.

AdRec:
An Adaptive Recommender System

John O'Donovan, BSc

A thesis submitted to University College Dublin for
the degree of MSc

February 2005

Department of Computer Science,
University College Dublin
Belfield,Dublin 4.

Head of Department: Professor Barry Smyth

Supervisor: John Dunnion

Abstract

Information filtering techniques are becoming more widely used as available information spaces grow exponentially larger. New techniques for filtering information are being developed to tackle the information overload problem. This thesis presents an assessment of the performance of three popular recommendation algorithms over a range of diverse data. Our aim is to show that the relative performance of these algorithms varies as they are applied to different data, and that these differences can be harnessed to develop an Adaptive Recommender System.

We classify our datasets based on a set of their salient features, including user-item ratio, rating distribution, data type, and sparsity of the data. We run performance tests on Item-Based Collaborative Filtering, Pure Collaborative Filtering and an Association-Rule Based Filtering algorithm using four experimental datasets. These tests consider neighbourhood size and density of the training set.

Based on the results of our tests, and the values each set produces for the classification metrics, we develop a regression function for the purpose of predicting the suitability of a particular recommendation algorithm to a dataset previously unseen by the system. This is done using only the new dataset's classification values. Our empirical results show that there is a significant difference in the relative performance of these algorithms over the four data platforms, and that the resulting regression function correctly predicted the best-performing algorithm for the new dataset.

Acknowledgements

This has been an introduction to what will hopefully be an interesting research career. I would like to thank all in the office for putting up with my bad jokes. John Dunnion for making this research possible, and my parents for getting me here in the first place.

List of Publications

- O'Donovan, John and Dunnion, John (2003) "A Comparison of Collaborative Recommendation Algorithms over Diverse Data Types" *14th Irish Conference on Artificial Intelligence and Cognitive Science*, Dublin 2003.

- O'Donovan, John and Dunnion, John (2004) "A Framework for the Evaluation of Collaborative Recommendation Algorithms in an Adaptive Recommender System" *In Proceedings of the International Conference on Computational Linguistics (CICLing-04), Seoul, Korea.*

- O'Donovan, John and Dunnion, John (2004) "Adaptive Recommendation: Putting The Best Foot Forward" *In Proceedings of 3rd International Symposium on Information and Communication Technologies (ISICT)*, Las Vegas, Nevada. USA. June 16-18 2004.

- O'Donovan, John and Dunnion, John (2004) "Evaluating Information Filtering Techniques in an Adaptive Recommender System" *In Proceedings of 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, Eindhoven, The Netherlands. August 23-26 2004.

- O'Donovan, John and Dunnion, John (2004) Mheitifhoglaim do mholadh dhearadh innill. *In John O'Donovan, editor, Sruth Gaeilge sa 15u Comhdhail ar an Intelacht Shaorga agus ar na hEolaiochtai Cognaiochta (AICS 04)* Caislean an Bharraigh, Contae Mhaigh Eo, Eireann., pages 18-20, September 9-12 2004.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The amount of information that is available on the Web is becoming more vast each day. Extracting and filtering this information to our own particular requirements and tastes is now an essential task.

We all employ a range of methods to acquire the information which we use to make our daily decisions. Throughout history, people have relied on methods such as soliciting the opinions of trusted friends to make their decisions. People also rely somewhat on luck in their decision-making process. Interesting and useful items can sometimes be happened upon completely by accident, resulting in much timesaving, since there is no need to trawl through irrelevant information to find what is required. This decision-making technique however, can obviously not be relied upon.

The collaborative or social filtering process that has been in use for so long on an informal human basis can be automated to cope with the growing problem of information overload. For example, a person may consult a friend with whom they share similar tastes in movies as a basis for deciding whether or not to see a new film. We rely on a news-team to bring us interesting reports, and on DJs and music guides to provide us with information on the songs most suited to our tastes.

One of the goals of advertising is to increase the trust of customers in a particular product. For each successful recommendation we receive, its source becomes more trusted. This trust means that a future recommendation will be assigned more weight in our decision making.

1.0.1 Information Overload

Information Overload is a relatively new concept. As our available information space grows ever larger, thanks largely to the proliferation of the Web, we have to

face the task of tailoring our information spaces to meet our individual needs. A significant portion of current AI research is dedicated to solving the information overload problem. This research can be categorised into two main streams: firstly, information filtering which is a passive approach to tackling the information overload problem, in that the the filters work in the background without any real effort from a user. An exact definition of information filtering has not yet been properly defined. A prominent AI researcher, Douglas Oard, gives the following definition: "Generally, the goal of an information filtering system is to sort through large volumes of dynamically generated information and present to the user those which are likely to satisfy his or her information requirement." *

The second category is information retrieval (IR). IR is a more pro-active approach to solving the information overload problem. Usually, an IR system is considered to have the function of "leading the user to those documents that will best enable him/her to satisfy his/her need for information" [5]. Oard also provides two insightful viewpoints for the comparison of information filtering to information retrieval: " If one considers information retrieval from a very general 'information selection' viewpoint, information filtering is simply a special case in which the information space is very dynamic. If, on the other hand, your personal definition of information retrieval involves selection of relatively static information in response to relatively dynamic queries, then information filtering is best viewed as the dual problem to information retrieval." †

1.0.2 Recommender Systems

Recommender systems are an emerging solution to the information overload problem, in the category of information filtering. They are designed to help tailor a user's information space by suggesting items the system believes will be of interest to the user, based on some comparison of that user's rating history to other users' rating histories. An increasing number of online stores provide recommender systems on their sites, e.g. E-Bay‡, Amazon.com§, etc. There are two main bases upon which such systems operate: Collaborative [63] and Content-Based Filters [45]. A Content-Based approach to the recommendation task compares similarities in descriptions of content of items to user profiles to arrive at its goal. In the Collaborative approach, which is the focus of this thesis, users provide ratings for items in a particular domain, and the system exploits similarities and differences

*http://www.ee.umd.edu/medlab/filter/filter_definition.html
†http://www.ee.umd.edu/medlab/filter/filter_definition.html
‡http://www.ebay.com/
§http://www.amazon.com

between users based on these items to compute its recommendations: If users A and B rate k items similarly, they share similar tastes and should rate other items similarly. Collaborative Filtering (CF) techniques do not require items to be machine-analysable, and can arrive at serendipitous recommendations, that is, they can recommend relevant items that are completely different from those in a user's profile. They also require little knowledge-engineering overhead [56]. There are many implementations and variants of collaborative filtering techniques in today's recommender systems but all are restricted by the same fundamental drawbacks:

- *Sparsity Problem* [56]: In any given case, it is unlikely that two users have co-rated many of the items in the system. Accurate similarity measurements depend on rich user profiles with high overlap which can be costly to attain;
- *Latency Problem* [56]: This affects new, uncommon, or esoteric items. These items will not be recommended by a system until they are included (if ever) in a sufficient number of user profiles [56].

1.1 Motivating Example

	Frank Sinatra	The Strokes	David Bowie	Metallica	Norah Jones
TOM	5	4	1	⊥	3
SUE	5	⊥	⊥	⊥	-1
JOE	⊥	⊥	⊥	5	4
ED	-5	-3	5	⊥	⊥
MAY	5	4	⊥	-3	2

Figure 1.1: Fictitious ratings for five users on popular music.

Each of the collaborative recommendation techniques covered in this research operates in a different manner on the training data it is provided. The diagram in Figure 1.1 shows a fictional dataset of friends who have provided their preferences for some popular musical artistes. Ratings are on a scale of -5 ('did not like at

all') to 5 ('liked very much'). We introduce this simple example to illustrate the difference in operation of the algorithms used throughout this thesis. It is these differences that provide both the requirement for an adaptive recommender, and the basis of AdRec's approach to its development. The $\perp$ symbol denotes a null rating (ie the person has not rated the item). To illustrate the operation differences between algorithms we will show how each would calculate a rating for user *May* on item 'Bowie'.

A rudimentary, base-level CF system might simply predict the group average for this item. This would be a reasonable estimate, but does not model the fact that we always weight our friends' recommendations based on our sense of trust. This method will also give everyone the same rating: Although Joe and May have different tastes, they will still get a predicted rating of 3 for Bowie ((5+1)/2) using the average rating approach.

Our collaborative algorithms are a little more complex than this, however. A User-Based algorithm will correlate the users and give May a recommendation for 'Bowie' based on the ratings of users who are similar to her. We can see that Tom has rated items similarly to May, and has given a rating of 1 to 'Bowie'. We can use this positive correlation to suggest to May a similar rating of 1 for that item. This rating may be further weighted according to the degree of similarity between May and Tom. With User-Based Collaborative Filtering (UBCF) it is also possible to use negative correlation: Ed has rated similar items to May, but has rated them with an obvious difference in taste. This information can be harnessed to give May a rating based on dissimilarity of taste with Ed: ie a rating of -5 (5 times a -1 similarity value).

The Item-Based Collaborative Filtering (IBCF) algorithm is again more complex. It calculates similarities between items rather than users. The algorithm looks at co-rated items to build a similarity model. This can be seen in Figure 1.2. To provide May a rating for 'Bowie', the algorithm looks at the similarities between 'Bowie' and every other item that May has rated. It then returns the sum of their similarities times May's ratings for those items.

In this example, as can be seen from Figure 1.2, the similar items are 'Sinatra' and 'The Strokes', so May gets a predicted rating for item 'Bowie' of $R_{u,i} = (0.2 \times 0.5) + (0.2 \times 0.4) = 0.18$.

We can see here that these recommendation techniques not only operate differently on the training data, but also provide different ratings. In the next section, we outline how these differences in recommendation techniques can be harnessed to provide the best available recommendations for a particular dataset from a range of available recommendation techniques.

	Frank Sinatra	The Strokes	David Bowie	Metallica	Norah Jones
Frank Sinatra	1				
The Strokes	.4	1			
David Bowie	.2	.2	1		
Metallica	⊥	⊥	⊥	1	
Norah Jones	.2	.4	⊥	⊥	1

Figure 1.2: IBCF Similarities.

1.2 Objectives

Different implementations of collaborative filtering will be affected to varying degrees by the problems inherent in all CF systems. Their performance will change based on the dataset they operate on, and the information they harness to compile a similarity model. For example, for a situation where the set of items to be recommended is relatively small and static, and there are a large number of users, it would be advisable to employ an item-based approach [65][33] to CF, since the similarity model is built up over the large number of user profiles. In this situation a user-based filtering approach, as in [6], would not perform as well since there would be insufficient items in each profile to provide the level of overlap required for a reliable similarity model.

Our objective in this work is to define a technique to adaptively apply the most efficient collaborative filtering algorithm to an individual data source. To achieve this adaptability in our system, we make the assumption that our datasets can be adequately classified (for CF purposes) by a set of their salient features. These include user-item ratio, sparsity and data type, and will be discussed in detail in Section 3.4. We test three collaborative recommendation algorithms (User-Based, Item-Based and Rule-Based CF) and a base line algorithm on four different experimental datasets (EachMovie, PTV, Jester and MovieLens), and note the performance of each method with respect to the classification metrics above. From this information we developed a regression function for algorithm prediction based on the classification metrics alone.

We tested the performance of this function by introducing another dataset

Symbol	Description
$A..Z$	Sets
U	The set of Users
I	The set of Items
$i, j...$	Items
$u, v...$	Users
r_a	The set of Active Users Ratings
r_u	The set of Current Users Ratings
$r_{a,i}$	The Active User's Rating of Item i
$r_{u,i}$	The Current User's Rating of Item i
$\perp$	The null rating

Table 1.1: Document Nomenclature.

which was classified according to the required metrics, and the resulting values were run through the regression function to attain an algorithm prediction. Our thesis is that if we can successfully perform this algorithm prediction task, we can form the basis of a generic recommender system, which can include cutting-edge filtering techniques in a given system without having to manually tailor the recommendation engine for that system.

1.3 Document Outline

This thesis is divided into 6 chapters. Chapter 2 gives a review of the state of the art in collaborative recommender systems, and an overview of the types of recommendation algorithms in this area, focussing on the shortcomings of existing techniques. Chapter 3 details the design of the AdRec system, describing the algorithms and the experimental data used. Chapter 4 deals with implementation issues. Chapter 5 presents experiments and results, and the final chapter contains a summary and a discussion of potential future research.

1.4 Nomenclature

Throughout this document, unless otherwise stated, the symbolic notation in Table 1.1 is used.

