Association Rule Mining for Collaborative Recommender Systems

by

Weiyang Lin

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2000

APPROVED:

Professor Carolina Ruiz, Thesis Advisor

Professor Sergio A. Alvarez, Thesis Advisor

Professor Lee Becker, Thesis Reader

Professor Micha Hofri, Head of Department

Abstract

This thesis provides a novel approach to using data mining for e-commerce. The focus of our work is to apply association rule mining to collaborative recommender systems, which recommend articles to a user on the basis of other users' ratings for these articles as well as the similarities between this user's and other users' tastes. In this work, we propose a new algorithm for association rule mining specially tailored for use in collaborative recommendation. We make recommendations by exploring associations between users, associations between articles, and a combination of the two. We experimentally evaluated our approach on real data for many different parameter settings and compared its performance with that of other approaches under similar experimental conditions. Through our analysis and experiments, we have found that association rules are quite appropriate for collaborative recommendation domains and that they can achieve a performance that is comparable to current state of the art in recommender systems research.

Acknowledgments

I would like to express my gratitude to my thesis advisors, Professor Carolina Ruiz and Professor Sergio A. Alvarez, who have been helping me go through all the research and writings of this thesis. Without their advice and efforts, I would have not been able to complete this thesis. Their seriousness to research has impressed me a lot.

My thanks are also due to my reader, Professor Lee Becker, who has taken his time to read my thesis in his busy days, and Professor David Brown, who allowed me to use his PC in the AI lab exclusively which enabled me to do lots of experiments. Thanks also to the faculty members of our department, who have given me guidance and financial support during the two years of my study here.

I also send my gratitude to lots of my friends here, Lili Chen, Hong Su, Xin Zhang, Marton E. Balazs, Janet Burge and John Shut..., who have accompanied me and helped me in these two years, and made me feel Worcester as my family.

Special thanks to my husband, my parents and my sister, who have given me their love and have been always encouraging me and supporting me through all the hard times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Recommender systems try to recommend articles of potential interest to a user with respect to the user's individual preferences. Such recommender systems are the focus of current interest in part because of their importance for electronic commerce. Currently, online recommendation services span the areas of book, music, movie, web page and restaurant recommendations, demonstrating the wide range of application domains of existing recommender systems.

Almost all recommender systems fall into two categories: content-based recommendation and collaborative recommendation. As mentioned in [BS97], content-based recommendation tries to recommend articles similar to those articles the user has liked, whereas collaborative recommendation tries to find some users who share similar tastes with the given user and recommends articles they like to that user. Content-based recommendation and collaborative recommendation both have their own advantages and drawbacks. But collaborative recommendation is more popular than content-based recommendation, mainly because in many domains (such as music, restaurants) it is hard to extract useful features from articles, which is generally a step required for content-based recommendation.

Although there seems to be an increasing commercial demand for collaborative recommendation techniques, both the number of available published techniques and the information about their performance are quite limited. Hence it is of great importance to explore more techniques for this domain. Through our analysis and experiments, we have found that association rules are quite appropriate for this task and they can achieve good performance.

Association rule mining was proposed in [HHC66, HH77] and later in [AIS93]. Given a set of transactions, where each transaction is a set of items, an association rule is a rule of the form $X \Rightarrow Y$, where X and Y are sets of items (also called itemsets). The meaning of this rule is that the presence of X in a transaction implies the presence of Y in the same transaction. An example of an association rule in the basket market analysis domain is: "90% of transactions that contain bread and butter also contain milk; 30% of all transactions contain the three of them". Here, $X = \{\text{bread, butter}\}$, $Y = \{\text{milk}\}$, 90% is called the confidence of the rule, and 30% the support of the rule. The confidence of a rule measures the degree of the correlation between itemsets, while the support of a rule measures the significance of the correlation between itemsets. The problem of mining association rules is to find all association rules that are above the user-specified minimum support and minimum confidence.

The original motivation of association rules was to take advantage of massive amounts of sales data, which stores items purchased on a per-transaction basis. It aims to discover all significant associations between items. Some later work extended the basic association rules to classification domains[SA96, LHM98]. Association rules have also been widely used in web mining. They have been used to mine path traversal patterns and to facilitate the best design and organization of web pages [CMS97, CSM97, CPY98] Recently, Fu, Budzik and Hammond[FBH00] developed a good framework to mine association rules in navigation history for recommending web pages. Actually, we developed our ideas independently. But they use the original Apriori algorithm to mine rules. As we will discuss later, the traditional association rule mining algorithms are not good enough for recommender systems.

Our motivation to mine association rules for recommender systems comes from the following observation: Rules like "90% of users who like article A and article B also like article C, 30% of all users like all of them" and "90% of articles liked by user A and user B are also liked by user C, 30% of all articles are liked by the three of them" are very useful for recommendation purposes. We refer to rules of the first kind as article associations and rules of the second kind as user associations in this thesis. Article associations represent relationships among articles and user associations represent relationships among users. We can also explore article associations and user associations on two levels (*like* and *dislike*) by using extensions of the basic association rules. One example of two level user associations is "90% of articles liked by user A and disliked by user B are liked by user C, 30% of all articles are liked by user A and C and disliked by user B".

The goal of this thesis is to explore approaches of applying association rules for collaborative recommender systems and to evaluate the performance of those approaches. Towards this end, we have designed and implemented an adaptive-support algorithm to mine association rules which is particularly tailored to collaborative recommender systems. We also implemented the process of making recommendations by using user associations, article associations and a combination of the two. We employed the cross-validation approach to evaluate their performance with various parameter settings. Foremost among such parameters are the minimum confidence and minimum support values used for mining rules. We used the EachMovie [McJ97] dataset as our test-bed, which is provided by Digital Equipment Research Center and publicly available on the web. Our results are analyzed and compared with those of other systems. Our code is written in C++.

1.2 Our Approach

As we have mentioned above, we propose user associations and article associations as useful for collaborative recommendations. Also, we explore associations on one level (*like*) and two levels (*like* and *dislike*). Considering these two factors, we have several options. For each option, there are two problems that need to be addressed:

- How to map users' ratings for articles into "transactions"?

  Association rules are mined from a set of "transactions". For collaborative recommendation, we usually have some users' ratings for articles. How to convert ratings to "transactions" is determined by which kind of associations and how many levels of associations we want to discover.

- How to make recommendations by association rules?

  Once that we have mined the associations from the transactions, we need to determine how to use the association rules to produce good recommendations.

1.2.1 Mining Association Rules for Collaborative Recommender Systems

Association rule mining is the basic part of our work. Our adaptive-support algorithm to mine Association Rules for Collaborative Recommender Systems (AR-

CRS) is adapted from the Apriori algorithm [AS94a] and the CBA-RG algorithm [LHM98]. The adaptation is necessary for two main reasons:

1. In order to make recommendations to each user, we need to mine an appropriate number of rules for each user or article. This is in some degree dependent on the minimum support and minimum confidence specified for the mining process. If the minimum support and minimum confidence are too high, we can not obtain enough rules for recommendation, if they are too low, we will be in danger of having an unacceptable long period of runtime. We could specify a minimum confidence, but because users' tastes and articles' popularities vary widely, the appropriate minimum support for each user/article must be chosen independently. To the best of our knowledge, previous algorithms do not provide a mechanism to choose a proper minimum support for the given minimum confidence and the desired range for the number of rules.

2. Since we are only interested in predicting articles that a given user would like, we only need rules with one target item in the head. For example, for a user $user_{target}$ with user associations, we only need rules with [$user_{target}$: *like*] in the head. Such rules could be mined more efficiently than the rules with several targets. Considering sometimes we need to mine rules online, the efficiency is of great importance.

1.2.2 User Associations

In order to obtain *like* associations among users, we have each user correspond to an "item" and each article rated by users correspond to a "transaction". If a user likes an article, then the transaction corresponding to the article contains that user; otherwise, the corresponding transaction does not contain the user. From here, we can mine *like* associations among users, for example, "90% of articles liked by user A and user B are also liked by user C, 30% of all articles are liked by all of them".

In order to mine *like* and *dislike* associations among users, we employed the idea of mining categorical association rules (see Section 2.1.3) by extending each user, say $user_k$, to two "items", one item corresponding to [$user_k$: like] and another item corresponding to [$user_k$: dislike]. From here, we could mine rules like "90% of articles liked by user A and disliked by user B are liked by user C, 30% of all articles are liked by user A and C and disliked by user B".

During our tests, for each given target user, we employ the *4-fold* cross-validation approach. First, we randomly divide all the articles this user has rated into 4 groups. Then we run four rounds of test, each time choosing one group of articles as test data and the other three groups as training data. So every article this user rated will be used as a test article once.

With user associations, firstly we mine the association rules for the given user on the fly by using his/her training data together with the chosen collaborative users' rating data, and then for each of his/her test articles, we decide if we should recommend this article according to the rules we have just mined and the ratings on this article from the collaborative users. The strategy is described in greater detail in Section 4.2.

1.2.3 Article Associations

Similar to the user associations, in order to mine *like* associations among articles, we have each article correspond to an "item" and each user correspond to a "transaction". If a user likes an article, then the transaction corresponding to the user contains the article; otherwise, the corresponding transaction does not contain the article.

During the training process, we use the ratings from collaborative users to mine rules for each article, and store the rules in files. So the mining process can be done offline. During the test process, we also use the *4-fold* cross-validation approach to run four tests for each user. For each test article of a given user, we read in the rules for that article and decide if we should recommend this article according to whether some rules are fired by the training articles. The strategy is described in greater detail in Section 4.2.

1.2.4 Properties of Our Approach

After analyzing the characteristics of association rules, we found that our approach has the following properties.

1. The confidence of a rule measures the degree of the correlation among users or among articles, while the support of a rule measures the significance of the correlation among users or among articles. So we have two useful measures to evaluate the correlation indicated by a rule.

2. Instead of identifying some similar taste users for a given user, we use the overlap of some users' tastes to match one given user's taste, which could in some sense solve the problem of finding similar users for users with unique tastes. For example, the rule "90% of articles liked by user A and user B are also liked by user C" uses the overlap of user A's and user B's tastes to match user C's taste.

3. Our approach can achieve real-time recommendation. Although the traditional association rule mining process is computationally expensive, the framework we designed can mine the appropriate association rules in real time.

1.3 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2 introduces some necessary background knowledge. This includes the concepts of the basic and extended association rules as well as the algorithms to mine rules. It also discusses the current collaborative recommendation techniques.

Chapter 3 describes our new adaptive-support algorithm for mining association rules for recommender systems and some implementation details.

Chapter 4 discusses our approaches to make recommendations via association rules: user associations and article associations.

Chapter 5 shows the results of our experimental tuning and evaluations of user associations, article associations, and a combination of the two.

Chapter 6 summarizes our findings and discusses possible future work.

Chapter 2

Background and Related Work

This chapter describes basic and extended association rules as well as algorithms to mine association rules. It also discusses current collaborative recommendation techniques.

2.1 Association Rules

As mentioned in the introduction, association rules were independently introduced by P. Hajek, et al. [HHC66, HH77] and by Agrawal, et al. [AIS93]. While [HHC66] and [HH77] introduce association rule mining as a machine learning approach to the logic of discovery, [AIS93] concentrates on the mining of associations over sales data. One of the original motivations of the latter approach is to help supermarket managers to analyze past transaction data and to improve their future business decisions including catalog design, store layout design, coupon design and so on. Given a large database of customer transactions with each transaction storing items purchased by a customer during a visit, association rule mining aims to discover all significant associations between items in the database. A large variety of association rule mining algorithms have been published in the literature. Apriori [AS94a] and

DIS [BMUT97] are two of them. One extension of the basic binary association rules, called categorical association rules, [SA96] finds associations between attributes with categorical values. An example is: "[Profession: Professor] and [Married: Yes] $\Rightarrow$ [NumCars: 2]". Categorical association rules have the potential to extend association rules to general classification domains. Some results of adapting those rules to classification tasks are shown in [LHM98, HH77]. [LHM98] presents the CBA-RG algorithm (which is based on the Apriori algorithm) and a good framework to perform the so-called associative classification. Our adaptive-support algorithm to mine association rules for recommender systems is adapted from the Apriori algorithm and the CBA-RG algorithm.

