collaborative filtering. In *Proceedings of Recommender Systems Workshop at 1999 Conference on Research and Development in Information Retrieval*, August 1999.

[32] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pages 175–186, New York, NY, USA, 1994. ACM Press.

[33] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[34] J. B. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *Proceedings of the First ACM Conference on Electronic Commerce*, pages 158–166, 1999.

[35] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260, New York, NY, USA, 2002. ACM Press.

[36] R. U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95)*, pages 210–217. ACM Press, May 1995.

[37] L. Si and R. Jin. Flexible mixture model for collaborative filtering. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

[38] A. Tombros, R. Villa, and C. J. van Rijsbergen. The effectiveness of query-specific hierarchic clustering in information retrieval. *Information Processing and Management*, 38(4):559–582, 2002.

[39] L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*, Menlo Park, CA, USA, 1998. AAAI Press.

[40] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishing, 1999.

[41] K. Wittenburg, D. Das, W. Hill, and L. Stead. Group asynchronous browsing on the world wide web. In *Proceedings of Fourth International World Wide Web Conference*, pages 51–62, 1995.

[42] Yahoo. `http://www.yahoo.com`, June 2005.