1.5 Summary

In this opening chapter, we provided a brief introduction to the problem of information overload, and the recommender systems that are evolving to tackle this problem. We defined the overall objective of this research as the design, development and testing of an adaptive recommender system using regression functions learned from training examples. Using a fictitious example, we introduced some of the recommendation algorithms that will be dealt with later in the thesis.

Chapter 2

Background Research

2.1 Introduction

This chapter provides a detailed literature review of the state of the art in personalisation research, with a specific focus on recommender systems. We review some of the main ideas of personalisation, and overview some existing personalisation and recommender systems. We examine the corpus of algorithms used for recommendation, looking in detail at the drawbacks and benefits of each.

2.2 Personalisation

Recommender systems are a means of performing personalisation. These systems fall under the general heading of Adaptive Hypermedia, which can be defined as systems that can adapt to a user's tastes and requirements. The goal of a personalisation system is to tackle the information overload problem by tailoring the information space of each user to his/her particular tastes, whether common or esoteric. One of the biggest problems that is currently facing personalisation systems is the eternal personalisation-privacy trade-off. Users want personalised information, but are they willing to give up some personal information in order to get this quality service? Research in [72],[60] and [78] examines this problem.

Section 2.4 provides a look at three example personalisation systems in detail. We will look briefly at some others in this section for the purpose of comparison.

PTV [75][74] and [73] is a personalised TV listings service, providing users with personalised TV guides based on their learned viewing preferences. It employs a hybrid recommendation strategy using both content-based and collaborative methods. Its server uses explicit user ratings on various programmes, channels and viewing times to build a profile on the user. Personalised recommendations

are then based on this learned profile.

Letizia [41], makes a representation of a user's interests based on the Web pages they visit and actions they perform in this process. The goal of this personalisation is to aid a user in browsing the Web. To this end Letizia browses the Web 'in sync' with a user. As the user looks at a Web page the system can then 'jump' ahead of the user by following hyperlinks on the current Web page and subsequent Web pages in an effort to find interesting pages for the user. Any interesting paths the system finds for the user can then be suggested when requested by the user. In this way Letizia helps with a user's Web journey. Personal Web Watcher (PWW) [40] and Letizia are content-based systems that recommend Web-page hyperlinks by comparing them with a history of previous pages visited by the user. Letizia uses the spare CPU time when the user is actually reading a Web page to follow the adjoining hyperlinks. A user's profile is composed of keywords extracted from the visited pages. PWW uses an offline period to generate a bag of words-style profile for each of the pages visited during the previous browsing session. Hyperlinks on new pages can then be compared to this profile and graded accordingly.

CASPER [61] [8] is a case-based personalisation system for an online recruitment environment. CASPERs Personalised Case Retrieval (PCR) component is a two-stage personalised job case retrieval system that combines a server-side similarity-based retrieval component with a client-side case-based personalisation component. This combination benefits from superior cost, flexibility, and privacy characteristics. It shares many of the same motivations and objectives as PTV and the systems in Section 2.4. Like GroupLens (see Section 2.11.2) and PTV, CASPER maintains long-term user profiles, and like PWW and Letizia it implements a client-side personalisation policy. PWW, Letizia, PTV and CASPER systems all use a content-based approach to personalisation. CASPER's PCR uses a structured case-base profile representation, unlike the others, which use unstructured keyword-type profiles.

2.3 Information Filtering

The ultimate goal of an information filtering system is to sort through large volumes of dynamically-generated information and present to the user those which are likely to satisfy his or her information requirement.

In order to sharpen this definition, a distinction should be drawn between information retrieval and information filtering. In some domains (e.g. USENET News)* the retrieval effort is minimal because the information is directed to the

*http://groups.google.com/

user. In other domains (e.g. the World Wide Web) the retrieval effort can be considerable because no mechanism exists to draw new information to the attention of a filtering system.

2.4 Recommender Systems

Recommender systems are a recently conceived personalisation tool for tackling the information overload problem by attempting to tailor each user's information space to meet their individual information needs. Recommender systems have been deployed across many different domains, from movie and book recommenders to people recommenders. To date, most recommendation systems are based on one or more of a set of standard recommendation algorithms. (or hybrids of these). Standard recommendation algorithms include the class of collaborative recommendation algorithms, (discussed in the following section) and the more basic class of content-based approaches to recommendation, which we touch on in section 2.7.1.

2.5 Collaborative Filtering

Collaborative filtering models the social process of asking trusted friends for their opinions or 'recommendations' on items, for example, asking a friend for his opinion about a movie he has seen.

Automated Collaborative Filtering (CF) was first introduced in 1989 by John Hey of LikeMinds, and furthered in 1994 by the GroupLens [63] research group. The task of a CF system is to predict items that a particular user will find useful. For the remainder of this document we will refer to the user receiving recommendations as the *active user*. Predictions are based on a database of other users' votes on the set of recommendable items. CF systems can be categorised into two main streams: Memory-Based CF, which operates over the entire database to make a prediction in a totally online process, and Model-Based CF, which uses the database to compile a similarity model in an offline process.

As stated by Herlocker et al. [30], "collaborative filtering provides three key additional advantages to information filtering that are not provided by content-based filtering." Collaborative filtering allows for filtering on data that is hard to analyse by automated processes, such as movies and feelings, as it is the human user who rates the data and from this the relevance of the data to other users is computed. Also, with collaborative filtering there is the ability to filter items based on quality and taste, as humans are capable of analysing on such dimensions

Action	Notes
purchase (price)	buys item
assess	evaluates or recommends
repeated use (number)	e.g. multiple check-out stamps
save/print	saves document to a personal storage
delete	deletes an item in a shopping basket
refer	cites or otherwise refers to an item
reply (time)	replies to an item
mark	add to a 'marked' or 'interesting' list
examine/read (time)	looks at a whole item
consider (time)	looks in abstract
glimpse	sees title/surrogate in list
associate	returns in search
query	association of terms from queries

Table 2.1: Methods for Non-Invasive Inference of User-Ratings.

while it is hard for computers to do so. Finally, collaborative filtering systems can give serendipitous recommendations. Herlocker et al found that "serendipitous recommendations occur frequently in the movie domain, with the collaborative filtering system accurately recommending movies that a user would never have considered otherwise".

Another common categorisation of CF systems is the manner in which they ascertain user ratings. There are two main rating types: Implicit and Explicit. Implicit ratings are a passive, non-invasive method of taking information from users, which has the advantage of not bothering the user with tedious questions about their tastes which are generally in the form: "rate the following items...".

Explicit rating schemes have advantages in that the user tells the system exactly what tastes they have, and they may consequently feel a sense of interaction with the system, and therefore an increased sense of trust. However, user tastes are changeable, and users may not be inclined to update their preferences. Also, there is a danger that the time required to input ratings may outweigh the user's attention span. It has been shown in [60] that the accuracy of inferred ratings from implicit ratings is in line with those from explicit ratings.

Table 2.1, from Nichols [22], outlines some of the methods used to infer user ratings non-invasively.

2.5.1 Memory-Based Collaborative Filtering

Memory-based algorithms utilise the entire user-item database to generate a prediction. These systems employ statistical techniques to find a set of users, known as neighbours, that have a history of agreeing with the target user (i.e. they either rate different items similarly or they tend to "buy" similar sets of items). Once a neighbourhood of users is formed, these systems use different algorithms to combine the preferences of neighbours to produce a prediction or top-N recommendation for the active user. The techniques, also known as nearest-neighbour or user-based collaborative filtering are the most popular form of CF, and are widely used in practice. See [60] [52] [32] and [63] for example.

2.5.2 Model-Based Collaborative Filtering

At a general level, model-based collaborative filtering algorithms provide item recommendations by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings on other items. The model building process is performed by various machine learning algorithms such as Bayesian network [62], clustering [64], and rule-based approaches [56]. The Bayesian network model formulates a probabilistic model for collaborative filtering problem. The clustering model treats collaborative filtering as a classification problem and works by clustering similar users into the same class and estimating the probability that a particular user is in a particular class C, and from there computes the conditional probability of ratings. Our baseline algorithm, Zero-r, operates on this principle. The rule-based approach applies association rule discovery (Section 2.5.5) algorithms to find associations between co-rated items and then generates item recommendation based on the strength of the association between items, as in [33] and [67].

2.5.3 User-Based Collaborative Filtering

User-based Collaborative Filtering algorithms are a subset of the *memory-based* class of algorithms. Our User-Based CF is performed by calculating the Pearson correlation coefficient [9] of two vector representations of users, using this to define peer-groups, and suggesting items a user's peers liked that have not yet been seen by the user. As detailed earlier, there are a range of methods for calculating similarity coefficients, such as Cosine Similarity 2.9.1, Jaccard's coefficient [35], etc. We use Pearson's correlation as it is the most widely used in CF and allows

us to make a more direct comparison with other existing systems. When the peer-group of users is chosen based on the similarity to the active user (the user receiving recommendations), a weighted combination of their ratings is used to generate predictions, as seen in [45]. We next examine the user-based CF algorithm in terms of how it builds its model and calculates predictions. We will repeat this examination for all of our recommendation algorithms.

Model-Building and Similarity Capture

The following is a step-by-step outline of the CF algorithm:

1. Calculate the similarity between the active user and every other user.
2. Form the peer group by selection of the top-n most similar users.
3. Select items rated highly in the peers' profiles which are not in the active users profile.
4. Return a ranked list of these items as the recommendation set.

The following formula calculates the Pearson Correlation in the first step:

$$corr_{x,y} = \frac{\sum_{u \in U}(R_{k,x} - \overline{R}_x)(R_{k,y} - \overline{R}_y)}{\sqrt{\sum_{u \in U}(R_{k,x} - \overline{R}_x)^2 \cdot \sum_{u \in U}(R_{k,y} - \overline{R}_y)^2}}$$

where $corr_{x,y}$ is the Pearson correlation coefficient between user x and user y, $R_{k,x}$ is the rating of item k by user x and $\bar{R}_x$ is the average item rating by user x.

Prediction Calculation

The prediction calculation is done using the following formula, which calculates the weighted average of deviations from the mean of the neighbours:

$$p_{x,i} = \overline{r}_x + \frac{\sum_{u \in U}(r_{u,i} - \overline{r}_u) \times P_{x,u}}{\sum_{u \in U}(P_{x,u})}$$

where $p_{x,i}$ is the predicted rating of item i by user x, and $p_{x,u}$ is the similarity between users x, and u.

2.5.4 Item-Based Collaborative Filtering

Item-based collaborative filtering algorithms are a general subset of the *model-based* class of CF algorithms. The first real implementation of an item-based CF algorithm was shown in [19]. Item based CF algorithms use a pre-computed model based on the items in the system rather than the users (as with user-based CF systems). Historical information on items in the system is analysed to ascertain relations between items. An example of this might be "the purchase of one item often leads to the purchase of another item." This type of filtering can be extremely beneficial to a system with a relatively static number of items, eg McDonalds restaurants, which have millions of users and a small, static number of items. For example, using IBCF, we can easily spot that people who purchase fries also purchase a coke, and use this information in our recommendation process. This technique is dealt with in detail in [32] and [33]. It can be used in tandem with other filters to overcome some of their inherent shortcomings. Recent research in [65] shows that this approach can be as accurate as the common user-based approach (in [63] for example) while the speed of the item-based approach is much greater.

The main difference between Item-Based CF (IBCF) [65] and the User-Based algorithm is that IBCF makes its predictions based on a model of item similarity rather than user similarity. The IBCF algorithm looks at a set of items the active user has rated and computes their similarity to a target item i, in order to evaluate it for recommendation. The IBCF algorithm selects the k-most similar items $\{i_1, i_2, ..., i_k\}$ and their corresponding similarity values $\{s_1, s_2, ..., s_k\}$. The prediction is found by taking the active user's ratings on these similar items, and weighting these by multiplying them by their similarity values. Again, the algorithm boils down to two main phases: Phase 1 is the Model Building phase and Phase 2 is the Prediction Calculation phase.

Model Building and Similarity Capture

The Model Building phase involves computing similarity between items. Our particular implementation uses a correlation-based similarity metric, again calculated using Pearson correlation.

To make this algorithm as efficient as possible, we use the cases where items i and j have been co-rated by users. In Figure 2.1, using Pearson's equation from Section 2.5.3, the set U consists of only those users who have rated both item i and item j in the diagram, in this case, users 0 and 3. This calculation results in an $n \times n$ matrix of item similarities, where n is the number of items in the dataset.