2.1.1 Problem Description and Decomposition

Definitions

As mentioned in the introduction, given a set of transactions, where each transaction is a set of items, an association rule is a rule of the form X $\Rightarrow$ Y, where X and Y are sets of items. The meaning of this rule is that the presence of X in a transaction implies the presence of Y in the same transaction. X and Y are respectively called the *body* and the *head* of the rule. Each rule has two measures: confidence and support. The confidence of the rule is the percentage of transactions that contain Y among transactions that contain X; The support of the rule is the percentage of transactions that contain both X and Y among all transactions in the input data set. In other words, the confidence of a rule measures the degree of the correlation between itemsets, while the support of a rule measures the significance of the correlation between itemsets.

To consider an example, assume we have a database of transactions as listed in

Table 2.1, for association rule "{A} $\Rightarrow$ {C}", the confidence of the rule is 66%, and the support of the rule is 50%.

Transaction Id	Purchased Items
1	{A, B, C}
2	{A, D}
3	{A, C}
4	{B, E, F}

Table 2.1: Sample Transactions

There could be any number of items present in the body and in the head of a rule. A user could also specify some rule constraints, for example, he/she might only be interested in finding rules containing certain items.

Problem Description

The traditional association rule mining problem definition is: given a set of transactions, where each transaction is a set of items, and a user-specified minimum support and minimum confidence, the problem of mining association rules is to find all association rules that are above the user-specified minimum support and minimum confidence.

Traditional Problem Decomposition

We call a set of items an itemset. The support of an itemset is the percentage of transactions that contain this itemset among all transactions. An itemset is *frequent* if its support is greater than the user-specified minimum support. The problem of discovering association rules could be decomposed into two subproblems:

1. Find all frequent itemsets.

2. Generate association rules from frequent itemsets: for example, if {a, b, c, d} and {a, b} are frequent itemsets, then compute the ratio:

$$confidence = \frac{support_{\{a,b,c,d\}}}{support_{\{a,b\}}}$$

If *confidence* is not less than the user-specified minimum confidence, then "{a, b} ⇒ {c, d}" is one desired association rule. This rule satisfies the minimum support constraint because {a, b, c, d} is a frequent itemset.

The second part of the process described above is relatively straightforward. [AS94b] provides a fast algorithm for this process. But the process of discovering frequent itemsets is computationally expensive, usually requiring multiple passes over the whole database. The Apriori algorithm [AS94a] is an algorithm that is widely used for this task.

2.1.2 Algorithm Apriori

The Apriori algorithm generates frequent itemsets by making multiple passes over the transaction data. We use k-itemsets to denote itemsets of size k. The first pass finds the frequent 1-itemsets. For pass $k > 1$, it generates the candidate frequent k-itemsets using the frequent (k-1)-itemsets; then it scans all transactions to count the actual supports of the candidate k-itemsets; at the end of pass k, it collects the candidate k-itemsets whose supports are above the minimum support as the frequent k-itemsets.

Candidate Generation

Candidate k-itemsets are generated by performing two operations on frequent (k-1)-itemsets: join and prune. In the first join step, two different frequent (k-1)-itemsets which share the first k-2 items are joined together to generate a candidate frequent k-itemset. In the next prune step, we delete all candidate k-itemsets which have non-frequent (k-1)-subset. These two steps are correct due to the fact that any subset of a frequent itemset must also be frequent.

2.1.3 Categorical Association Rules

As mentioned in [SA96, LHM98], association rules can be easily extended so that they can express mine associations among categorical attributes, which can take a number of various values.

A transaction data set can be seen as a relational database table with boolean valued attributes that correspond to items and records that correspond to transactions. The value of an attribute for a given record is "1" if the corresponding item is present in the corresponding transaction, "0" otherwise. For example, the transactions contained in Table 2.1 can be expressed as Table 2.2. So the basic association rules can be viewed as finding associations between the "1" values of those boolean attributes.

Transaction Id	A	B	C	D	E	F
1	1	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	1	0	0	0
4	0	1	0	0	1	1

Table 2.2: Sample Transactions with Boolean Valued Attributes

For categorical attributes, instead of having just one field in the table for each attribute, we want to have as many fields as the number of attribute values so that we can mine associations among those values. For example, if we convert Table 2.3 to Table 2.4, we can perform the basic association rule algorithm to mine rules like

"[Profession: Professor] and [Married: Yes] $\Rightarrow$ [NumCars: 2]".

PersonId	Profession	Married	NumCars
1	Professor	No	1
2	Student	No	0
3	Professor	Yes	2
4	Student	Yes	1
5	Student	No	1

Table 2.3: Table with Categorical Attributes

PersonId	[P: P]	[P: S]	[M: Y]	[M: N]	[NC: 0]	[NC: 1]	[NC: 2]
1	1	0	0	1	0	1	0
2	0	1	0	1	1	0	0
3	1	0	1	0	0	0	1
4	0	1	1	0	0	1	0
5	0	1	0	1	0	1	0

Table 2.4: Table with Only Boolean Attributes

2.1.4 Associative Classification

Association rules aim to discover all significant associations among items in a database. There is no predetermined target before the mining process except for the minimum support and minimum confidence constraints. In contrast, traditional classification rules aim to find rules that classify data records into several predefined classes. Classification rules always have one predetermined target, the class. Liu, Hsu, and Ma [LHM98] proposed associative classification, which integrates these two rule mining techniques. Associative classification focuses on mining a special subset of association rules whose heads are restricted to the class attribute. Such rules can be used for the purpose of classification. The algorithm they proposed to mine *class association rules* (CARs) is called CBA-RG, and is shown in Figure 2.1.

```
Input: Transaction Data: D, targetClass, minConfidence, minSupport
Output: mined association rules with targetClass and its possible values in
the heads, and the minimum confidence and minimum support constraints
satisfied.

1)   $F_1 = \{\text{frequent 1-ruleitems}\}$;
2)   $CAR_1 = genRules(F_1)$;
3)   for ($k = 2$; $F_{k-1} \neq \emptyset$; $k++$) do Begin
4)     $C_k = candidateGen(F_{k-1})$;
5)     for each data record $d \in D$ do Begin
6)         $C_d$ = all candidates in $C_k$ contained in $d$ ;
7)       for each candidate $c \in C_d$ do Begin
8)           $c$.condsupCount++;
9)           if $d$.class = $c$.class then $c$.rulesupCount++;
10)      end
11)    end
12)    $F_k = \{c \in C_k \mid c.rulsupCount \geq minsupCount\}$;
13)    $CAR_k = genRules(F_k)$;
14)  end
15)  $CARs = \bigcup_k CAR_k$ ;
```

Figure 2.1: The CBA-RG Algorithm [Liu et al. 98]

CBA-RG is adapted from the Apriori Algorithm. In the algorithm, a *ruleitem* has the form of $< condset, y >$, where *condset* is an itemset and y is a class label (e.g. [NumCars: 2]). The support count of the *condset* (called $condsupCount$) is the number of records in the transaction data D that contain the *condset*. The support count of the *ruleitem* (called $rulesupCount$) is the number of records in D that contain the *condset* and has class y. This *ruleitem* corresponds to a rule: *condset* $\Rightarrow y$, whose support is the percentage of the data records that contain the *condset* and has class y, i.e. ($rulesupCount$ / $sizeofDataset$) * 100%, and whose confidence is ($rulesupCount$ / $condsupCount$) * 100%.

The main difference between Apriori and CBA-RG is that CBA-RG will only mine rules for the target classes (e.g. [NumCars: 0], [NumCars: 1], [NumCars: 2], ...), and it could generate rules on the fly in the process of mining frequent *ruleitems*, i.e., it combines the two steps of mining association rules into one.

2.2 Recommender Systems

Recommender systems have recently attracted much research attention. The motivation of recommender systems is to scan a large number of articles automatically so as to recommend articles of potential interest to a given user on the basis of the user's taste. There are many online recommendation services available, which span the areas of book, music, movie, web page and restaurant recommendations. Examples of such systems are FireFly (www.firefly.com), WiseWire (www.wisewire.com), Content Advisor (www.contentadvisor.com), Jester (shadow.ieor.berkeley.edu/humor), Gustos (www.gustos.com), GroupLens (www.cs.umn.edu/Research/GroupLens). Some popular online CD stores and bookstores (e.g. amazon.com) also provide recommendation services.

2.2.1 Content-based and Collaborative Recommendation

There are currently two underlying categories of recommendation techniques: content-based recommendation and collaborative recommendation [BS97]. Content-based recommendation tries to recommend articles similar to those articles the user has liked. Generally, a user's profile is constructed by analyzing and extracting useful features from the content of the articles that the user has rated. Then recommendations are made based on this user's profiles. Content Advisor is a content-based web page recommender, which stores the features of articles in databases and responds to users' searches for the content they desire. Collaborative recommendation makes recommendations to a user based on the similarities and dissimilarities between this user and other users. Generally, it tries to find a group of people who share similar tastes with the given user and recommends articles they like to that user. The similarities or dissimilarities between users are calculated from their article's ratings. Gustos, GroupLens, FireFly and Jester are all examples of collaborative recommender systems.

Content-based recommendation and collaborative recommendation both have their own advantages and drawbacks. But collaborative recommendation is more popular than content-based recommendation, mainly because in many domains (such as music, restaurants) it is hard to extract useful features from articles, which is generally a step required for content-based recommendation. Many research efforts are invested in exploiting efficient algorithms for collaborative recommendation. But despite the increasing commercial demand for collaborative recommendation techniques, both the number of available published techniques and the evaluation of their performance are limited. The next section describes some of the main collaborative recommendation techniques.

2.2.2 Current Collaborative Recommendation Techniques

As mentioned in the introduction, many recommender systems use simple predictive techniques to make recommendations. One such technique that is widely used is the so called correlation-based method. Some other algorithms recently proposed for recommender systems employ techniques from statistics and machine learning. Examples of them could be found in [BP98, BHK98, SM95]. We briefly describe some of those methods below.

The Correlation-based Method

The vast majority collaborative recommender systems use the correlation-based method to model similarities between users, where they calculate the vote from the active (or target) user (indicated with a subscript a) for article j, $p_{a,j}$, as a weighted sum of the votes of other users. The weights reflect correlations among users.

One example of the formula given in [RIS+94] is:

$$p_{a,j} = \overline{v}_a + k \sum_{i=1}^{n} w(a,i)(v_{i,j} - \overline{v}_i)$$

where $v_{i,j}$ corresponds to the vote from user i on item j; $\overline{v}_i$ corresponds to the average vote from user i; n is the number of users in the collaborative database; k is a normalizing factor such that the absolute values of the weights sum to one; and the weights $w(i,a)$ reflect the similarity between each user i and the active user, and are calculated as:

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

where the summations over j are over the articles for which both users a and i have voted.

As mentioned in [BP98], such correlation-based methods suffer from several drawbacks. Firstly, the significance of the correlations between users are not measured; Secondly, when no correlation is found between two users, some potentially useful information about these two users is lost; Most importantly, if two users do not rate articles in common, they can not be similar under the correlation method even if they share common interests. Our approach can possibly overcome these drawbacks.

Bayesian Classifier and Bayesian Network Model

Breese, Heckerman and Kadie [BHK98] list and test several algorithms for collaborative recommendations. They propose a new approach for finding dependences among articles by using a Bayesian classifier and a Bayesian network model. The idea of this approach is similar to our article associations. But (1) they need to calculate the conditional probabilities of all the possible ratings for an article given all possible ratings for other articles, which is computationally expensive and (2) they can not estimate how good a prediction they made is. Our approach (1) only needs to find some significant dependences among articles, which is above a certain minimum support and (2) we can evaluate a prediction according to the support and confidence of the rule. Since we are only concerned with recommending a certain number of interesting articles, not predicting the ratings for all articles, the significant dependencies are good enough.