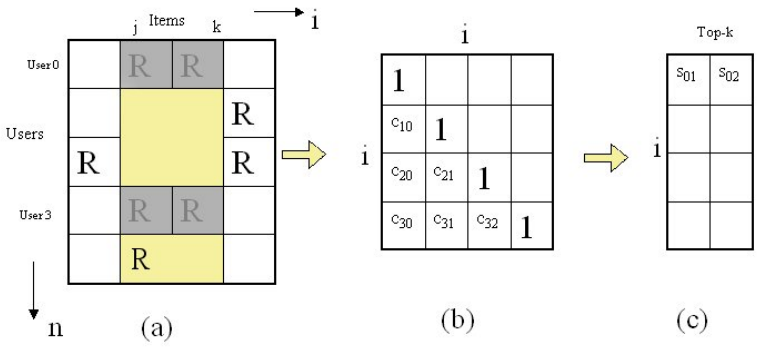


Figure 2.1: Offline Computation of the Item-Based Model.

Since we are generally only interested in the top-n most similar items to each other item, we can sort each item profile in non-increasing order, and store only the top-n from this list as our model. An important advantage of this method is that the entire model-building phase is an off-line computation, resulting in rapid recommendations for online users. This method is also scalable to any number of users, as their recommendations are generated from a static item-item model.

Prediction Calculation

The prediction phase of the IBCF algorithm is done as follows:

1. User rates a set U of items.
2. A Candidate set C of items is formed by taking the union of the k most similar items for each item $j \in U$. (This is done by simply by taking the top-k from the ordered lists computed in the model-building phase).
3. Any duplicate items are removed, ie any $j \in (C \cap U)$.
4. Find the similarity between each item $c \in C$ and the set U. This is done by consulting the Item-Item correlation matrix from phase 1, and summing the correlation values between each $c \in C$ and every item in U.
5. C is then sorted in non-increasing order w.r.t these similarities.
6. The top-n items in C are the Recommendation Set.

2.5.5 Collaborative Filtering using Mined Association Rules

The possibility of mining frequent itemsets over "market basket data" was introduced by Agrawal et al. in [2]. This technique is a case-based approach to collaborative filtering, as discussed by O'Sullivan in [56]. Case-based reasoning (CBR) is a well researched field of AI which relies heavily on good content descriptions of its 'cases'. [56] examines ways to leverage the CBR technique to perform collaborative filtering. This technique treats profiles in a collaborative filtering system as cases. There are two main factors involved here: similarity computation and recommendation. In the AdRec system, the Case-Based Recommendation component uses data mining [7] to arrive at item similarities and a weighted-sum method in the prediction phase. It has been shown in [7] that the Apriori rule mining algorithm is a successful method of performing similarity computation. The AdRec system implements similarity computation for the association rule-based technique via Apriori and also using Item-Based Collaborative Filtering to get a correlation matrix. This helps overcome some of the inherent problems with sparse datasets, specifically the overlap problem discussed in Section 2.6.2.

The association rule-based CF algorithm is also seen in [72]. It is, in fact, very similar to the IBCF algorithm described in the previous section. The Apriori algorithm [2][7] discovers hidden relationships between items by generating rules of the form $A \Rightarrow B$, where A and B are itemsets. We can then use these rules to generate an item-item similarity matrix similar to that in Figure 2.1 above. Association rules of this form have two important metrics: Rule Confidence and Rule Support. The confidence of a rule $A \Rightarrow B$ is the percentage of profiles containing A which also contain B. The support of a rule of this kind is the fraction of the total profiles in a database which uphold the rule:

$$support(A \Rightarrow B) = P((A \cup B) \subseteq T) \tag{2.1}$$

$$confidence(A \Rightarrow B) = \frac{support(A \cup B)}{support(A)} \tag{2.2}$$

This association rule based algorithm is very similar to IBCF in that it contains a model-building phase and a prediction phase.

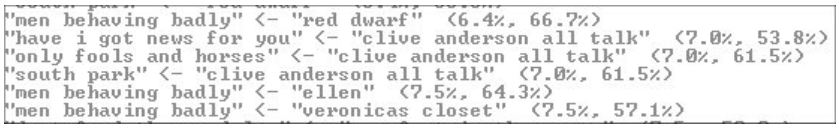


Figure 2.2: Sample Apriori Output for the PTV Dataset

Model-Building and Similarity Capture

Taking each set of items in a profile as an itemset, the Apriori algorithm can derive item-item association rules and confidence levels. As can be seen in Figure 2.2 below, these rules only incorporate binary-chaining, that is, they do not consider rules of the form $A \rightarrow B \rightarrow C$. This type of rule would identify further relations between the items in a dataset, and so can be forwarded as a method to combat the sparsity problem in CF. There is however, a trade-off with this approach: The more tenuous links become between items, the greater the risk of dissimilarity. Recommending a bad item is generally worse than failing to recommend a good item [56].

Prediction Calculation

The prediction phase of this algorithm is the same as in the IBCF algorithm. Thus the only difference between the two algorithms lies in the model-building. Item-Item similarity is derived directly from the association rules in order to compute the model. This is achieved by taking the confidence levels for rules generated by the Apriori algorithm, and using them as probabilities in the Item-Item similarity matrix. There are many ways to combine the probabilities with the support values to attain the Item-Item model. These are discussed in detail in [56]. For our testing, we simply use the rule support multiplied by its probability to arrive at a similarity value. For example in Figure 2.2 the first rule is: "men behaving badly" $\rightarrow$ "red dwarf" is <6.4%, 66.7%>. From this we get a similarity for these two items of 4.26% (or 0.426 for our calculations).

$$sim(A, B) = confidence(A \Rightarrow B) * support(A \Rightarrow B) \tag{2.3}$$

2.5.6 The Apriori Algorithm

The Apriori algorithm is a popular data mining algorithm. Details of its use in the AdRec system as part of a recommendation strategy were described in Section 2.5.5. In this section we describe the algorithm itself.

Association rule induction is a powerful method for so-called market basket analysis, which aims to find regularities in the shopping behaviour of customers of supermarkets, mail-order companies, etc. With the induction of association rules one tries to find sets of products that are frequently bought together, so that from the presence of certain products in a shopping cart one can infer (with a high probability) that certain other products are present. Such information, expressed in the form of rules, can often be used to increase the number of items sold, for instance, by appropriately arranging the products in the shelves of a supermarket (they may, for example, be placed adjacent to each other in order to invite even more customers to buy them together) or by directly suggesting items to a customer which may be of interest to him/her.

An example of an association rule is "If a customer buys wine and bread, he often buys cheese too." It expresses an association between (sets of) items, which may be products of a supermarket or a mail-order company, special equipment options of a car, optional services offered by telecommunication companies, etc. An association rule states that if we pick a customer at random and find out that he selected certain items (bought certain products, chose certain options etc.), we can be confident, quantified by a percentage, that he also selected certain other items (bought certain other products, chose certain other options, etc.). A more formal definition of the Apriori problem itself is follows.

Problem Statement: Let $I = \{i_1, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items. We say that a transaction T contains itemset X, if $X \subseteq T$. Itemset X has support "s" in the transaction set D if no less than s transactions in D contain X. Given a set of transactions D, the problem of mining frequent itemsets is to generate all itemsets that have support greater than the user-specified minimum support.(see Section 2.5.5)

The general procedure for this algorithm is as follows:

- Assume $a_{k,j}$ is the k^{th} value of the j^{th} attribute
- Assume $\bar{s_r}$ is the minimum coverage for a combination of r attributes
- Identify all pairs $(a_{k,j}, a_{l,m})$ where $j \neq m$ (ie different attributes) that have coverage greater than $\bar{s_2}$
- Identify all triplets $(a_{k,j}, a_{l,m}, a_{i,n})$ where $j \neq m \neq n$ that have coverage greater than $\bar{s}_3$
- Once all desired combinations are identified, compute the accuracy for each combination
- Report only those combinations above a user-defined minimum threshold.

In our implementation of this algorithm, only rules of the form $X \Rightarrow Y$ are used. O'Sullivan et al. [56] discuss rule chaining as a method of improving the performance of the Apriori algorithm in case-based recommender systems. This is where one item can indirectly imply another item by incorporating rules of the form $X \Rightarrow Y \Rightarrow Z$.

2.6 Collaborative Filtering Problems

There exists a host of standard problems which all collaborative recommenders must address if they are to produce high-speed and high-quality recommendations. Despite the surge of research interest in collaborative recommenders, these problems have not yet been overcome.

2.6.1 The Sparsity Problem

CF systems require a sufficient number of users to join the system and to provide ratings before the system can provide quality recommendations with good accuracy and coverage. Even with a large membership, each user must provide enough rating information to achieve a reasonable level of profile overlap so that similarity with a peer-group can be ascertained by the system.

It has been shown by Resnick [63] and later by Konstan [68] that at low rating density levels, many users receive recommendations which are below the

standard of non-personalised recommenders. Much research has been carried out into potential solutions for this problem. O'Sullivan et al [56] use case-based recommendation to decrease sparsity in the PTV dataset. Melville and Mooney [45] employ content-based methods to decrease sparsity in the EachMovie dataset, and Goldberg et al. [23] use the Eigentaste algorithm on the Jester dataset. We define sparsity in the AdRec system in a similar manner to that of [56]:

$$1 - \frac{number\ of\ nonzero\ entries}{number\ of\ total\ entries} \tag{2.4}$$

This simple formula always produces a sparsity value between 0 and 1.

2.6.2 The Latency Problem

The latency problem is the lack of sufficient user-ratings on items in a CF system to provide enough overlap for good recommendations. There are a number of influencing factors on the number of user ratings of items. These include the "gray sheep" problem [13], where a user's tastes are unique or esoteric, and therefore there is insufficient overlap to provide recommendations. The *early rater* problem affects new, unique, or esoteric items. It essentially means that an item does not have enough ratings to get recommended by the system. In some CF research, this problem has also been termed the *ramp-up* [79] problem, or the *cold-start* problem [56]. Smyth et al. [72] propose use of the Apriori [7] algorithm as a technique for combatting the latency problem in CF systems.

2.6.3 Scalability Issues in CF

Recently, much CF research has focussed on the other problems in this section, leaving the scalability somewhat on the back-burner. This may be as a result of hardware advances: cheaper memory and faster processors. It may also be a factor that early CF systems (for example the Tapestry system [22]) were relatively small and did not have massive databases to contend with. Scalability in CF systems has now become a vital issue: the Amazon.com database scales to millions of users and items, and the scale of the CDNow [32] and MovieLens [63] systems are also examples of the importance of scalability.

Goldberg's Eigentaste algorithm [23] claims to produce recommendations in linear time to the number of items in the system. However, this involves a lot of offline computation and model-building. Item-Based techniques such as those from Fisk [19], Karypis [33] and Sarwar [67] tend to scale upwards with a lot more ease than common user-based CF implementations, the fastest of which

operate in quadratic time complexity, with a less-than-linear gain in throughput for an incremental use of hardware resources [63]. Solutions such as dimensionality reduction via factor analysis and singular value decomposition have been used to combat the latency problem by Goldberg in [23], and by Billsus and Pazzani in [6], where the singular value decomposition (SVD) of the original user ratings matrix is used to project user ratings and rated items into a lower-dimensional space. By doing this they eliminate the need for users to have co-rated items in order to be predictors for each other.

2.6.4 Privacy Issues in CF

There exists a permanent trade-off in CF, and all personalisation techniques, between the level of privacy and the quality of personalisation. The more a CF system knows about a user, the more equipped it is to provide that user with high quality recommendations.

The development of more general Web standards has solved some CF privacy issues. One solution is the Open Profiling Standard (OPS). which was introduced by the World-Wide-Web Consortium (W3C)[†] as a way of giving control of data back to the user.

The W3C approach to the privacy versus personalisation tradeoff is known as the Platform for Privacy Preferences Project (P3P) which aims to give Internet users more control over their personal information. P3P technology is designed to allow consumers to express their privacy preferences through their browser, which communicates those preferences to Web sites in a machine-readable format. It offers a technological alternative to having consumers read the privacy policy at each site. A users browser would automatically read a sites privacy policy to see whether it meets the users preferences. If a site shares data in ways that go beyond the users preferences, the user can terminate the connection. This new technology, discussed in [32] is expected to change the face of recommendation techniques as we know them today.

2.6.5 Vulnerability of CF to Attacks

Some more recent CF research evaluates the *robustness* of CF algorithms. It is possible that users can make errors and be careless when providing ratings to the system. How well can a CF system withstand misleading and inaccurate ratings? More importantly, how well can a CF system stand up to deliberate, malicious skewing of ratings for some private benefit or other? For example, a

[†]http://www.w3c.org

record company could boost album sales by making a music recommender system recommend particular albums more highly. O'Mahony et al [55] gives a good account of the stability of a CF system in the face of adverse recommendations. In some cases, the cost of skewing the ratings may be extremely prohibitive, ie users may actually have to purchase a product in order to provide a rating. Many CF systems however, have much less security for ratings. Malicious agents such as Web-bots can cause much damage.

O'Mahony et al defines two aspects to robustness, Recommendation *Accuracy* and Recommendation *Stability*. With Recommendation *Accuracy*, the question is, are the products recommended after the attack actually liked? Recommender *Stability* assesses whether the system recommends different products after the attack, (regardless of whether customers like them). These are two distinct, but not independent aspects. For example: If a recommender has 100% accuracy (with or without misleading ratings), then it is completely stable. However, the recommendation policies for a product which nobody likes are stable for both `recommendToAll()` and `recommendToNone()`, yet one policy is always right and the other is always wrong. It has been shown in [55] that a malicious agent can "force a recommender system to behave in an arbitrary fashion, by adding to the ratings matrix enough rogue users to completely dilute the rating preferences of the real users." This represents a 'back door' into many companies' marketing strategies, and could prove very costly for the victims of a CF attack.

Kushmerick [39] formalises robustness in machine learning terms, developing two theoretically justified models of robustness. This type of CF analysis guides us a little closer to a comprehensive theory of collaborative recommendation.

2.7 Other Filtering Techniques

This section provides a summary of filtering techniques which we have not used in the AdRec system. Table 2.2, from Burke [10] gives a good overview of the following recommendation techniques. In this description, **U** is the set of users whose preferences are known, **I** is the set of items over which recommendations might be made, **u** is the active user, ie the user to whom we are making recommendations, and **i** is some item for which we would like to predict **u**'s preference.

2.7.1 Content-Based Filtering

Content-Based recommenders automatically gather a user profile based on the features of the items that user has rated. Content-based methods are there-

Technique	Background	Input	Process
Collaborative	Ratings from **U** of items in **I**	Ratings from **u** of items in **i**	Identify users in **U** similar to **u**, and extrapolate from their ratings of **i**
Content-based	Features of items in **I**	**u**'s ratings of items in **I**	Generate a classifier that fits **u**'s rating behaviour and use it on **i**.
Demographic	Demographic information about **U** and their ratings of items in **I**	Demographic information about **u**.	Identify users that are demographically similar to **u** and extrapolate from their ratings of **i**.
Utility-based	Features of items in **I**	A utility function over items in **I** that describes **u**'s preferences	Apply the function to the items and determine **i**'s rank.
Knowledge-based	Features of items in **I**. Knowledge of how these meet a user's needs	A description of **u**'s needs or interests	Infer a match between **i** and **u**'s need

Table 2.2: Different Recommendation Techniques (from [10]).

fore generally restricted to domains where items have rich textual descriptions, for example Web pages. The type of profile developed depends on the learning method employed. Burke [10] describes the use of Decision Trees, Neural Networks and Vector-Based representations of user profiles in content-based recommenders. Content-Based recommenders are emergent systems (as are CF systems), in that user-models are built up, extended and refined as the system is used.

Advantages of content-based methods include the following:

- Knowledge of domain is not required;
- Adaptive quality improves over time;
- Implicit user-feedback is sufficient.

Some of the disadvantages of these methods are:

- Recommendation quality is dependant on a rich historical dataset;
- They suffer from the "ramp-up" problem, ie new users have to rate a sufficient number of items before the system can reward them with quality recommendations;
- Content-based methods cannot provide serendipitous recommendations. ie: users are generally recommended items which are similar to those already seen. This lack of diversity rules out recommendation of items which may be of interest to a user. O'Sullivan et al. [56] describe this problem in more detail.

2.7.2 Utility-Based Filtering

Both utility-based and knowledge-based filters differ from content-based and CF methods in that they don't attempt to compile long-term profiles of users. Instead, recommendations are made based on an assessment of the correlation between a user's need and the set of available options. Utility-based techniques make recommendations based on the computation of a utility function created for each individual user. Linden et al. [43] describe a candidate critique agent, which develops and refines a utility function based on dialogue with the user. The basic operation of this critique agent is as follows:

- Suggest optimal and near-optimal solutions based on the system's current model of the user's preferences;
- Elicit and refine the user model;

- Possibly by using bad candidates, encourage the user to refine criteria;
- Indicate the range of available solutions in the dataset.

These agents provide additional information about a user's preferences, which ultimately leads to better recommendations. Utility-Based methods have the following advantages:

- There is no ramp-up problem: users do not have to wait for the system to build up a detailed profile before good recommendations can be made;
- This technique is sensitive to changes in user preference. (Does not rely on old models etc.);
- Can include non-product features.

Some of the disadvantages include:

- A user must enter a utility function, which can be time-consuming;
- The system's suggestive ability is static, ie the system does not learn.

2.7.3 Knowledge-Based Filtering

In knowledge-based filters, an attempt is made to draw inferences about user requirements and tastes. The difference between the inferences made by a knowledge-based filter and the general inferences of the other recommendation techniques is that this filter uses functional knowledge, i.e. they use knowledge about how a particular item can satisfy a specific user requirement. This knowledge affords the system the ability to reason about the relationship between a user need and a candidate item. User profiles can be built using any knowledge which supports this inference. Googles *PageRank* [57] uses information about links between Web-pages to define hubs and authorities. Other knowledge-based recommender systems include *Notebook Expert* [32] and *FindMe* [38].

The advantages of knowledge-based methods include:

- They do not suffer from the ramp-up problem;
- There is sensitivity to changes in user-preferences and tastes;
- Non-product features can be included;
- The technique can map from user needs to products.

Some of the disadvantages of knowledge-based filtering techniques are:

- There is a *lot* of knowledge-engineering required, which is a big drawback;
- Essentially the system does not learn;
- Because knowledge-based techniques require lots of information about the particular product domain they operate in, these systems have a poor performance record in large or dynamic product spaces.

2.7.4 Demographic Methods

Demographic recommendation techniques use a form of stereotyping to arrive at recommendations. Users are categorised into demographic groups based on personal attributes, and recommendations are made based on the tastes of a demographic group.

One of the most popular systems of this kind is the Lifestyle Finder [38], in which a technique of *elicitation* and *exploitation* is used to acquire user information. The Lifestyle Finder elicits information in a "friendly" manner, which does not require the user to answer any identifying information. Once a user has answered some subtle lifestyle-related questions, the system places the user into a pre-defined demographic cluster. Web pages liked by the cluster are then recommended to the user. The system uses 62 clusters based on a commercial consumer marketing system. If a user falls into more than one cluster, then all matching clusters form the candidate set for recommendation. A user can refine this set by answering more questions. The data used to form demographic clusters is taken from user's home pages. WINNOW [24] is used to learn a classifier for users based on home-page data. In the evaluation, 40% of users liked the recommended pages, compared with 30% for random recommendations.

Advantages of this method of recommendation include:

- The system's adaptive quality improves over time;
- No domain knowledge is needed;
- This technique can identify cross-genre niches.

Disadvantages of demographic filtering include:

- The system will suffer from the new item ramp-up problem;
- Users suffer from the "gray sheep" problem [13], where they have unique, esoteric tastes and therefore a severe lack of overlap with other users' profiles;
- The quality of recommendations depends on a large historical dataset;

- Gathering of demographic data can be a costly task.

2.8 Hybrid Information Filtering Techniques

Most recommender systems use collaborative or content-based Filtering to predict new items of interest for a user. Both of these methods fail to provide good recommendations in many situations [10]. For example, a content-based recommender will only generate recommendations of items that have similar content to ones seen before, thereby ignoring any diversity in our tastes. Hybrid recommenders have been forwarded as a solution to this, among other problems, [10]. For example, content-based filtering can make up for the initial poor performance of the collaborative technique.

A good example of this synergy in operation is in PTV [72] [16]. PTV contains an effective profiling system that reocords user-specific TV programme interests. The CF component of the filtering system initially gives poor ratings since the collaborative matrix is sparse and overlap is low. During this "teething" phase Content-Based filtering is used to give recommendations. Once-off programmes cannot be picked up by the CF system since there is not sufficient time for users to rate them. The content-based component may recommend these shows. Once the collaborative matrix starts to fill up, CF overrides the content-based approach. In a paper on PTVplus [72], a new approach to the integration of the above techniques is outlined. Data Mining techniques are used to extract association rules between TV programmes; these are then used to counter the overlap problem with traditional CF. Metadata about programmes is "extended" by new techniques (eg "likes a" implies "likes b") and then used to enhance the content-based part of the system. Another hybrid system, INVAID [35], is an integration of content-based and collaborative recommendation. Experimental evidence that a content-collaborative hybrid performs better in a recommender system is demonstrated in Melville and Mooney's Content-Boosted Collaborative Filtering [45], where the hybrid technique is evaluated in a movie recommender. Content-Based predictions are used to "pad out" the user-rating matrix in order that the CF technique can provide better recommendations. They tested this hybrid idea against pure Content-Based, pure CF and a naïve hybrid technique, using Receiver Operating Characteristic (ROC) and Mean Absolute Error (MAE) measurements. The naïve hybrid actually performed generally worse than the pure techniques, whereas the Content-Boosted hybrid outperformed every other technique. This research shows that the manner in which the Content-Collaborative hybrid is implemented is of paramount importance in a system. A "bad" hybrid is shown here to be worse

than a pure technique. The application stage of the AdRec system will support relevance feedback to optimise the balance between techniques and arrive at better recommendations.

2.8.1 Approaches to Hybridisation

There are a number of different approaches to hybrid recommender systems. Burke gives a good overview of these in [10], and Jameson et al. provide a very thorough review in [32]. The following are the main approaches to hybridisation that have been defined in the literature.

Weighted

In this approach, the scores (votes) from several recommendation techniques are combined together to produce a single recommendation. For example, Burke's *Entree* System [10] uses a weighted combination of five recommendation methods.

Switching

In the switching method, the system switches between recommendation techniques based on the current situation. For example, for items which have a rich textual description a content-based approach would be used, whereas conversely, a collaborative approach might be employed for non-machine-analysable items, [32].

Mixed

A mixed approach is when recommendations from several different recommenders are presented at the same time. An example of this approach is Amazon's [‡] Web pages.

Feature Combination

In this approach, features from various recommendation data sources are thrown together into a single recommendation algorithm. For example Case-Based Recommendation [56] uses both item features and responses of other users to generate a recommendation for a user.

[‡]http://www.amazon.com

Cascade

In a cascading hybrid system, one recommender refines the output given by another recommender. For example, Burke's *Entree* recommender [10] employs a content based approach, but uses a collaborative solution in a tie-break situation in the same way. cf Weighted Hybrids. Melville and Mooney [45] augment their CF algorithm with a content-based component.

Feature Augmentation

In feature augmentation hybrid systems, the output from one recommendation technique is used as input for another. An example of the use of this technique is shown by Good et al. in [25], where the CF sparsity problem is tackled by treating agents as users, and employing their item ratings in the recommendation process. One issue with this approach is that the agents are not providing real data, so the actual live user ratings get "diluted" by the agent ratings. It might be a feasible solution to employ a weighting mechanism to this technique whereby real user ratings get higher weights than agent-generated ones.

Meta-Level

In this approach, the model learned by one recommender system is used as the input to another. An example of this is the "collaboration by content" in [10], where terms in a content-based profile are assigned weights, and these term-weights are passed to a collaborative recommender and used in place of item ratings.

2.9 Similarity Algorithms

Each recommendation algorithm must implement some manner of similarity computation. This section gives a brief overview of three of the most common methods of similarity computation in recommender systems.

2.9.1 Cosine-Based Similarity

In cosine-based similarity, two potentially similar items are modelled as vectors in m-dimensional user-space. The similarity between two items is measured by computing the cosine of the angle between the two vectors. The similarity between items i and j, denoted by $sim(i, j)$ is given by

$$sim(i,j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|^2 * \|\vec{j}\|^2} \tag{2.5}$$

where "." denotes the dot-product of the two vectors.

2.9.2 Jaccard Similarity

The Jaccard similarity coefficient (also known as the Tanimoto coefficient) is used to measure the similarity between profiles containing *binary* elements. This technique examines the OR case as well as the AND case. For example, in two profiles A and B, this technique will look at the items which are present in both sets A and B, items which are present in A but are absent in B, and items which are absent in A but are present in B. In other words, it measures the ratio of *intersection* of profiles to the *union* of profiles.