Neural Networks Paired with Feature Reduction Techniques

Billsus and Pazzani [BP98] present a framework for applying machine learning algorithms paired with feature reduction techniques, such as singular value decomposition(SVD) or information gain, for collaborative recommendations. Firstly, they use feature reduction techniques to reduce the dimension of the rating data, and then neural networks are applied to the simplified data to construct a model for recommendation. While this framework is good for neural networks, it is not so appropriate for rule-based learning. For example, after the singular value decomposition, the previous boolean valued matrix becomes a matrix containing continuous numbers. So we need to discretize those continuous numbers in order to derive rules from them, which is an additional problem. In Chapter 5, we compared our approach with this approach as well as the correlation-based method under some similar experimental conditions.

Association Rules

Association rules have been widely used in web mining. They have been used to mine path traversal patterns and to facilitate the best design and organization of web pages [CMS97, CSM97, CPY98] Recently, Fu, Budzik and Hammond[FBH00] developed a good framework to mine association rules in navigation history for recommending web pages. Actually, we developed our ideas independently. But they use the original Apriori algorithm to mine rules. As we will discuss later, the traditional association rule mining algorithms are not good enough for recommender systems. Also, they only test the performance of their system on some simple artificial dataset.

Chapter 3

Our Adaptive-support Algorithm for Mining Association Rules

In this chapter we describe our algorithm to mine association rules (AR-CRS). The association rule mining is the basic part of this thesis work. Our algorithm AR-CRS (Association Rules for Collaborative Recommender System) is adapted from Apriori [AS94a] and CBA-RG [LHM98]. This algorithm adjusts the minimum support of the rules during mining in order to obtain an appropriate number of significant rules for the target predicate.

The Apriori algorithm is a widely used algorithm to mine association rules. It aims to find all the associations among items, i.e., given a set of transactions (with each transaction containing a set of items), a specified minimum support and minimum confidence, it tries to find all the association rules satisfying the minimum support and the minimum confidence constraints. CBA-RG is an algorithm that aims to mine association rules to predict the class of a data record. It focuses on mining a special subset of association rules whose heads are restricted to the target classes. Our algorithm AR-CRS is even more focused than CBA-RG in the sense that we are only interested in mining rules with one specified target value in the head. Also, our algorithm adjusts the minimum support of the rules during mining in order to obtain an appropriate number of significant rules for the target item.

3.1 Problem Description

Our mining algorithm focuses on mining rules for only one target user or article at a time. This has the following advantages:

1. Since we are only interested in predicting articles that a target user would like, in user associations, we only need rules with [$user_{target}$: *like*] in the rule head. Such rules could be mined more efficiently than the rules with arbitrary heads. Since we need to mine user associations online, the efficiency of the process is of great importance.

2. By mining article associations for one article at a time we are able to obtain rules for articles that have only received a limited number of ratings, for example a new movie. This would not be possible if we mined article associations for all articles at once, because rules for new articles would fail to have the necessary support.

3. A significant amount of runtime is saved by mining rules only over the subset of the transaction data that is related to the target user or article instead of over the whole data.

Also, we want to specify a high minimum confidence and, instead of a minimum support, a range for the number of rules before the mining process. This is because of the following reasons: The confidence reflects the degree of the correlations among users or among articles, and the support reflects the significance of the correlations. Hence, it is evident that they should both be very important for making recommendations. We could expect that the higher the confidence and the support, the higher the quality of the recommendations. But it is difficult to choose a proper minimum confidence and support for each user/article before the mining process, because users' tastes and articles' popularities vary widely. If the minimum confidence and support are set too high, we can not obtain enough rules for accurate recommendation; If they are set too low, we will be in danger of having an unacceptable long period of runtime. Also, we feel that for each user/article, a limited number of rules are good enough for recommendation, too many rules are not necessary. Considering these factors, we think a best way is to specify a high minimum confidence and a range for the number of rules, and let the system find a proper minimum support that produces the desired number of rules. By doing this, actually we choose only a small number of the most significant rules above the specified minimum confidence for recommendation.

So our problem of mining association rules for recommender systems could be described as follows:

Problem Definition

Given a transaction dataset, a target item, a specified minimum confidence and a desired range [$minRulenum$,$maxRulenum$] for the number of rules, find association rules with the target item in the heads of the rules such that the number of rules is in the given range, the rules have the highest possible support, and the rules satisfy the minimum confidence constraint.

Note. Since we use the same algorithm to mine user associations as well as article associations, we use the term *target item* to denote "target user" in the case of user associations and “target article” in the case of article associations.

3.2 Algorithm Description

For the problem we described above, the development of a new algorithm was necessary for the following two main reasons:

- Previous algorithms do not provide a mechanism to choose a proper minimum support for the given minimum confidence and the desired range for the number of rules.
- It is only necessary to mine rules for the target user/article. Although CBA-RG has addressed the problem of mining rules only for target classes, our algorithm is even more focussed since it mines rules for only one target class value: $[item_{target}$: like$]$.

Our AR-CRS algorithm solves the above problem. AR-CRS consists of two parts: AR-CRS-1 and AR-CRS-2.

AR-CRS-1

In order to mine only a given number of most promising rules for each target item, we use AR-CRS-1 to control the minimum support count and find the rules with the highest supports. The minimum support count is the minimum number of transactions that satisfy a rule in order to make that rule frequent, i.e., it is the multiplication of the minimum support and the whole number of transactions. The overall process is shown in detail in Figure 3.1. It consists of three parts:

1. AR-CRS-1 initializes the minimum support count according to the frequency of the target item and calls AR-CRS-2 to mine rules.

2. When AR-CRS-2's output is returned, AR-CRS-1 will check first if the number of rules returned is equal to $maxRulenum$ (as we describe below, AR-CRS-2 terminates the mining process when the number of rules generated is equal to $maxRulenum$). If it is, that means the minimum support count is low which results in more than $maxRulenum$ rules, so the AR-CRS-1 will keep increasing the minimum support count and calling AR-CRS-2 until the number of rules is less than $maxRulenum$.

3. Finally, AR-CRS-1 will check if the number of rules is less than $minRulenum$; if it is, it will keep decreasing the minimum support count until the rule number is greater than or equal to $minRulenum$.

```
Input: Transactions, targetItem, minConfidence, minRulenum, maxRulenum,
Output: mined association rules with targetItem in the heads. The number of
rules is in the range [minRulenum, maxRulenum], the rules have the highest
possible support, and satisfy the minimum confidence constraint.

    1)   Set initial minsupportCount based on targetItem's like ratio;
    2)   R = AR-CRS-2();
    3)   while (R.rulenum = maxRulenum) do
    4)     minsupportCount++;
    5)     R_1 = AR-CRS-2();
    6)     if R_1.rulenum > minRulenum then R = R_1;
    7)     else return R;
    8)   end
    9)   while (R.rulenum < minRulenum) do
    10)    minsupportCount--;
    11)    R = AR-CRS-2();
    12)  end
    13)  return R;
```

Figure 3.1: The AR-CRS-1 Algorithm

Within a given support, rules with shorter bodies are mined first. Hence, if with minimum support count say 15 there is no rule available, but with minimum support count 16 there are at least maxRulenum rules, then AR-CRS-1 will return the shortest maxRulenum rules with support count of at least 16.

AR-CRS-2

AR-CRS-2 is a variant of CBA-RG [LHM98] and therefore of the Apriori algorithm [AS94a] as well. AR-CRS-2 is a variant of CBA-RG in the sense that instead of mining rules for all target classes, it only mines rules for one target item. It differs from CBA-RG in that it will only mine a number of rules within a certain range. When it tries to generate a new rule after having obtained $maxRulenum$ rules already then it simply terminates its execution and returns the rules it has mined so far.

Here we use *k-condset* to denote a set of items (or *itemset*) of size k which could form a rule: *k-condset* $\Rightarrow$ *target-item*. The support count of the *k-condset* (called $condsupCount$) is the number of transactions that contain the *k-condset*. The support count of the corresponding rule (also called $rulesupCount$ of this *k-condset*) is the number of transactions that contain the *condset* as well as the target item.

AR-CRS-2 is very similar to CBA-RG. Association rules are generated by making multiple passes over the transaction data. The first pass counts the $rulesupCounts$ and the $condsupCounts$ of all the single items and finds the frequent 1-*condsets*. For pass $k > 1$, it generates the candidate frequent *k-condsets* by using the frequent $(k-1)-condsets$; then it scans all transactions to count the $rulesupCounts$ and the $condsupCounts$ of all the candidate *k-condsets*; finally, it will go over all candidate *k-condsets*, selecting those whose *rulesup* is above the minimum support as frequent *k-condsets* and at the same time generating rules *k-condset* $\Rightarrow$ *target-item*, if the confidence of the rule is above the minimum confidence. The algorithm is presented in Figure 3.2.

```
Input: Transactions, targetItem, minConfidence, maxRulenum, minsupportCount
Output: mined association rules with targetItem in the heads. The number of
rules is in the range [0, maxRulenum], the rules satisfy the minimum
confidence and minimum support constraints.

1)    $F_1 = \{\text{frequent 1-condsets}\}$;
2)    $R = \text{genRules}(F_1)$;
3)    if $R$.rulenum = maxRulenum then return $R$;
4)    for $(k = 2; F_{k-1} \neq \emptyset; k++)$ do Begin
5)      $C_k = \text{candidateGen}(F_{k-1})$;
6)      for each transaction $t \in Transactions$ do Begin
7)        $C_t = \text{all candidate condsets of } C_k \text{ contained in } t$;
8)        for each candidate $c \in C_t$ do Begin
9)          c.condsupCount++;
10)         if t contains targetItem then c.rulesupCount++;
11)       end
12)     end
13)     $F_k = \{c \in C_k \mid c.\text{rulesupCount} \geq minsupportCount\}$;
14)     $R = R \cup \text{genRules}(F_k)$;
15)     if $R$.rulenum = maxRulenum then return $R$;
16)   end
17)   return $R$;
```

Figure 3.2: The AR-CRS-2 Algorithm

3.3 Real-time recommendation

A requirement of many recommender systems is realtime response. Our algorithm can satisfy the real time constraint for the following reasons:

- We mine rules offline for article associations;
- The training data to mine rules for one target user is only a small subset of all the ratings, i.e., the ratings from training users and the target user for the articles that the target user has rated. So the training data size is small;
- The mining process AR-CRS-2 will stop after it mines $maxRulenum$ rules. If the $maxRulenum$ is small, it is very fast;

- In the main process, we choose an initial minimum support count according to the frequency of the target user. For most users, the main process only needs to call the mining process two or three times. We can switch to article associations for users who need more iterations.

3.4 Algorithm Implementation

3.4.1 Key Operations

According to the above descriptions, the main operations of this algorithm are the following:

- subset test: how to find all candidate *condsets* that are contained in one transaction;
- candidate generation - join step: how to find frequent *condsets* that could be joined together;
- candidate generation - prune step: how to test if any (k-1)-subset of a candidate *k-condset* is a frequent (k-1)-*condset*;

3.4.2 Data Structures

Considering the possibility of large amounts of data, the above three key operations could be very expensive. So it is very important to use nice data structures that facilitate those operations. As mentioned in [AS94a], using a hash-tree to store candidate itemsets and a bitmap to store a transaction could speed up the support counting process.

Bitmap

We use a bitmap to represent a transaction, each bit corresponding to an item. If the transaction contains the item, then the value of the bit corresponding to the item is 1, otherwise it is 0.

Hash-tree

A node of the hash-tree either contains a list of itemsets (a leaf node) or a hash table (an interior node). In an interior node, each bucket of the hash table points to another node. The root of the hash-tree is defined to be at depth 1. An interior node at depth d points to nodes at depth d+1. Itemsets are stored in the leaves. When we add an itemset c, we start from the root and go down the tree until we reach a leaf. At an interior node at depth d, we decide which branch to follow by applying a hash function to the dth item of the itemset. All nodes are initially created as leaf nodes. When the number of itemsets in a leaf node exceeds a specified threshold, the leaf node is converted to an interior node.

Set-tree

Since we will not use a database to store itemsets, we also need a data structure to facilitate the join and prune operations on candidate itemsets. We found a hash-tree also has some advantage for doing this. But a big problem is how to organize a list of itemsets contained in a leaf node. To solve this problem we designed and implemented a new tree structure: set-tree. An example of a set-tree is given in Figure 3.3.

A set-tree stores a set of itemsets in a tree. Each node could point to a sibling and a child. A node at depth n corresponds to the nth element of an itemset whose elements are in order. For example, Figure 3.3 represents four itemsets: {1, 2, 3,

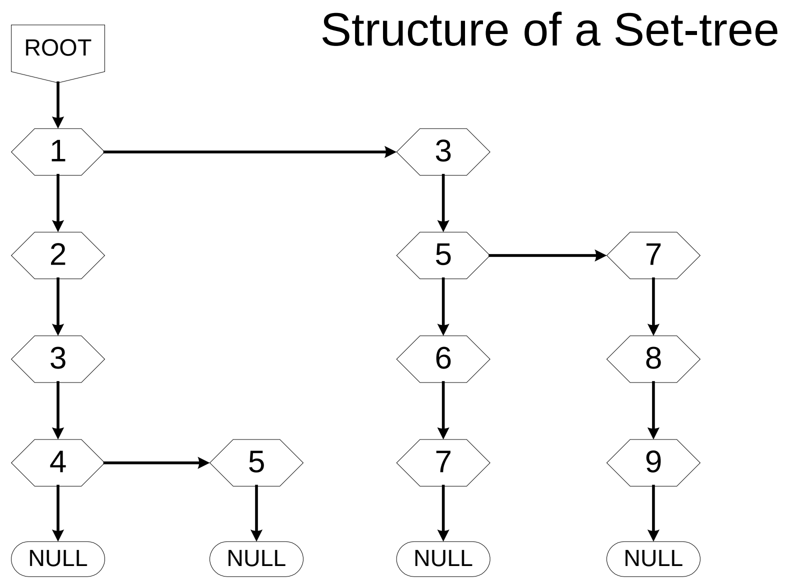


Figure 3.3: Structure of a Set-tree

4}, {1, 2, 3, 5}, {3, 5, 6, 7} and {3, 7, 8, 9}.

Combining Set-trees with a Hash-tree

An interior node of a hash-tree is a hash table, while a leaf node of a hash-tree is a set-tree. As shown in Figure 3.4, a hash-tree grows when we add more itemsets. At first, the root node of a hash-tree points to a set-tree that contains all the itemsets. It is converted to an interior node, a hash table that hashes on the first item of the itemsets, when the number of itemsets is greater than a split threshold. A leaf node is converted to an interior node when the number of itemsets it contains grows above the threshold, and if the leaf node is on the nth level of the tree, after the conversion, the corresponding hash table will hash on the nth item of itemsets.

Using a hash-tree plus set-trees to store itemsets facilitates the support counting of candidate itemsets, the join operation, the looking up of an itemset and even the prune operation. Figure 3.5 shows the join operation based on a set-tree. It is very easy to locate the itemsets that could be joined together. Using set-trees also

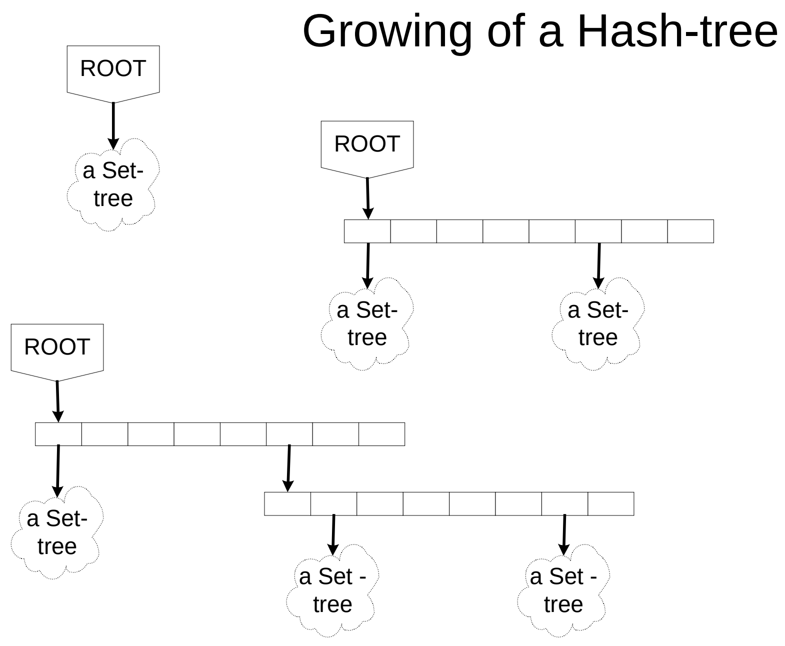


Figure 3.4: Growing of a Hash-tree

makes the conversion of an interior node to a leaf node very easy. The threshold determines the maximum size of a set tree. By choosing the threshold, we can adjust the tradeoff between memory space and running time.

Finally, we use these data structures to implement the following concepts:

- one transaction: a bitmap;
- all transactions: an array of transactions;
- candidate itemsets: a hash-tree with set-trees;
- frequent itemsets: a hash-tree with set-trees;

3.5 The Input Data

Our system takes in the target item ID and the transaction data related to the target item and mines rules for the target item. The transaction data are stored

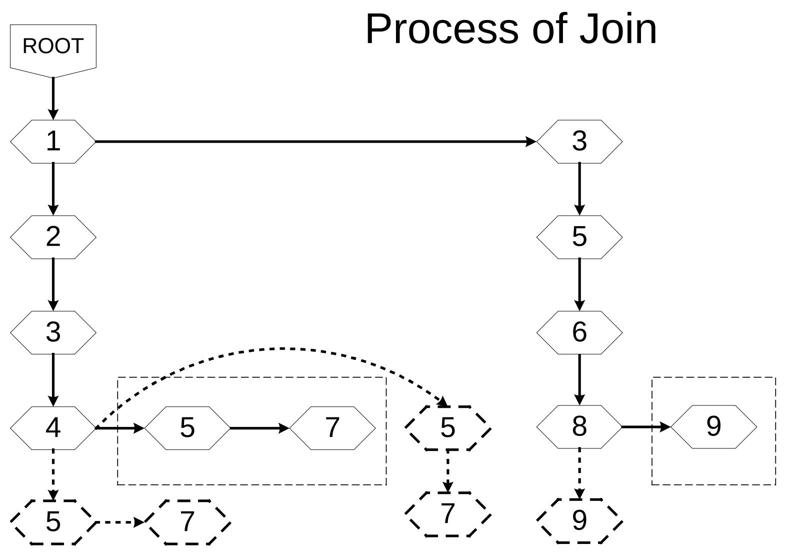


Figure 3.5: Join Process

in files. We have each transaction stored in one file, which is called a transaction file. For each target item, there is one transaction list file containing the IDs of transactions related to the target item. In the following, we give the examples of transaction data for like user associations.

- As we have described in Section 1.2.2, in user associations each user corresponds to an item and each article corresponds to a transaction. So each transaction file contains the ratings for an article from the collaborative users (the users whose tastes are used to match the target user's taste). We give each file a name that reflects the corresponding article, e.g., "article1.dat", "article5.dat", "article14.dat",... Each transaction file contains a bitmap, each bit in the bitmap corresponding to one collaborative user, with value "1" representing that this user likes the article and value "0" representing dislike or not rate.

- For each user, only articles that he/she has rated can be used to mine rules for him/her. So we have a list file for each target user which contains all the articles he/she has rated. Its first line contains the total number of articles the target user has rated. The rest of the file has two columns: the first column is the article ID and the second column reflects if the target user likes the article, where value "1" means he/she likes the article and value "0" means he/she dislikes the article. For example, the transaction list file for person70007 (named "person70007.dat") is as follows:

```
``108 articles rated.
  1        1
  5        0
  14       0
  ...''
```

Having transaction list files and transaction files, we can easily read in all the transactions related to a target item.

3.6 The Output Rules

Examples of output rules of our system are shown below:

- With user associations, our system will mine association rules like:
  "[person69: like] AND [person580: like] $\Rightarrow$ [person70007: like]" for *like* associations, or
  "[person358: like] AND [person677: dislike] $\Rightarrow$ [person70007: like]" for *like* and *dislike* associations;

- With article associations, it will mine association rules like:
  "[movie30: like] AND [movie160: like] $\Rightarrow$ [movie450: like]" for *like* associations, or

"[movie70: like] AND [movie289: dislike] $\Rightarrow$ [movie450: like]" for *like* and *dislike* associations.

3.7 Summary

We have presented a new algorithm for mining association rules with a specific target predicate in the heads of the rules. Such rules are needed in applications such as recommender systems. Unlike most existing association rule mining algorithms, which require that the minimum support of the rules to be mined be specified in advance, our algorithm adjusts the minimum support during the mining process so that the number of rules generated lies within a specified range. This keeps the running time under control, and ensures that enough rules are available for the target predicate.

Chapter 4

Recommendation via Association Rules

We have described our algorithm for mining association rules. This chapter describes our approaches to use association rules for recommendation: user associations and article associations. For these two types of associations, we basically employ the same mining process (with different types of transaction data), but quite different recommendation strategies.

4.1 The Training Data

Association rules are mined from a set of "transactions". For collaborative recommendation, we usually have users' ratings of articles. How to convert ratings to "transactions" is determined by which kind of associations and how many levels of associations we want to discover.

4.1.1 User Associations

Firstly, we are interested in predicting if a user would like an item. Hence we map the ratings for an item into two categories: *like* and *dislike* according to whether the rating for the item is greater than or less than some threshold value.

Then we convert the *like* and *dislike* ratings into transactions by following the process below:

- In order to obtain *like* associations among users, we have each user correspond to an "item" and each article rated by users correspond to a "transaction". If a user, say $user_k$, likes an article, then the transaction corresponding to the article contains the item $[user_k : like]$; If $user_k$ dislikes or did not rate the article, then the corresponding transaction does not contain the corresponding item. From here, we can mine *like* associations among users, for example, "90% of articles liked by user A and user B are also liked by user C, 30% of all articles are liked by all of them", or simply denoted as "$[user_a : like]$ AND $[user_b : like] \Rightarrow [user_c : like]$ with confidence 90% and support 30%".

- In order to mine *like* and *dislike* associations among users, we employed the idea of mining categorical association rules (see Section 2.1.3) by extending each user, say $user_k$, to two "items", one item corresponding to $[user_k : like]$ and another item corresponding to $[user_k : dislike]$. If $user_k$ likes an article, then the corresponding transaction contains the item $[user_k : like]$; If $user_k$ dislikes the article, the corresponding transaction contains the item $[user_k : dislike]$; If $user_k$ didn't rate the article, the corresponding transaction doesn't contain either of these two items. From here, we could mine rules like "90% of articles liked by user A and disliked by user B are liked by user C, 30% of all articles are liked by user A and C and disliked by user B", or simply denoted as "$[user_a : like]$ AND $[user_b : dislike] \Rightarrow [user_c : like]$ with confidence 90% and support 30%"

More specifically, the final training data for a target user is obtained as follows: Firstly, split the articles this user has rated into training articles and test articles, then have each training article correspond to one training transaction. So the number of training articles determines the total number of training transactions. We also have each collaborative user correspond to one item for *like* associations and two items for *like* and *dislike* associations. Whether a transaction contains an item is determined as above. Also, the target user corresponds to one and only one item, i.e., $[user_{target} : like]$, which is contained in a transaction if the target user likes the corresponding article. Table 4.1 gives an example of the training data to mine *like* and *dislike* user associations for a target user.