2.9.3 Adjusted-Cosine Similarity

One fundamental difference between the similarity computation in user-based CF and item-based CF is that in the case of user-based CF the similarity is computed along the rows of the matrix but in the case of item-based CF, similarity is computed along the columns, i.e. each pair in the co-rated set corresponds to a different user (Figure 2.1). Computing similarity using the basic cosine measure in the item-based case has one important drawback: the difference in rating scales between different users are not taken into account. The adjusted cosine similarity offsets this drawback by subtracting the corresponding user average from each co-rated pair. Formally, the similarity between items i and j using this scheme is given by

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R_u})^2}}.$$

Here $\bar{R_u}$ is the average of the u^{th} user's ratings.

2.9.4 Correlation-Based Similarity

In correlation-based similarity, the similarity between two items i and j is measured by computing the Pearson correlation $corr(i,j)$. To achieve this we must first isolate the co-rated cases (i.e. cases where the users rated both i and j) as shown in Figure 2.1. Let the set of users who rated both i and j be denoted by U, then the correlation similarity is given by the equation in Section 2.5.3.

2.10 Machine Learning in Recommender Systems

Since the adaptation of a recommender system to a particular dataset is essentially a classification problem, Machine Learning is at the heart of an adaptive recommender. Machine Learning plays an increasingly important role in user adaptive systems. Accordingly, many different techniques have been successfully employed in various approaches. Schwab [70] shows that a basis for a recommender system (the ELFI system [70]) can be developed by using machine learning on positive training examples. Here it is stressed that choosing the right metrics for the classification task is essential for good system performance. In the AdRec system we are classifying the datasets rather than the users. This does limit the number of available metrics from which to choose for the classification task.

2.10.1 Machine Learning using Regression Techniques

Linear regression is the relation between selected values of x and observed values of y (from which the most probable value of y can be predicted for any value of x). In the AdRec system our observed values of Y will be our algorithm performance metrics on test data, while our selected values of X are the classification metric values for our four datasets. In this manner we employ simple linear regression modelling to build a function that can predict the performance of a recommender algorithm with respect to a classified dataset.

Zhang et al. [82] use linear classifiers in a model-based recommender system. They compare their method with another model-based method using decision trees and with memory-based methods using data from various domains. They show that the use of linear classifiers outperforms a commonly proposed memory-based method in accuracy tests, and also show that the system has a better tradeoff between off-line and on-line computational requirements. Herbrich et al. [29] use ordinal regression to aid machine learning in an information filtering task of learning the order of documents with respect to an initial query. They also show that this approach outperforms more naïve approaches.

2.11 Tapestry

After John Hey's "likeminds" (the original company behind the MovieCritic recommender system[§]) the first use of the term "collaborative filtering" was in the Tapestry system presented by Goldberg et al. in [22]. This was an email filtering

[§]http://www.moviecritic.com/

system in which a CF process was defined as a process in which "people collaborate to help one another perform filtering by recording their reactions to documents they read." Users of the system can annotate documents with keywords and phrases in order to share information on their preferences. Recommendations are made when the user issues a complex query on these words and phrases, and the system chooses items based on the annotations and on the identity of the annotator. One general query of this type is "show me documents that user z annotated as interesting" The system also allows for implicit user-feedback on queries. For example knowing that user *a* only sends email responses to documents he finds interesting, selection could be made on this basis.

On closer analysis, Tapestry is just an extended querying system, where users can benefit from the annotations of other users. There are a number of limitations to this system, however. The main drawback to the system is that users must have some knowledge of or have a prior relationship with the advisor, or else his recommendation could not be trusted. Another problem with the system is that users are free to annotate items with any keywords they choose, thereby making the probability of two users using the same set of keywords very unlikely. Tapestry [22] was a fairly cumbersome CF system, but it laid the groundwork for the more advanced systems we have today. The other example systems discussed in this section do not rely on a prior relationship between users in order to recommend items.

2.11.1 Ringo

Shardanand and Mae's online music recommender RINGO [71] is a statistical CF system. Users rate albums which they have listened to, and the system gives them a list of albums it thinks they will like. One interesting feature of this system is that users can append new items (albums) to the system's itemset. This resulted in a >500% increase in the itemset size in the first six weeks of operation.

Predictions are computed in this system by a computation of weighted averages of user-ratings. Weight intensity depends on similarity between the active user and the user whose ratings are being weighted. This is computed using one of three common similarity metrics: Mean Squared Difference (MSD), which is simply the average deviation in either direction; Pearson correlation, described in detail in section 2.5.3; and constrained Pearson correlation. The latter metric attempts to use both positive and negative correlations between users. Ringo has a seven-point rating scale varying from "much liked" to "average" to "awful". Ratings above 4 indicate liked items, while ratings below 4 indicate disliked items. Due to

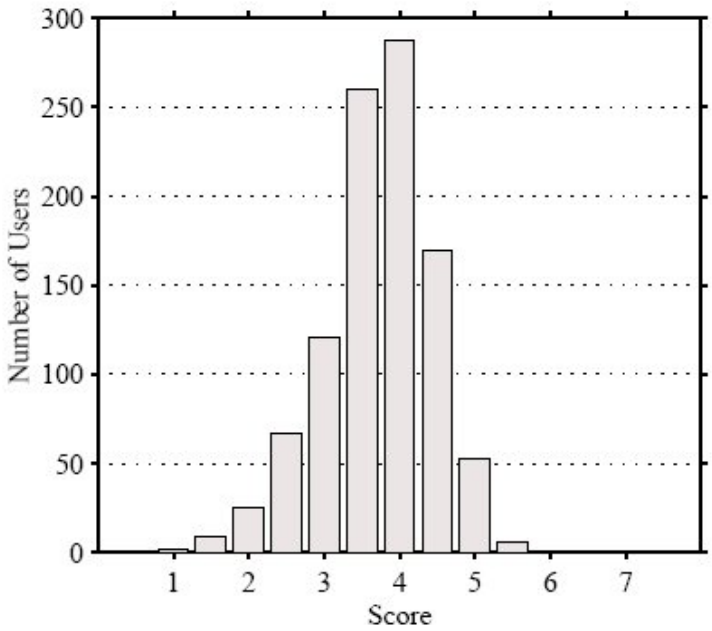


Figure 2.3: Ratings Distributions in RINGO.

the apparent Gaussian distribution of ratings [14], illustrated in Figure 2.3, this metric will only increase a similarity coefficient between two users if they have both rated an item either positively or negatively. Ringo has another prediction algorithm known as *artist-artist* correlation, which uses the correlations between items, calculating a weighted average of the active user's scores on similar items. This technique is not too dissimilar to the Item-Based CF algorithm described in Section 2.5.4, and by Fisk in [19], and Sarwar et al. in [67]. The results shown in [71] show poorest performance from the artist-artist technique, which is in stark contrast to the results in [19] and [67].

2.11.2 Grouplens

The GroupLens Research Project has been carrying out research into CF and recommender systems since 1992. The GroupLens system is one of the first automated collaborative filtering (ACF) systems. The work on this system has been extensively reported; see, for example, [65], [68], [25], [69] and [67].

The GroupLens ACF system was first introduced as a solution to the information overload problem in Usenet News, which was losing popularity due to the mass of irrelevant information it contained for any one user.

This system identifies peer groups based on the Pearson correlation (see Section

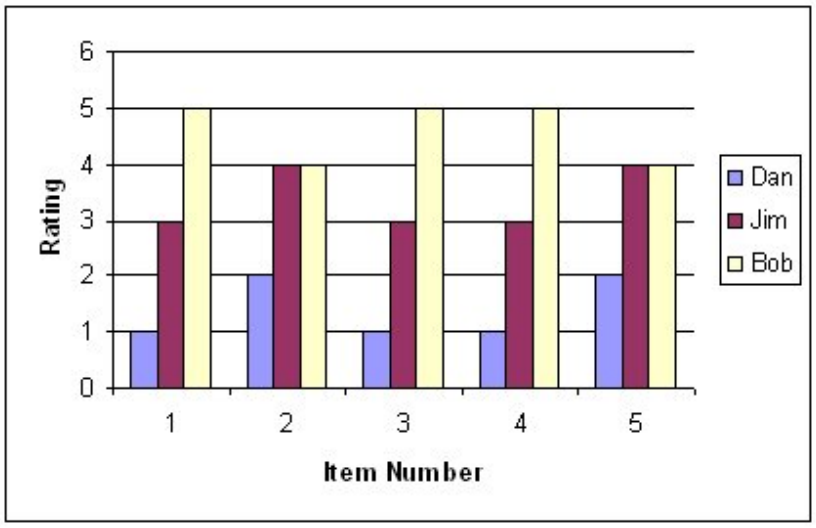


Figure 2.4: Rating Histories Example

	Dan	Jim	Bob
Dan	1	1	-1
Jim	-1	-1	-1
Bob	-1	-1	1

Table 2.3: Correlations generated from Ratings History in Figure 2.4

2.5.3 and [9]). Two users may not rate items identically, but if they have a high number of co-liked and co-disliked items then they will have a positive correlation value. Predictions are then generated using the deviation from the active user's mean rating.

Figure 2.4 depicts an example ratings history for three users and Table 2.3 shows the resulting correlations (Example taken from [12]).

2.11.3 INVAID

INVAID (An Intelligent NaVigational AID for the World Wide Web) [35], [36] and [37] is presented as a solution to the information overload problem. As the name suggests, it is an intelligent navigational aid for the Web. It uses adaptive hypermedia techniques to ease the task of finding information on the Web and to personalise the Web experience for users. It uses filtering techniques to find interesting pages for a user based on information obtained via user modelling techniques. The information filtering sub-system in INVAID consists of a collaborative

filtering component and a content-based filtering component. The collaborative filtering component takes user profiles containing URLs, then uses this information to form peer groups of users and subsequently chooses URLs to suggest to users. In tandem with this, URLs which match a user's profile are suggested by the content-based filtering component.

For information filtering to occur, it is necessary to acquire information about the user and store this in user profiles. User modelling techniques make this possible. A user can be tracked as they navigate the Web, that is to say, some form of computer agent can follow a user on their journey through the Web. Such an agent can establish which information a user is interested in. Information gathered by an agent about users' interests can then be used to examine the pages on the Web and select the pages of greatest interest to a user. Such Web personalisation alleviates the burden imposed on the user. It allows for "interesting" pages to be presented to a user while imposing a minimal effort on the user and it can speed up Web searches. Also, it allows users to view interesting pages that they might otherwise have missed entirely.

The INVAID system uses a twofold personalisation process. It tracks user interactions with the Web, i.e. user requests to the Web and the responses received are both intercepted. This permits two means of finding interesting information for a user. Information obtained from a user's Web requests is combined with information from all other users' requests. Then, where appropriate, URLs viewed by other users can be suggested to a particular user. The other means of finding information for a user of this system is to look at the information sent from the Web to that user. The system can then search the Web for similar information to that in which the user has shown an interest. INVAID makes a representation of users' interests, which are stored in user profiles. For each user two separate user profiles are maintained. One user profile contains a list of URLs visited by the user, along with associated information for these URLs. This associated information consists of explicit relevance feedback in the form of a user grading of a Web page and various forms of implicit relevance feedback. All information associated with a URL is combined to obtain an overall measure of interest for a user in the URL. The other user profile contains a list of keywords, with weights of importance, obtained from the pages visited by the user.

This system is similar in functionality to the Letizia system [41], although the approach to aiding Web browsing is tackled differently.

2.12 Summary

This chapter has dealt with recent research in the areas relevant to this work. We have introduced several example recommender systems, and discussed the techniques and technologies that drive these and the AdRec system.

Specifically, we have described three algorithms: user-based, item-based and association rule-based collaborative filtering. We have examined how these algorithms operate, from modelling user profiles to the prediction task, and we have looked at common problems and drawbacks associated with these algorithms. This chapter also examined the different approaches to hybridisation of recommendation algorithms, and the costs and benefits of each.

The next chapter describes the design and development of our system, and describes the data on which it operates, and the classification of this data by the system.

Chapter 3

The AdRec System

This chapter introduces the AdRec system. AdRec (short for Adaptive Recommender) is a recommendation system which has the ability to dynamically adapt the recommendation algorithm it uses to suit the particular dataset it is being used on.