Article ID	[C1: L]	[C1: D]	[C2: L]	[C2: D]	[C3: L]	[C3: D]	[Target_U: L]
1	1	0	1	0	1	0	0
2	0	1	0	0	1	0	1
3	1	0	0	1	1	0	1
4	0	1	0	1	0	1	0
5	1	0	0	1	0	0	1

Table 4.1: Training Data for User Associations

Here [C1: L] means $[collaborative\_user_1 : like]$, [C1: D] means $[collaborative\_user_1 : dislike]$. So from the transaction data in the table, we can mine rules like: "$[collaborative\_user_1 : like]$ AND $[collaborative\_user_2 : dislike] \Rightarrow [target\_user : like]$ with confidence 100% and support 40%."

4.1.2 Article Associations

In order to obtain *like* associations for a target article, we have each article correspond to an "item" and each training user who rated the target item correspond to a "transaction". If a training user likes an article, then the transaction corresponding to the user contains the item corresponding to the article; If the user dislikes or did not rate the article, then the corresponding transaction does not contain the corresponding item. From here, we can mine *like* associations among articles. For example, Table 4.2 gives an example of the training data to mine *like* article associations for a target article.

User ID	[A1: L]	[A2: L]	[A3: L]	[A4: L]	[A5: L]	[A6: L]	[Target_A: L]
1	1	0	0	0	1	0	0
2	0	1	0	0	1	0	1
3	1	0	0	1	1	0	1
4	0	1	1	1	0	1	0
5	1	0	0	1	0	0	1

Table 4.2: Training Data for *Like* Article Associations

Here [A1: L] means $[article_1 : like]$, and the first column corresponds to the collaborative user ID. So from the transaction data in the table, we can mine rules like: "$[article_1 : like]$ AND $[article_4 : like] \Rightarrow [target\_article : like]$ with confidence 100% and support 40%."

4.2 Recommendation Strategy

4.2.1 User Associations

For user associations, basically we have rules vote for articles, where each vote is weighted according to the quality of the rule (referred to as the *score* of the rule), if an article receives a score above a certain value, then that article is recommended. The whole strategy is described in more detail below:

1. For user associations, the rules we have are in the form of: $[collaborative\_user_1 : like]$ AND $[collaborative\_user_2 : dislike] \Rightarrow [target\_user : like]$. For a test article of the target user, if the $collaborative\_user_1$ likes this article and the $collaborative\_user_2$ dislikes this article, then we say this rule fires for this article.

2. We associate a score to each rule, which is the product of the support and the confidence of the rule.

$$score_{rule_k} = support_{rule_k} * confidence_{rule_k}$$

3. We assign a score to each test article, which is the sum of the scores of all the rules that fire for this article. Assume a user has n rules, then the *score* of one of his/her test articles $article_i$ is calculated as below:

$$score_{article_i} = \sum_{k=1}^{n} (score_{rule_k} |\ rule_k \text{ fires for } article_i)$$

4. If $score_{article_i}$ is greater than a threshold, then we will recommend $article_i$ to the given user.

We use the product of the support and the confidence of a rule as this rule's score, because we believe that the support and the confidence represent the quality of a rule, which will directly influence the success of the recommendations made by this rule, i.e., the higher the support and the confidence, the better the recommendations made by the rule. Our experiments verified this assumption (see Section 5.2.2).

We employ a score threshold and only recommend articles whose scores are above the threshold in order to in some degree reduce the influence of some noisy rules (rules that overfit the training data and are not accurate for prediction). For example, if there are only a small number of rules with low minimum support and confidence which recommend an article, then the weighted vote to recommend this article is not so strong. So by choosing an appropriate threshold, one can filter out such kind of articles.

How to choose the score threshold is of great importance. We do not want to recommend bad articles nor to filter out good articles. Basically, we considered two kinds of choices: a constant threshold and a linear threshold (which is a linear function of the number of rules obtained for a target user). By choosing a constant threshold, we assume that if an article is recommended by a large enough number of rules, this article should be recommended. But considering that the number of noisy rules may grow as the total number of rules grows, it is hard to choose a constant threshold that works for both large numbers of rules and small numbers of rules. So we also tried thresholds that are linear functions of the number of rules. We ran some tests on both of these two kinds of thresholds and the results are shown in Section 5.2.2.

4.2.2 Article Associations

For article associations, we use a different strategy. Because not all the articles are good for recommendation, we only recommend articles whose rules' supports above a support cutoff.

1. Given a rule: $[article_1 : like]$ AND $[article_2 : like] \Rightarrow [target\_article : like]$, if the target user likes $article_1$ and $article_2$ (which can be known from the training articles of the user), then we say this rule fires for this article.

2. Specify a support cutoff.

3. For a test article of the target user, if there is a rule with support above the specified support cutoff which fires for this article, then this article is recommended.

After we have known which value is best for the support cutoff in the system tuning process, we can only mine rules above the support cutoff during the mining process. This may seem similar as Apriori or CBA in the sense that a minimum support is specified in the mining process, but the difference is that we only mine rules with the highest possible support for one article at a time and the number of rules obtained lies within the specified range. Our mining process has the following advantages:

- By mining article associations for one article at a time, only ratings related to the target article are used for mining, which is only a small subset of the whole rating data. The support of a rule is calculated over the small subset of the whole rating data, which enables us to obtain rules for articles that have only received a limited number of ratings, for example a new movie. This would not be possible if we mined article associations for all articles at once, because rules for new articles would fail to have the necessary support over the whole rating data.

- A significant amount of runtime is saved by mining rules only over the subset of the rating data that is related to the target article instead of over the whole data. We have tried to use Intelligent Miner (IBM product) to mine article associations for all articles at once. It usually took several days and still could not terminate.

4.3 Recommendation Modes

The system modes for user associations and article associations are different. For user associations, we must mine rules online. This is because the ratings from the target user are needed to mine associations between the target user and other users; But for article associations, the mining process can be offline. This is because we can mine associations between articles from the collaborative users, i.e., the ratings from the target user are not necessary in the mining process. This difference results in the different response time of these two approaches.

Chapter 5

Experimental Tuning and Evaluation

This chapter shows the results of the experiments we performed to tune our system and to evaluate its performance. It includes the results for user associations, article associations, and a combination of the two. A comparison with other systems under somewhat similar experimental conditions is also given in the end of this chapter.

5.1 Experimental Protocol

5.1.1 The EachMovie Data Set

We use the EachMovie Dataset as the test-bed of our approaches. The EachMovie data set is an online data source provided by the Systems Research Center of DEC [McJ97]. It contains ratings from 72,916 users for 1,628 movies. The whole data set are stored in three tables:

- *Person* provides optional, unaudited demographic data supplied by each person;

- *Movie* provides descriptive information about each movie;
- *Vote* is the ratings from persons for movies. User ratings were recorded on a numeric six-point scale(0.0, 0.2, 0.4, 0.6, 0.8, 1.0).

5.1.2 Collaborative Users and Target Users

We performed two groups of experiments. In the first group of experiments, we chose the first 1000 users in the EachMovie dataset who have rated more than 100 movies as collaborative users, and the first 100 users whose userIDs are greater than 70,000 and who have rated more than 100 movies as target users. Some people in the database have only rated a couple of movies. Obviously, such ratings would not be suitable to be training data. By choosing collaborative users who have rated over 100 movies, we assume that those people have rated almost all the movies they have seen and such ratings are useful to mine associations exhaustively. Also, to choose target users who rated over 100 movies makes us have enough movies for recommendation as well as for test.

In order to compare our approach with other approaches, we performed a second group of experiments, for which we chose the first 2000 users in the database as collaborative user group, and 91 random users whose like ratios are less than 0.75 and who have rated 50 to 100 movies as target users.

5.1.3 4-fold Cross-validation

During our tests, for each given target user, we employ the *4-fold* cross-validation approach. First, we randomly divide all the articles this user has rated into 4 groups. Then we run four rounds of test, each time choosing one group of articles as test data and the other three groups as training data. So every article this user rated will be used as a test article once.

5.1.4 Performance Measures

We use the *accuracy*, a commonly used performance measure in machine learning, together with two standard information retrieval measures, *precision* and *recall*. Accuracy is the percentage of correctly classified articles among all those classified by the system; Precision is the percentage of articles recommended to a user that the user likes; Recall is the percentage of articles liked by a user that are recommended to him/her. More precisely,

$$accuracy = \frac{correctly\_classified\_articles}{total\_articles}$$

$$precision = \frac{correctly\_recommended\_articles}{total\_recommended\_articles}$$

$$recall = \frac{correctly\_recommended\_articles}{total\_articles\_liked\_by\_users}$$

For recommendation tasks, precision is perhaps most significant because we are more concerned about making high quality recommendations than about recommending a large number of items. So our goal is to achieve a very high precision with a reasonably high recall. But the accuracy reflects an implicit combination of the precision and the recall. In this sense, the accuracy is also important.

5.2 Experimental Evaluation

In this section, we report results of our experiments. For all the experiments, we employed the *4-fold* cross-validation approach when evaluating the performance for each test user, and reported the averaged performance over the 100 test users.

5.2.1 Parameters

The main parameters for our approaches are listed below:

- the *like* and *dislike* threshold to split the ratings into *like* and *dislike* categories;
- the maximum length of rules;
- the minimum confidence of the association rules;
- the specified range for the number of rules, i.e., the *minRulenum* and the *maxRulenum*;
- the score threshold for user associations;
- the support cutoff for article associations.

In order to choose appropriate parameters, we did some experiments. Some interesting results are shown in the next section. Currently, we choose 0.7 as the *like* threshold, i.e., if a user's rating for an article is greater than 0.7, then we think that user likes this article. By choosing this *like* threshold, the ratio of the number of *like* movies over the total number of movies (called *like ratio*) of all the test users is 0.45.

5.2.2 User Associations

Maximum Rule Length

We use rule length to refer to the number of the items present in the rule precedent. Table 5.1 lists the performance of different maximum rule lengths. Here we choose minimum confidence as 100% and rule number in the range of 5 to 100.

Rule Length	2	4	6	8	10
Accuracy	0.693712	0.696117	0.695676	0.69759	0.694547
Precision	0.704357	0.724006	0.733482	0.737896	0.736086
Recall	0.572606	0.545425	0.528625	0.528411	0.520813

Table 5.1: Performance for Different Maximum Rule Length

From Table 5.1 we could see that when the maximum rule length is around 8, we get the best performance. Generally, with a larger maximum rule length, we can get more rules above a certain minimum support and minimum confidence. But actually, there are very few rules with rule length greater than 8 which have a relatively high support and confidence. Also, the long rules have the danger of overfitting data. These are the main reasons why we can not achieve better performance with a rule length 10.

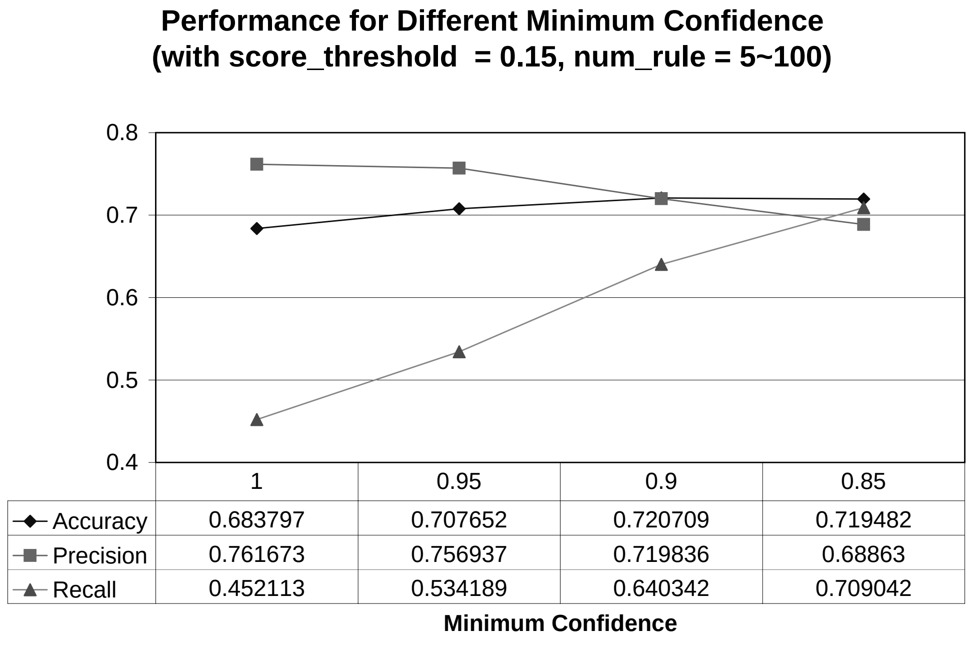
From these results, we can choose the maximum rule length to be 8. We use this maximum rule length for all the following experiments.

Minimum Confidence

We believe that the minimum support and minimum confidence are the two most important factors that influence the performance. Since the minimum support for each user is decided automatically during the mining process. It would be interesting to study the performance for different minimum confidence. We tested the performance when varying the minimum confidence. In our tests, we use two kinds of score thresholds: a constant threshold and a linear threshold (which is a linear function of the rule number). Their results are shown in Figure 5.1 and Figure 5.2.

From Figure 5.1 and Figure 5.2, we could see very clearly the following facts:

- The minimum confidence has a significant impact on the performance, i.e., the higher the minimum confidence, the higher the precision but the lower the



	1	0.95	0.9	0.85
◆ Accuracy	0.683797	0.707652	0.720709	0.719482
■ Precision	0.761673	0.756937	0.719836	0.68863
▲ Recall	0.452113	0.534189	0.640342	0.709042

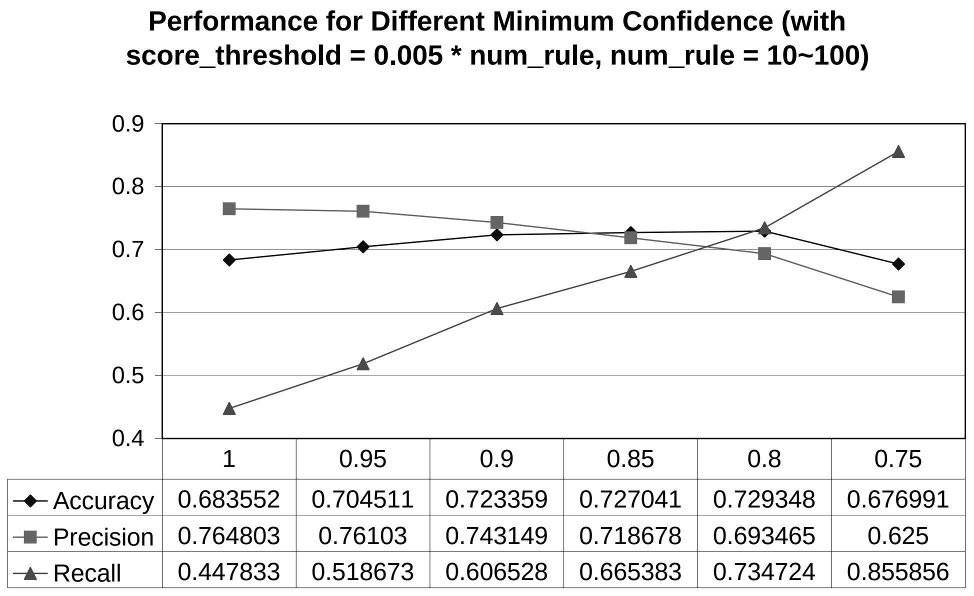
Figure 5.1: Performance for Different Minimum Confidence with a Constant Score Threshold

recall. We achieve the highest precision of 0.76 with a recall of 0.45 for the minimum confidence of 100%. This is not surprising because the confidence of a rule corresponds to the average precision of using this rule to recommend training articles to the target user,

- When the minimum confidence is varied, there shows a tradeoff between the precision and the recall. If we take accuracy as a measure that reflects both the precision and the recall, we achieve the best combination of the precision and the recall not with minimum confidence 100% but with minimum confidence from 80% to 90%. This is because we have more like articles identified as like, i.e., we have higher recalls.

- Also, we achieve a better combination of the precision and the recall by using a linear threshold.

Even though we think the precision is the most important thing for a recom-



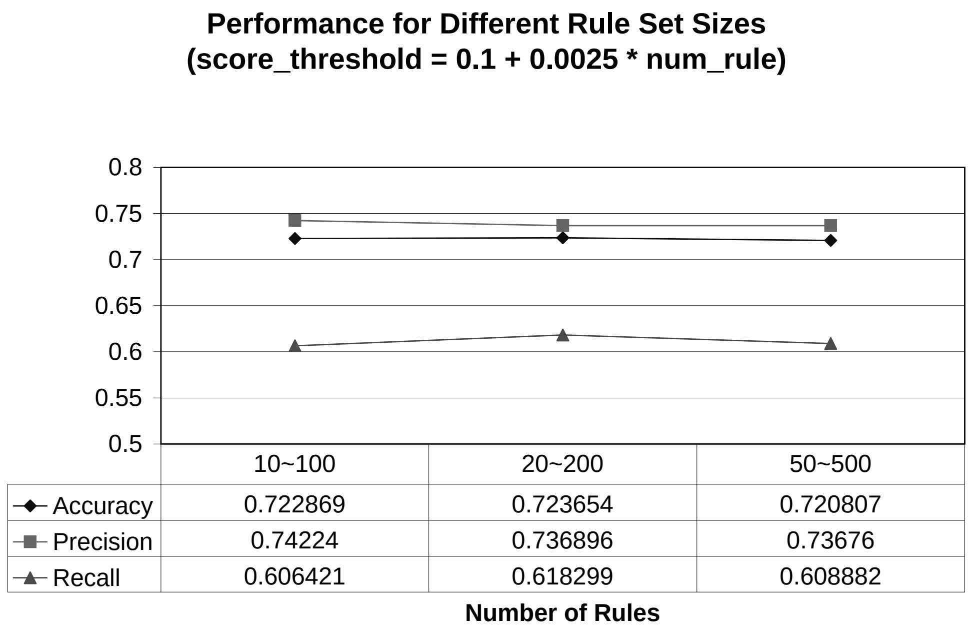
	1	0.95	0.9	0.85	0.8	0.75
◆ Accuracy	0.683552	0.704511	0.723359	0.727041	0.729348	0.676991
■ Precision	0.764803	0.76103	0.743149	0.718678	0.693465	0.625
▲ Recall	0.447833	0.518673	0.606528	0.665383	0.734724	0.855856

Figure 5.2: Performance for Different Minimum Confidence with a Linear Score Threshold

mender system, the best combination of the precision and the recall is also important in the sense that we can achieve both a higher precision and a higher recall from there. Under this consideration (which is actually verified in the next sections), we use linear thresholds and the minimum confidence of 0.9 for almost all the rest experiments.

Range for the Number of Rules

In order to decide what is the appropriate range for the number of rules, we did experiments with different rule set sizes. As shown in Figure 5.3, we achieve quite similar performance for the number of rules within 100, 200 and 500. But considering the much longer runtime required for larger rule numbers, we choose the range for the number of rules to be from 10 to 100. This result also verifies our assumption that it is enough to have a small number of rules, too many rules are not necessary.



	10~100	20~200	50~500
◆ Accuracy	0.722869	0.723654	0.720807
■ Precision	0.74224	0.736896	0.73676
▲ Recall	0.606421	0.618299	0.608882

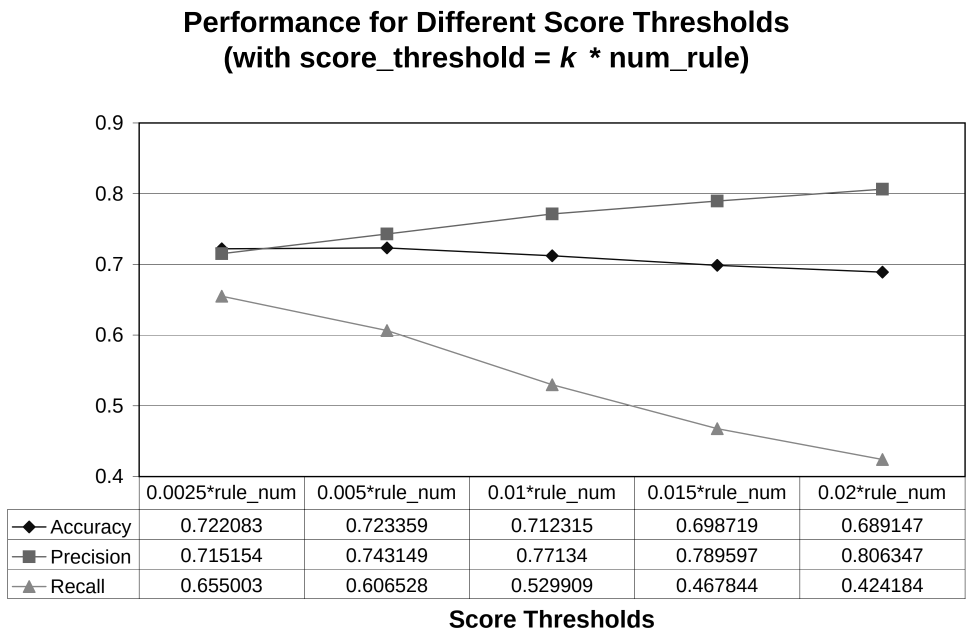
Figure 5.3: Performance for Different Rule Set Sizes

Score Threshold

Score threshold is also a very important parameter of our approach. We have decided to use a linear threshold, but still need to decide the base value and the slope of the linear function. Figure 5.4 and Figure 5.5 give the performance for different score thresholds.

From these two figures, we could see a general fact that the score threshold has a similar impact on the performance as the minimum confidence, i.e., the higher the score threshold, the higher the precision but the lower the recall. When the score threshold is varied, there shows a tradeoff between the precision and the recall.

Another important thing is: in Figure 5.4, we achieve a precision of 0.77 with a recall of 0.53 for the score threshold equal to $0.01 * rule\_num$. Both the precision and the recall are better than those with the minimum confidence of 100% and the score threshold of 0.15 (Section 5.2.2). Please notice that this can not be achieved by using the minimum confidence of 100%, because there is always a tradeoff between



	0.0025*rule_num	0.005*rule_num	0.01*rule_num	0.015*rule_num	0.02*rule_num
◆ Accuracy	0.722083	0.723359	0.712315	0.698719	0.689147
■ Precision	0.715154	0.743149	0.77134	0.789597	0.806347
▲ Recall	0.655003	0.606528	0.529909	0.467844	0.424184

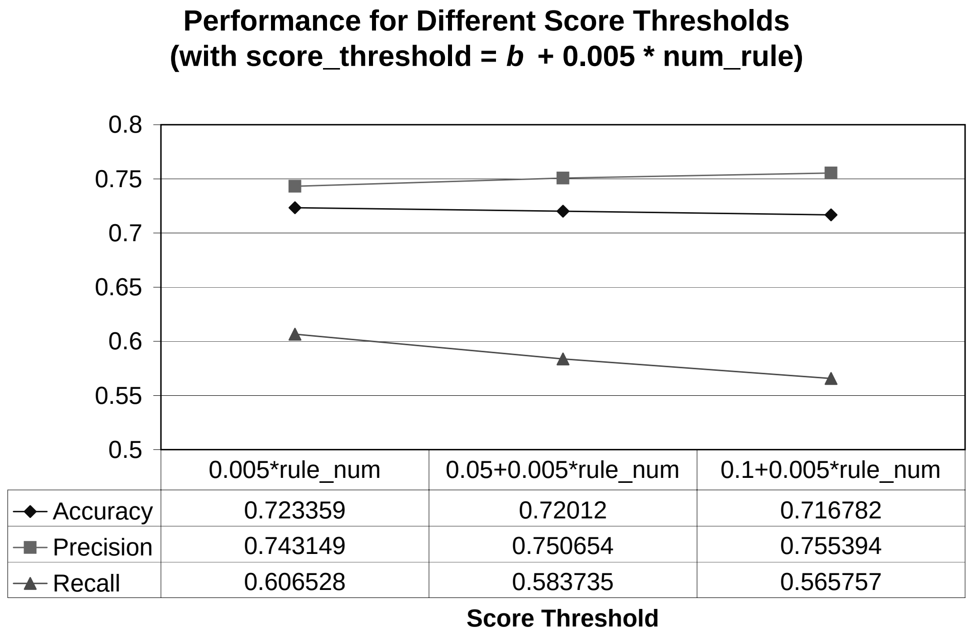
Figure 5.4: Performance for Different Score Thresholds I

the precision and the recall. This verifies our statement that a better accuracy can help us to find a more promising performance.