3.1 Introduction

We have shown in chapter 2 that there are serious shortcomings with current collaborative recommendation algorithms. Issues such as the sparsity and latency problems mean that recommenders cannot be trusted to give completely reliable results, or even in some cases to provide any results at all. This is the main motivation behind AdRec. We postulate that recommendation efficiency and accuracy can be increased by applying the most suited algorithm to the data. For example, in a situation where data is very sparse and there is not much overlap between profiles for a standard user-based CF algorithm to operate, employing an association rule-based approach should yield better results than the user-based approach since the former generally performs better on sparse data.

3.2 Recommendation: The AdRec Solution

The AdRec system is designed to enable users to deploy the most suitable of the available recommendation algorithms to their particular system for the small cost of classifying the data on which the recommendations are to be made.

At the outset, a hybrid approach to the system was proposed. This method, detailed in the next section, proved to be computationally expensive, requiring more than one recommendation algorithm to run online, thereby increasing the

systems time complexity to $O(2n^2)$. An adaptive approach was adopted in its place. This is described in Section 3.2.2.

3.2.1 Hybrid approach to the Recommendation task

There are many existing methods to combine recommender techniques into hybrid schemes. Jameson et al. [32] provide a good overview of these methods, most of which have been presented in Section 2.8.1.

The original hybrid technique was designed as follows: Each of the four recommendation algorithms was trained on the the 80% section of the data. (the training set). The remaining 20% of the data was split into two sections. The first for testing the performance of each individual recommendation algorithm, and the second section was used for testing the performance of the adaptive system as a whole. This split is illustrated in Figure 3.1.

The performance of each algorithm, and the type of test data used, is stored in a database (as shown in Figure 3.2). Thus far the hybrid approach to the filtering engine design is not dissimilar to the adaptive approach. The hybrid method then requires pre-coded crosses of each of the algorithms in the system. It was intended that, by referencing our performance database, we could employ a hybrid of the two best performing recommendation strategies based on a bare-bones classification of a new dataset. It was hoped to develop a synergy between the 1st and 2nd best performing algorithms. This original idea was abandoned in favour of an adaptive approach for two main reasons. Firstly, hybridisation was deemed to be computationally too expensive, secondly and more importantly, there were not enough collaborative filtering datasets from different domains available to make the technique viable.

Training and Testing in the Hybrid Approach

Testing of the hybrid approach was designed as follows. For a dataset previously unseen by the system, the testing and training of each individual algorithm is carried out on 80% of the overall data. The remaining 20% of the overall data is used to assess the performance of the hybrid technique against the original filtering techniques. A positive result involves the hybrid technique generating better results, using nearest neighbour (k-nn) and mean absolute error (MAE) testing, as described in Chapter 5.

Figure 3.1: Splitting Data in the Hybrid System.

3.2.2 Adaptive approach to the Recommendation task

To achieve this adaptability in our system, we make the assumption that our datasets can be adequately classified (for CF purposes) by a set of their salient features. These include user-item ratio, sparsity and data type. We tested our three collaborative recommendation algorithms (User-Based, Item-Based, and Rule-Based CF) on four different experimental datasets (EachMovie, PTV, Jester and MovieLens), and noted the relative performance differences of each method with respect to the classification metrics above. From this information it was possible to develop a regression function for algorithm prediction based on the classification metrics alone. We tested the performance of this function by introducing another dataset, SmartRadio, a music ratings set [26], [28]. This set is classified according to the required metrics, and the resulting values are run through the regression function to attain an algorithm prediction. If we can successfully perform this algorithm prediction task, we can form the basis of a generic recommender system, which can employ cutting-edge filtering techniques to a given system without having to manually tailor the recommendation engine for that system. The system design is completely modular, which has the advantage of allowing new techniques to be added as they develop.

Figure 3.2.2 is an overview of the AdRec system. Experimental data from any one of the datasets listed above is parsed into the correct format by a parser, discussed in Section 4.3.1. The data is then classified according to its salient features by a classification component (see Section 3.4). Once the data is properly classified, each algorithm is allowed make predictions on 80% of the data. An analysis module statistically assesses the performance of each algorithm (using mean absolute error as its main metric), and stores the performance grading of that algorithm and the corresponding dataset classification values in an SOL database. This is discussed in more detail in 4.3.2.

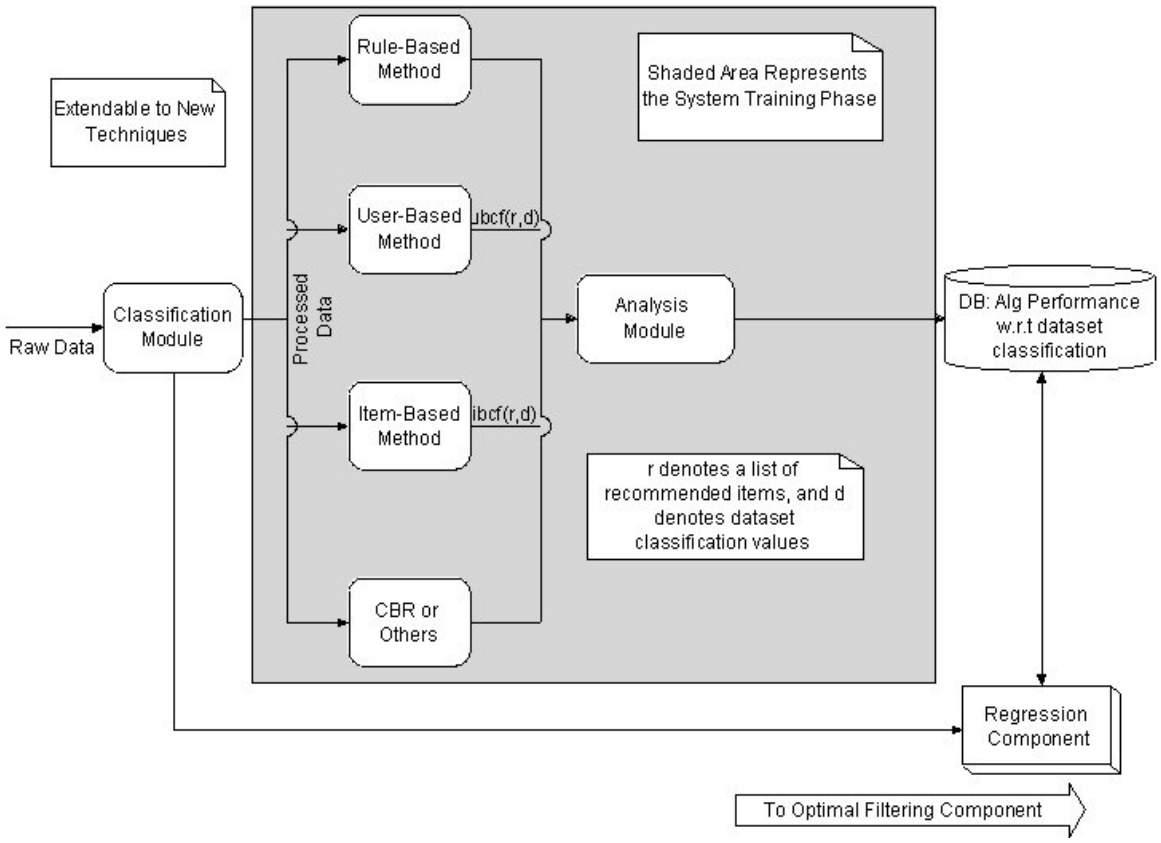


Figure 3.2: System Architecture.

3.3 Experimental Data

Data is at the core of the AdRec system. In order to train our system properly, we need a broad range of datasets, preferably with diverse values for the user-item ratio, density, etc. Currently there are a limited amount of experimental ratings datasets available for collaborative filtering research, and we feel that the results of this research would be greatly improved if the system could be trained on more than the four datasets we had available.

For training and testing our algorithms, the four experimental datasets used were: Jester [23] (an experimental dataset of jokes ratings, consisting of 21,800 users ratings of 100 jokes), EachMovie (73,000 users ratings of 1628 movies), PTV [16] (622 user ratings on TV programmes), and MovieLens [63] (100,000 user ratings in the movie domain). It is hoped to also include a customer-product purchase database from an online sales company in the near future. For the purposes of our testing we selected subsets of 900 profiles from each of the above datasets comprised of the largest profiles: those users who had rated 20 items or more (with the exception of PTV which only contains 622 profiles). For modularity

in our system, these datasets were all parsed into the same format and stored in an SQL database.

3.3.1 Movielens

MovieLens [68], [67], [30] and [63] is a database of movie ratings, collected from a movie recommendation website*. This dataset was collected by the GroupLens Research Project, a part of the Department of Computer Science and Engineering at the University of Minnesota. GroupLens was founded in 1992 to study collaborative filtering, information filtering and recommender systems. MovieLens was created in 1997 to prove just how well collaborative filtering works.

There are two MovieLens datasets available. The first consists of 100,000 ratings of 1682 movies by 943 users. (and is the set used in our testing) The second one consists of approximately 1 million ratings of 3900 movies by 6040 users, and would be a candidate dataset for any future work on this system.

3.3.2 EachMovie

Compaq Research (formerly Digital Research) ran the EachMovie recommender for 18 months to experiment with a collaborative filtering algorithm. When EachMovie† was shutdown, the dataset was released to the public for use in research. MovieLens was originally based on this dataset. It contains 2,811,983 ratings entered by 72,916 users for 1,628 different movies, and it has been used in numerous CF research projects.

3.3.3 Jester

Ken Goldberg from UC Berkeley has also released a dataset from the Jester Joke Recommender System. ‡ (see [23] and [45] for example). This dataset contains 4.1 million continuous ratings (-10.00 to +10.00) of 100 jokes by 73,496 users. Using the formula in Section 3.4 we get a sparsity of 53.59% for this dataset.

3.3.4 PTV

The PTV collaborative recommendation data for television listings was collected at University College Dublin, and then at ChangingWorlds§. PTV has been used

*http://movielens.umn.edu

†http://research.compaq.com/SRC/eachmovie/

‡http://shadow.ieor.berkeley.edu/humor/

§http://www.changingworlds.com

widely in collaborative filtering research: see [60], [16], [49], [53], [51], [56] and [75], for example. The version used in our system is a database of 622 users' ratings of TV programmes. Details on sparsity and user-item ratio are shown comparatively in Table 3.1. Ratings in this dataset are binary. A user's profile consists of a list of viewed TV programmes. If a user has viewed a TV programme, it is assumed that that is a "liked" programme.

3.3.5 SmartRadio

SmartRadio [28], [26] is a Web-based client-server application which uses streaming audio technology and collaborative recommendation techniques to allow users build and share music programmes. The SmartRadio[¶] dataset was collected at Trinity College Dublin over an audio-sharing application on their intranet. This dataset has approximately 10,000 items and 395 user profiles. A scale of 1 to 5 (i.e, [1,2,3,4,5]) is used in SmartRadio when users are rating tracks. A rating of 1 means the user strongly dislikes the track and a rating of 5 indicates the user strongly likes a track. Details are shown in Table 3.1.

3.4 Dataset Classification

We classify our datasets according to three metrics for the purposes of proposing the suitability of particular techniques to other datasets which fall into the same classification. The metrics employed are dataset size, dataset density and the ratio of users to items. Dataset size and domains are explained in the previous section, leaving sparsity and user-item ratio. Firstly, we define the sparsity of our data using the following formula (from [72]):

$$1 - \frac{number\,of\,nonzero\,entries}{number\,of\,total\,entries}$$

Table 3.1 details the sparsity and user-item ratios for our four datasets. There is a correlation between the performance results of our algorithms in Table 5.9 and the dataset classification values in Table 3.1. For example, the best result that we get is a predictive accuracy of about 76% from the user-based algorithm. This is on the EachMovie dataset, which has a sparsity rate of 97.6% which is more sparse than some of the other sets. However, this dataset is the largest and therefore gives the most amount of data on which to train the filtering algorithms.

Looking at the worst performer from Table 5.9, we can see that the SmartRadio dataset gets an average predictive accuracy of 56%, which is a very poor result.

[¶]http://smartradio.cs.tcd.ie/

Dataset	*User:Item Ratio*	*Sparsity (%)*	*Type*	*total possible ratings*
PTV	1:6	94.25	TV Programmes	2.3 Million
MovieLens	9:13	93.6	Movie Ratings	1.6 Million
Jester	9:1	46.41	Jokes Ratings	7.4 Million
EachMovie	9:17	97.6	Movie Ratings	118 Million
SmartRadio	1:9 (approx)	99.98	Music Ratings	4 Million

Table 3.1: Classification of Experimental Datasets.