Performance Distribution

Noticing that with a score threshold of $0.02 * rule\_num$ we could get a very high precision, we are interested in understanding how many articles we have recommended to each user with this score threshold. So we draw the distributions of the number of target users over each precision interval and recall interval of length 0.1. Figure 5.6 presents the distribution with a score threshold of $0.02 * rule\_num$, and Figure 5.7 presents the distribution with a score threshold of $0.005 * rule\_num$.

When choosing a score threshold of $0.02 * rule\_num$, many users receive very low recalls. The situation is much better for a score threshold of $0.005 * rule\_num$.



	0.005*rule_num	0.05+0.005*rule_num	0.1+0.005*rule_num
◆ Accuracy	0.723359	0.72012	0.716782
■ Precision	0.743149	0.750654	0.755394
▲ Recall	0.606528	0.583735	0.565757

Figure 5.5: Performance for Different Score Thresholds II

Different Like Ratios

Obviously, it is easier to do recommendation if a user's prior probability of liking an article is high. To understand by how much this will affect the performance, we draw a histogram on the average precision of users with different ranges of like ratios, which reflects a user's prior probability of liking a movie. The results are shown in Figure 5.8.

Even though the higher the like ratio, the higher the precision. Our recommendations are always better than random recommendations, whose probability of success is equal to the user's prior probability of liking an article. Also, from here we could predict the performance of choosing different *like* thresholds to map a rating to *like* and *dislike*.

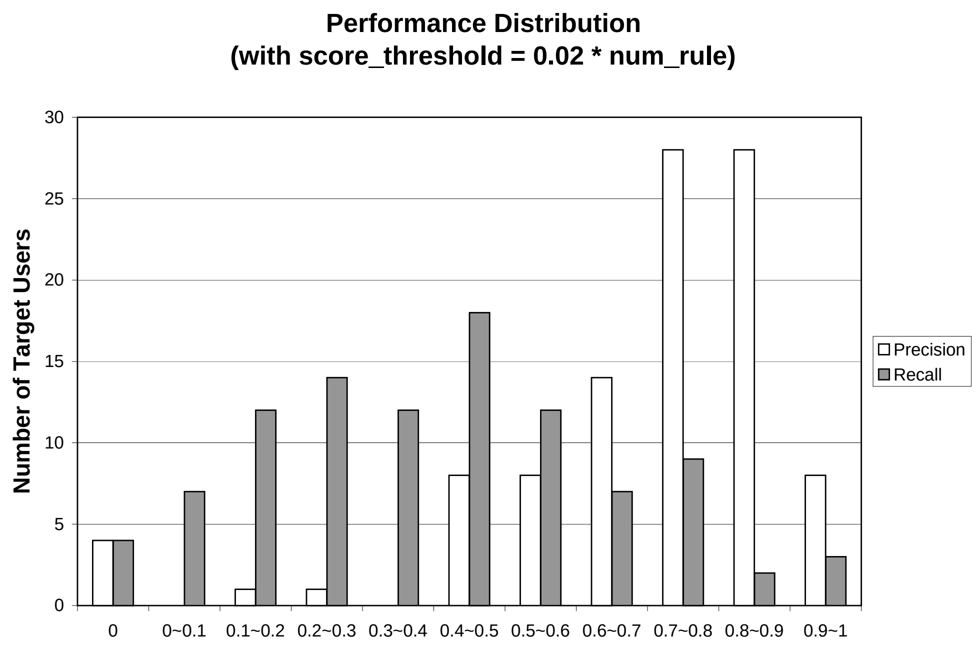


Figure 5.6: Distribution for Number of Target Users with Score Threshold = 0.02

Like and *Dislike* Associations

In order to obtain both *like* and *dislike* associations, we map a rating into *like* and *dislike* by using two thresholds: the *like* threshold and the *dislike* threshold. If a rating is greater than the *like* threshold, then it is mapped to *like*; Otherwise, if a rating is lower than the *dislike* threshold, it is mapped to *dislike*. So the *like* associations we discussed before can be considered as choosing the *dislike* threshold as 0. Figure 5.9 gives the comparison of the performance for different *dislike* thresholds.

We can not see significant difference between the performance for different *dislike* thresholds. So employing *like* and *dislike* associations does not outperform only employing *like* associations.

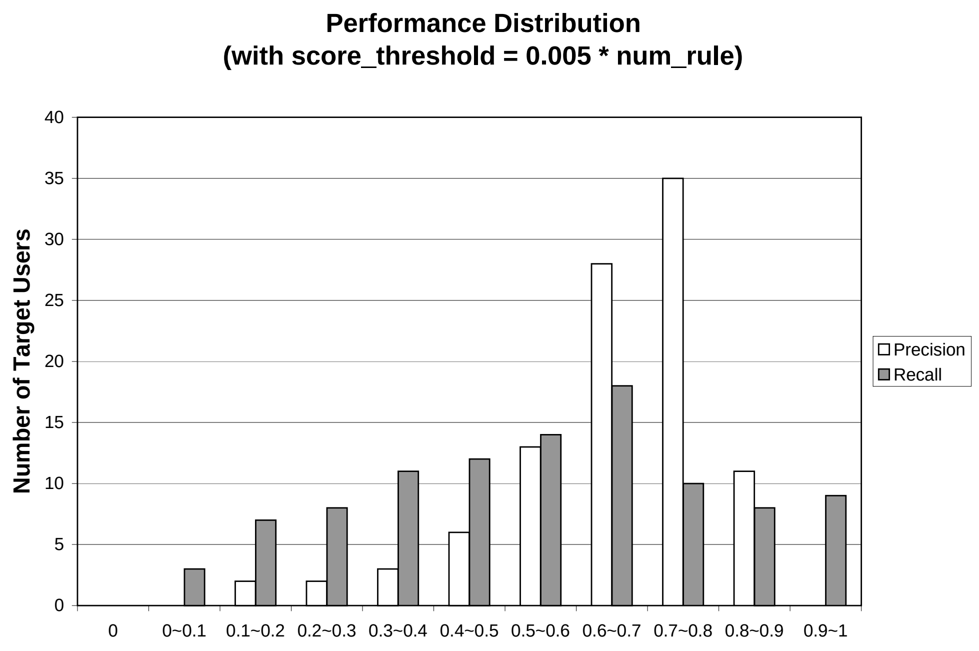


Figure 5.7: Distribution for Number of Target Users with Score Threshold = 0.005

5.2.3 Article Associations

In the experiments for article associations, we tested the performance for different support cutoffs, i.e., we only use rules above a specified support cutoff for recommendation during a test. The results are shown in Figure 5.10 and Figure 5.11.

We can see that the performance of article associations is not as good as that of user associations. The best accuracy we can achieve is 0.68. But one advantage of article associations is that the mining process can be offline and the whole recommendation process takes very little time.

5.2.4 Combining User and Article Associations

Even though the performance of article associations is not as good as that of user associations, we still feel it may be worthwhile to combine user associations with article associations. From our observations, we found that when a target user's

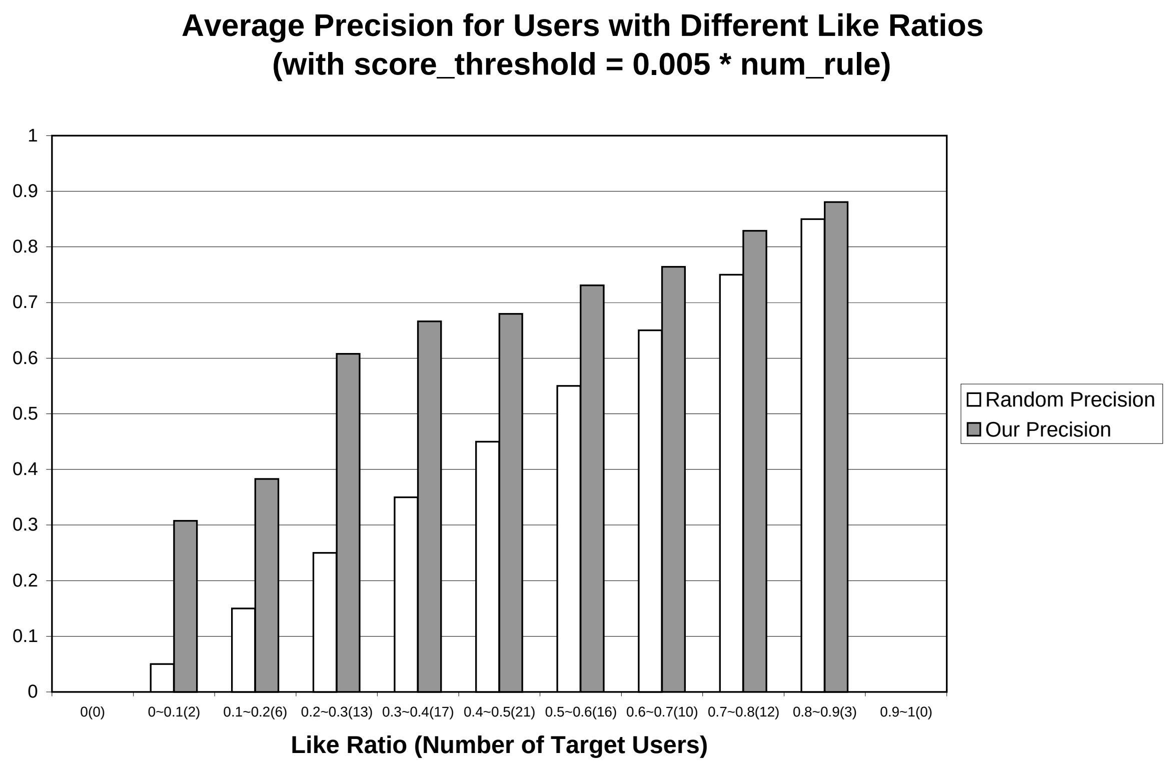
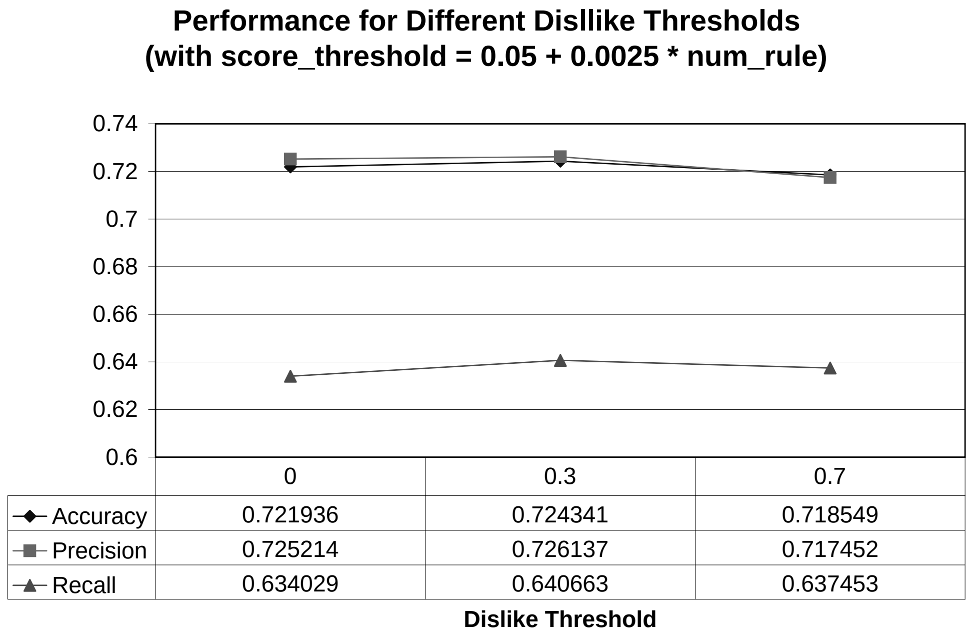


Figure 5.8: Precision for Users with Different Like Ratios

minimum support determined by the mining process is very low, it takes a very long time to mine rules for this user and at the same time the performance of using those rules for recommendation is very bad. So we tried the following strategy to combine user and article associations: If a user's minimum support is greater than a threshold, then we use user associations for recommendation, otherwise we use article associations for recommendation. Table 5.2 lists the performance for user associations, article associations and a combination of the two. We can see that by combining these two associations, the performance decreases a little bit, but we achieve a much faster response time, which makes the system realtime. So we think it is a good strategy to combine these two associations.



	0	0.3	0.7
◆ Accuracy	0.721936	0.724341	0.718549
■ Precision	0.725214	0.726137	0.717452
▲ Recall	0.634029	0.640663	0.637453

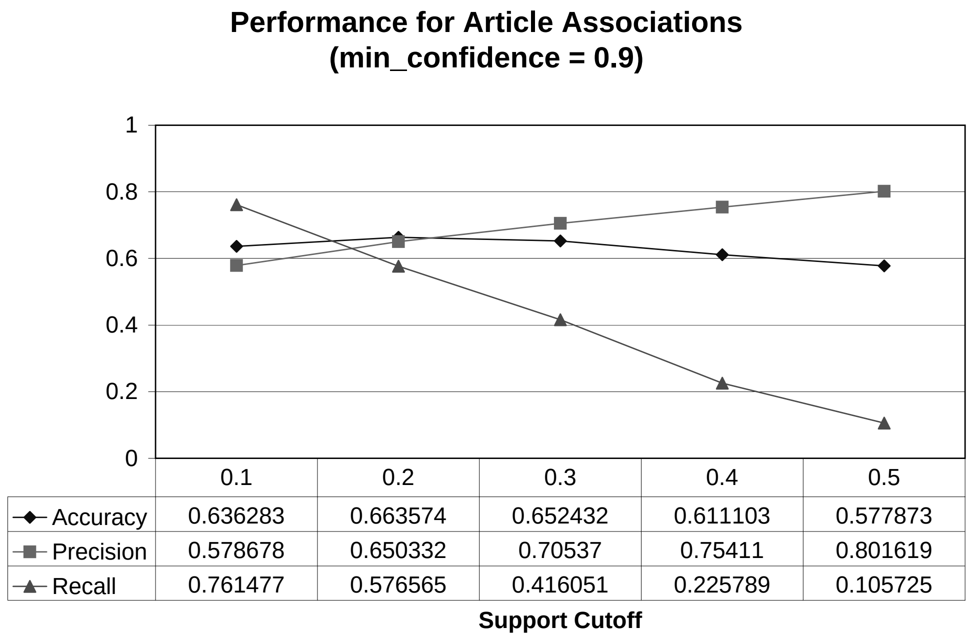
Figure 5.9: Performance for Like and Dislike Associations

	User Assoc	Article Assoc	Combined	Combined
Threshold			0.075	0.1
Accuracy	0.720	0.611	0.717	0.712
Precision	0.751	0.754	0.745	0.723
Recall	0.584	0.226	0.582	0.602
Avg. Runtime	14.2s	0.06s	5.2s	4.6s

Table 5.2: Combining User and Article Associations

5.3 Comparison with other systems

Billsus and Pazzani [BP98] have tested the performance of three collaborative recommendation techniques on EachMovie dataset, which include the correlation-based method, neural networks paired with Information Gain and neural networks paired with Singular Value Decomposition. In their experiments, they choose the first 2000 users as the collaborative users, another 20 random users whose like ratios are less than 0.75 as target users, and 50 random movies as training movies for each target user. The accuracies that these three approaches achieved are listed in Table 5.3.



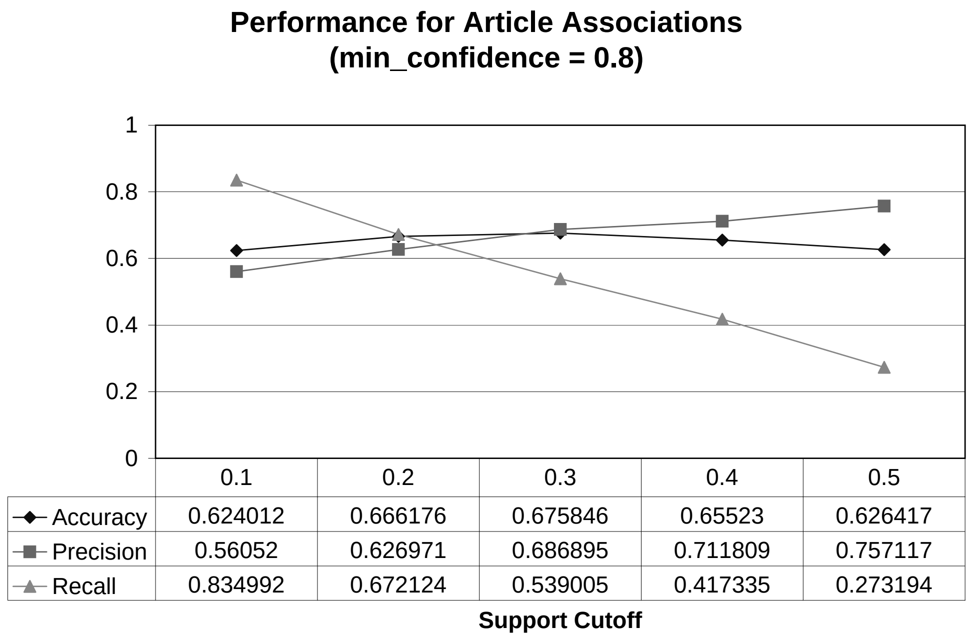
	0.1	0.2	0.3	0.4	0.5
◆ Accuracy	0.636283	0.663574	0.652432	0.611103	0.577873
■ Precision	0.578678	0.650332	0.70537	0.75411	0.801619
▲ Recall	0.761477	0.576565	0.416051	0.225789	0.105725

Figure 5.10: Performance for Article Associations I

	Correlation	InfoGain/ANN	SVD/ANN
Accuracy	0.644	0.67	0.679

Table 5.3: Accuracy for Three Collaborative Approaches

In order to compare our approach with those approaches, we tested our approach under similar experimental conditions. Firstly, we chose the same collaborative user group as their tests, i.e., the first 2000 users in the database. Then, we chose 91 random users whose like ratios are less than 0.75 and who have rated 50 to 100 movies as target users and we employ the *4-fold* cross-validation approach for our tests, which results in that the average number of training movies for each user is 53. The accuracy that our user associations achieved is 0.674, and the average response time is 0.55s. So we can see our approach is at the same level of the current state-of-the-art techniques.



	0.1	0.2	0.3	0.4	0.5
◆ Accuracy	0.624012	0.666176	0.675846	0.65523	0.626417
■ Precision	0.56052	0.626971	0.686895	0.711809	0.757117
▲ Recall	0.834992	0.672124	0.539005	0.417335	0.273194

Figure 5.11: Performance for Article Associations II

5.4 Summary

Some interesting results obtained through the experiments are summarized as follows:

- Our experiments verify our assumption that a limited number of rules is desirable for making recommendations to a user. Too many rules are not necessary and might be even problematic.

- The minimum confidence of rules has a great impact on the performance. This is not surprising because the confidence of a rule corresponds to the average precision of using this rule to recommend training articles to the user.

- It is not necessary to exploit both like and dislike associations. It doesn't improve the performance but increase the running time.

- We can achieve a realtime response and still good performance by combining user associations and article associations.

- Our approach is at the same level of the current state-of-the-art collaborative recommendation techniques.

Chapter 6

Conclusions and Future Work

The main contribution of this thesis is the new framework it presents to apply association rules for collaborative recommender systems. Through our extensive experiments, we see the patterns of the influence of different parameters on the performance of the system. Our experiments verify our hypothesis that a limited number of rules is desirable for making recommendations to a user. Too many rules are not necessary and might be even problematic. The confidence of a rule has a great impact on the performance, but the highest confidence, or a confidence of 100%, is not the best. By choosing a relatively high confidence, for example 85%, and an appropriate score threshold, we can achieve a better performance. Also, it is not necessary to exploit both like and dislike associations, because it does not improve the performance but increases the running time. Under similar experimental conditions, the performance of user associations is better than that of article associations.

We have found that our new approach can satisfy the realtime recommendation requirement (especially when user and article associations are combined), and can achieve very good performance, which is at the same level of the current state-of-the-art collaborative recommendation techniques.

We have also presented a new algorithm for mining association rules with a specific target predicate in the heads of the rules. Unlike most existing association rule mining algorithms, which require that the minimum support of the rules to be mined be specified in advance, our algorithm adjusts the minimum support during the mining process so that the number of rules generated lies within a specified range. This keeps the running time under control, and ensures that enough rules are available for the target predicate.

We only tested our approach on the EachMovie data set, which has been widely used as a testbed for recommender systems. It will be interesting to see the performance of our approach on some other data sets. In our recommendation strategy, we assign a score to a rule as the product of the confidence and the support of the rule. We may evaluate the quality of a rule or the quality of an article by using some other ranking mechanism, which can give accurate top 3 or top 10 recommendations. In our approach, we only use association rules to exploit the collaborative information. Actually, association rules can also be used to exploit the content-based information. We may achieve better performance by exploiting both. This is left as future work.

Bibliography

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993. ACM.

[AS94a] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, Santiago, Chile, 1994.

[AS94b] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. Technical Report RJ 9839, IBM Almaden Research Center, San Jose, California, June 1994.

[BHK98] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, WI, July 1998.

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 255–264, New York, May 1997. ACM, ACM Press.

[BP98] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proc. of the Fifteenth International Conference on Machine Learning*, Madison, Wisconsin, 1998. Morgan Kaufmann Publishers.

[BS97] Marko Balabanovic and Yoav Shoham. Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, March 1997.

[CMS97] R. Cooley, B. Mobasher, and J. Srivastava. Grouping web page references into transactions for mining world wide web browsing patterns. Technical Report TR 97-021, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA, June 1997.

[CPY98] M.-S. Chen, J. S. Park, and P. S. Yu. Efficient data mining for path traversal patterns. *IEEE Trans. on Knowledge and Data Engineering*, 10(2):209–221, March 1998.

[CSM97] R. Cooley, J. Srivastava, and B. Mobasher. Web mining: Information and pattern discovery on the world wide web. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, November 1997.

[FBH00] Xiaobin Fu, Jay Budzik, and Kristian J. Hammond. Mining navigation history for recommendation. In *Proceedings of the 2000 international conference on Intelligent user interfaces*, pages 106–112, New Orleans, LA, January 2000. ACM.

[HH77] Petr Hájek and Tomas Havranek. On generation of inductive hypotheses. *Int. J. Man-Machine Studies*, 9:415–438, 1977.

[HHC66] P. Hájek, I. Havel, and M. Chytil. The guha method of automatic hypotheses determination. *Computing*, 1:293–308, 1966.

[LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 80–86, New York, August 1998.

[McJ97] P. McJones. Eachmovie collaborative filtering data set. http://www.research.digital.com/SRC/eachmovie, 1997. DEC Systems Research Center.

[RIS+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW94)*, pages 175–186, 1994.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In *Proc. of the 1996 ACM SIGMOD Conference on Management of Data*, Montreal, Canada, June 1996. ACM.

[SM95] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the Conference on Human Factors in Computing Systems (CHI95)*, pages 210–217, 1995.