This is correlated with the very high sparsity value of 99.98% combined with the relatively small size of the dataset.

We attempt to capture the relationships between the values each dataset has for the classification metrics and the performance of each prediction algorithm on that dataset. This is done using linear regression techniques discussed in Section 3.5.

3.4.1 The User-Item Metric

The User-Item ratio is a valuable classification metric since our similarity algorithms for this research are based on these two features. Is has been shown in [65] that IBCF performs better with a greater number of users in an item-profile (ie the total set of ratings provided by users for a particular item) from the training set, whereas the total number of items is not as relevant. This is quite intuitive since there is obviously more information on which to base our similarity computations. The same holds true for the number of items in a user's profile in the UBCF algorithm. Another area of interest is the distribution on ratings within a dataset. For future work on dataset classification, Chauvnet's criterion [19] could be employed to attempt to remove possible misleading ratings. This is a statistical technique used to eliminate "bad" data using the supposition that it is statistically unlikely to turn up in the true Gaussian population.

Looking at Table 3.1 we can see that there is a diverse range for our user-item metric, MovieLens having 13 items for every 9 users, while the Jester dataset has 9 users for every 1 item. A regression analysis of the effects of this on the overall performance results (Table 5.9) is given in Section 3.5.

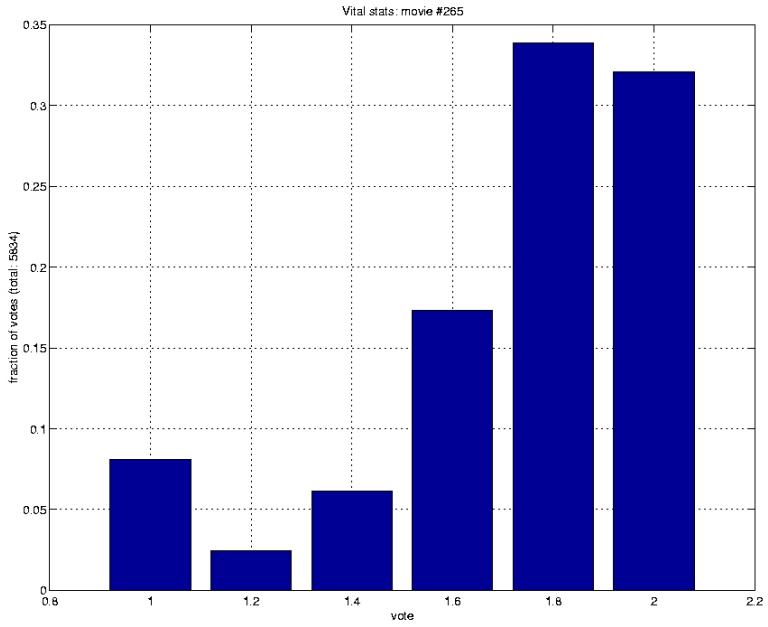


Figure 3.3: EachMovie: Sample Ratings Distribution for 'Water for Chocolate'

3.4.2 Ratings Distribution in Datasets

It has been shown in [72] that clusters of high ratings density occur in collaborative filtering datasets. The graphs in Figures 3.3 ('Water for Chocolate') and 3.4 ('Natural Born Killers') show ratings distributions for each of two items from the EachMovie dataset.

Dellorcas [18] gives a detailed account of the adverse effects of erratic distribution of ratings throughout CF datasets. We can see from the graphs in Figures 3.3 and 3.4, which plot the mean distribution of votes for the items against the percentage of total votes, that the ratings distribution changes over the items in the system. Is is shown empirically in [18] that this phenomenon can skew the results of CF analysis.

3.4.3 Effects of Rating Scale Differentiation on Prediction

Cosley et al. show in [14] that the mapping between users' opinions and rating scales is complex. As discussed in Section 3.3, the datasets used in the AdRec system all have different rating scales, and this may have an effect on the system's performance. Hill et al. [31] asked users to re-rate movies they had rated six weeks earlier. The 19 users who responded had a strong correlation (0.83) between their earlier and later ratings. This shows that there is a consistency in user ratings, but

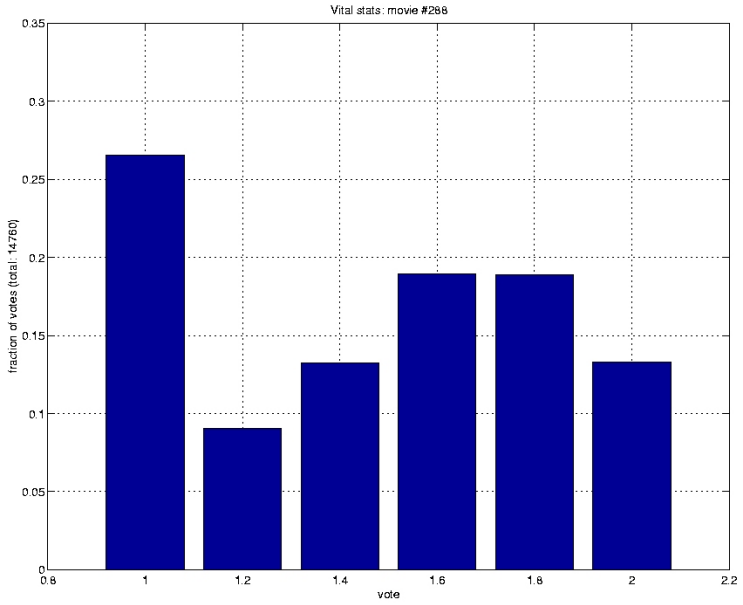


Figure 3.4: EachMovie: Sample Ratings Distribution for 'Natural Born Killers'

Dataset	*Rating Scale*	*'Good' Threshold*
PTV	1 or 0	1 only
MovieLens	1 to 5	>= 3
Jester	-10 to +10	>= 0
EachMovie	0 to 5	>= 2.5
SmartRadio	1 to 5	>= 3

Table 3.2: Rating Scales and Thresholds.

in [14] it is shown (using three separate rating scales on the MovieLens dataset) that different rating scales do affect user ratings. For each of our datasets we aimed to predict whether hidden items would be 'liked' or 'disliked'. To achieve this we had to define 'liked' and 'disliked' thresholds for each of the datasets we used. Table 3.2 illustrates the range of the rating scales for each of the datasets we used and shows the threshold for a 'good' recommendation for each of the rating scales. Our motivations for choosing the midpoint of each scale was straightforward: The PTV dataset, which contains binary ratings could only be compared in our tests with the other sets if we chose the 50% threshold value.

3.5 Linear Regression Modelling

A linear regression model [29] is built up using our evaluations in [49]. This is based on a predictive function of several variables, as described in [82]. The calculation of this function for the AdRec system is discussed in Section 3.5.1.

In a simple linear regression calculation we assume that the dependent (response) variable (the performance value of our filtering algorithms) is related to one independent (explanatory) variable through a linear function of the form

$$\hat{y} = b_0 + b_1 x$$

where $\hat{y}$ represents an average value based on the regression function evaluated for the observed value.

The simplest plot of a regression function is a best-guess straight line through some data points on an X-Y axis. For our calculations, which involve multiple regression (analysis of several variables), it was necessary to introduce a "response surface" graph for each individual algorithm in order to make predictions. This is a curved plane in 3-dimensional space. Wherever our new X_i values (representing the sparsity and user-item ratio of a new dataset) intersect this plane gives a predicted algorithm performance value.. We show the response surface for the Item, User and Rule-Based algorithms in Figures 3.5, 3.6 and 3.7 respectively.

In this section, we discuss two extensions to the basic linear regression model, the multiple linear regression model and the polynomial regression model, both of which are used for prediction in the AdRec system.

If we assume that the dependent variable Y is related to multiple independent variables $X_1, \ldots, X_m$, a common regression function is the multiple linear regression function

$$\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_m x_m,$$

where $\hat{y}$ represents an average value based on the regression function evaluated for the observed values $x_1, \ldots, x_m$, $b_0, b_1, \ldots, b_m$ are estimates of some unknown parameters $\beta_0, \beta_1, \ldots, \beta_m$ (in this case, each of the values for our dataset classification metrics) that are calculated from the data. If we assume that the dependent variable Y is related to one independent variable X through a polynomial function

$$\hat{y} = b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m,$$

we speak of a polynomial regression function. As before, $\hat{y}$ represents an average value based on the regression function evaluated for the observed value x and its powers and $b_0, b_1, \ldots, b_m$ are estimates of some unknown parameters that are calculated from the data. In our experiments, we considered a polynomial curve

as our regression function, as well as a linear one to assess different fits. The polynomial curve is given by the following equation.

$$\hat{y} = b_0 + b_1 x_1 + b_1 x_2 + b_2 {x_1}^2 + b_2 {x_2}^2,$$

where b_0, b_1 and b_2 are estimates of unknown parameters, x_1 and x_2 are values for user-item ratio and sparsity of each dataset, respectively. It must be noted that we reverted to using a linear regression function since the polynomial curve was overfitting the training data.

In addition to the research on simple linear regression, much work has been carried out on multiple linear regression techniques. Two references that are particularly useful for multiple linear regression are Neter et al. [47] and Herbrich [29].

In any prediction task using regression, it is important to define the most suitable function to map our observed results onto our data. In our analysis, it was first decided to try and look at each independent variable (X_i value) individually with respect to algorithm performance. It was then noted that this exercise would not yield any informative results since our X_i values are User-Item ratio and sparsity of each dataset. Since both of these values stem directly from the same data, it was a more logical choice to examine both X_i values in tandem.

Appendix 1 shows the quantative analysis of the multiple linear regression functions. One problem with this analysis was the availability of collaborative filtering datasets. It is difficult to ascertain a high quality prediction function with so few data points. The use of artificially generated CF datasets was considered, and would help the regression analysis greatly, but this would also dilute the real world data to such an extent that it was deemed inappropriate for our experiments.

In contrast to collaborative filtering systems, Ansari et al. [4] propose a hierarchical Bayes methodology which makes use of additional demographic data and external expert ratings for recommendation. They find that for their specific (small) data set, simple linear regression performs almost as well as their hierarchical Bayesian methodology. However, they argue that while linear regression forecasts have low error values for predictions close to the mean rating, they have high error on ratings with a high deviation from the mean rating.

3.5.1 Calculation of Linear Regression Functions

$$E\{Y\} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \tag{3.1}$$

Values for each $\beta_i s$ are obtained by going over all the classification metric values for each dataset, and the best performing algorithm for that set, and then solving the

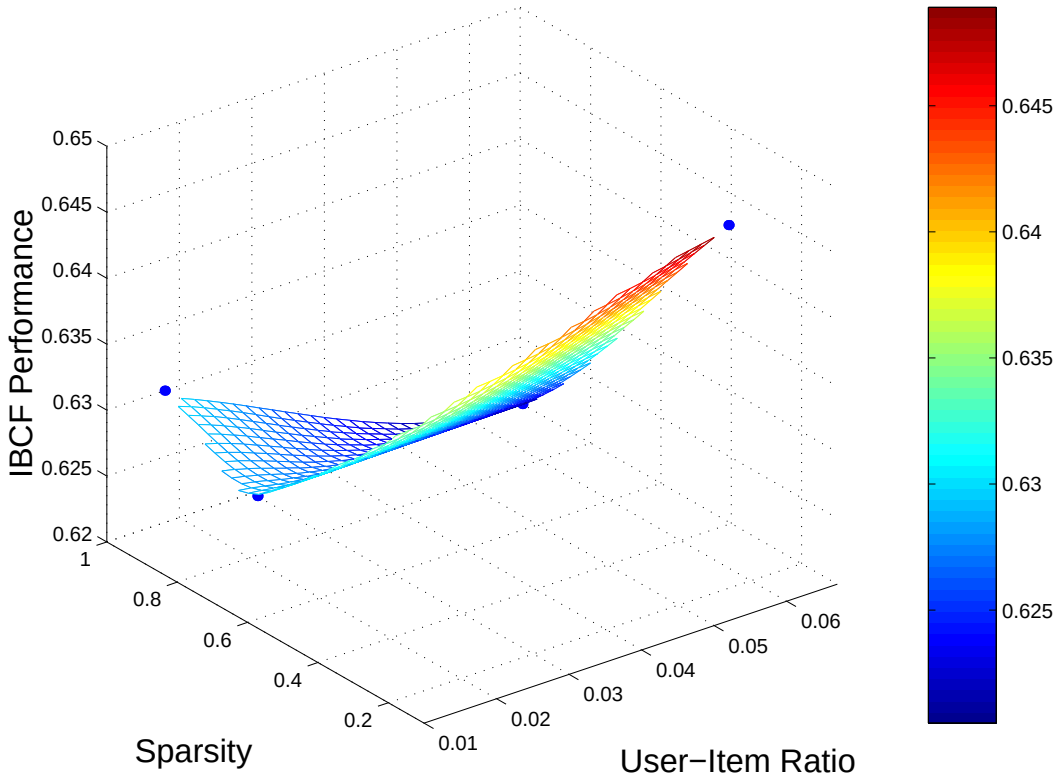


Figure 3.5: Response Surface Graph for the IBCF algorithm.

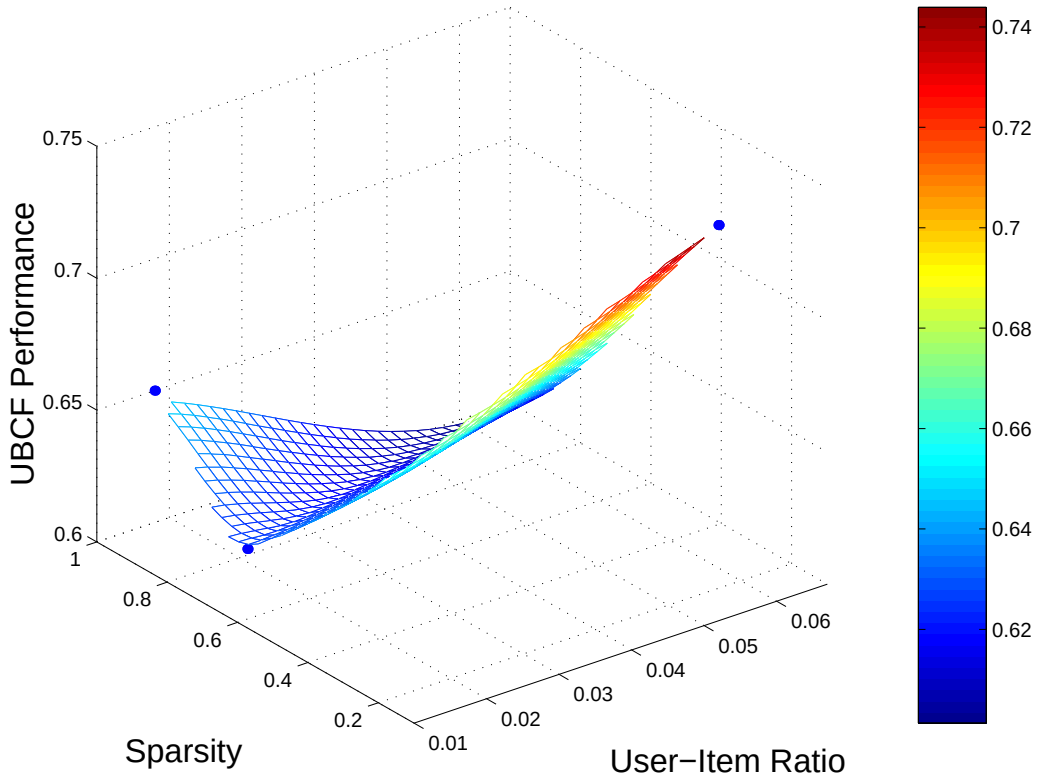


Figure 3.6: Response Surface Graph for the UBCF algorithm.

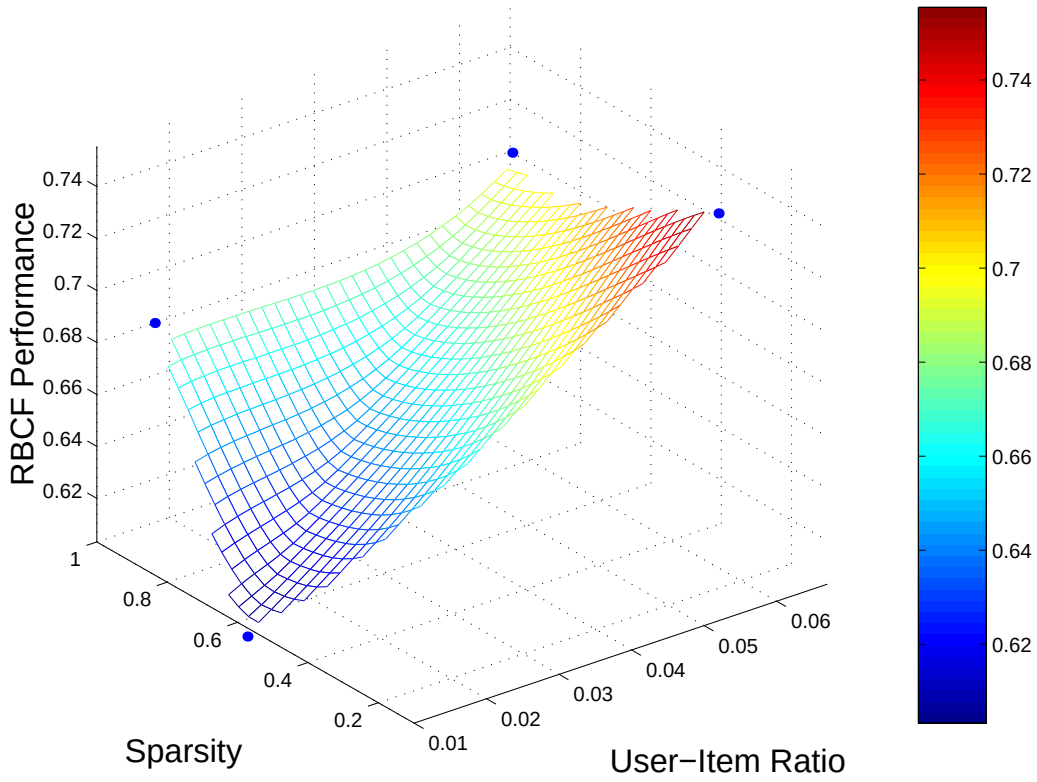


Figure 3.7: Response Surface Graph for the RBCF algorithm.

Dataset	UBCF	IBCF	RBCF	Zero-r	Best Performing Algorithm
PTV	63	65	68	42	Rule-based
MovieLens	62	60	70	56	Rule-Based
Jester	63	63	60	44	User-Based
EachMovie	65	75	76	50	Rule-Based

Table 3.3: Performance Results for each algorithm

resulting system of simultaneous equations. The SmartRadio dataset was classified according to the same metrics as the others and was found to be over 99% sparse ($\beta_1 = .99$ after normalisation) and have a user-item ratio of 1:9. ($\beta_2 = 0.111$). The calculation of the regression function for algorithm prediction is shown in full in Appendix A. Substituting in the performance values for each algorithm, shown in Table 3.2, and the classification values from all of our data, as seen in Table 3.3, we computed a unique performance function for each dataset.

Full details of all of the calculations for these functions is given in Appendix A. The regression function for the Item-Based algorithm is the following:

$$E\{IBCF\} = 10.38\beta_1 + 2.75\beta_2 + 2.55 \tag{3.2}$$

So our regression analysis over user-item ratio, sparsity and algorithm performance in 4 other datasets generates predicted scores for each algorithm on the SmartRadio dataset as follows:

1. RBCF = 66.44%
2. UBCF = 65.2%
3. IBCF = 64.016%
4. Zero-r = 58.20%

If we examine the actual predictive accuracy results for our algorithms on the SmartRadio dataset, we can see that although the regression function has overestimated the performance, it does predict *every* algorithm in the correct order. A possible explination for this overestimate is that the importance of the sparsity metric may not have been given enough weight in the regression function. The SmartRadio dataset is *extremely* sparse at 99.98% sparsity, much more sparse than any of the other datasets. This is our main theory to explain the overestimates in our predictions.

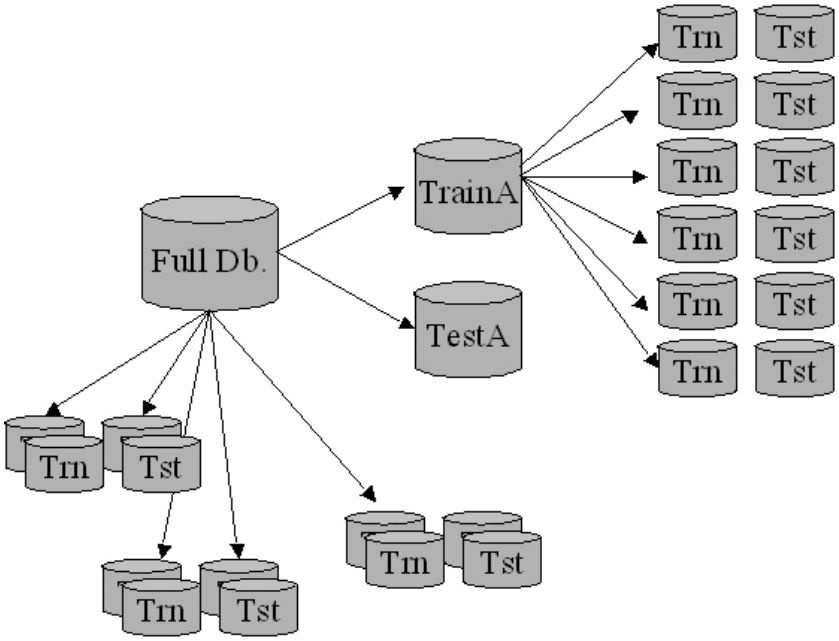


Figure 3.8: Splitting User Profiles.

The actual predictive accuracy scores on the SmartRadio dataset are:

1. RBCF = 57%
2. UBCF = 56%
3. IBCF = 55%
4. Zero-r = 53%

3.6 Training and Test Data

In order to train each of the recommendation algorithms we need to split our data into relevant training and test sets. The full database needed to be split into several sections for training and test data. There were two main splits required. The first was to separate some test data to assess the performance of the entire system. (ie the adaptive recommender). This test section was approximately 10% of the overall data for each of our datasets, and is denoted by the area *TestA* in Figure 3.8. This portion of the test data is used to assess the performance of the system, as shown by the non-shaded area of Figure 3.2.

The second split of our data was to assess the individual performance of each of our recommendation algorithms individually. This is basically the shaded area

in Figure 3.2, and the data is labelled as *TrainA* in Figure 3.8. The multiple train and test data partitions are shown to indicate the cross validation of our data that we performed. From our *TrainA* data, we randomly selected five different training and test sets, assessed the performance of our recommendation algorithms using standard 'leave one out' analysis for each selected partition, and then we averaged the results over all five partitions to get our final predictive accuracy values for each algorithm.

3.7 Summary

In this chapter we have introduced the AdRec system at a general level, and given a description of its overall architecture. We have discussed how the system evolved from a hybrid approach to the recommendation task to an adaptive one, and outlined the reasons for this. We have discussed our experimental data in detail, and shown how this data is classified by the system according to its salient features for collaborative filtering. We then show the calculation of predictive linear regression functions for each of our recommendation algorithms from our dataset classification values, and we explain and show how the system correctly predicts the relative performance of each recommendation algorithm from the regression function, even though there is an overall underestimate in the actual predictions. We provided some possible explanations for these underestimates. Finally, we explained in detail how data was allocated for training and testing in the AdRec system.

The following chapter will introduce the AdRec system from a more specific design and implementation perspective, detailing the technologies in the system along with the three tier architecture of the system.

Chapter 4

AdRec: System Design and Implementation

4.1 Introduction

The overarching goals for this system are to develop and run the algorithms shown in the previous section, test them on the four experimental datasets we have described, develop a regression function based on our implementation, and implement a testing procedure that can prove empirically that our regression function can accurately predict the best preforming algorithm for a fifth dataset, based on an assessment of its user-item ratio and its sparsity metrics.

This chapter deals with the design and implementation of the system as a whole. We also discuss the modular nature of the system, then explain the data issues such as choice of persistent storage. We examine system design from a UML perspective, with some detail on each algorithm. Then we discuss the three-tier architecture of the system, implemented using an SQL back end, Java for middleware and JSPs, Java Servlets and Java SWING for the interfaces to the system. Finally we will present some explanatory screen-shots of the SWING graphical user interface and of the online demonstration version of the system.

4.1.1 Modularity: The Key Factor

For the sake of simplicity and clarity, the system was broken down into modules. Figure 4.1 gives a breakdown of the different modules in the system. A big advantage of this modular design is that new recommendation algorithms can easily be added to the system as they are developed.

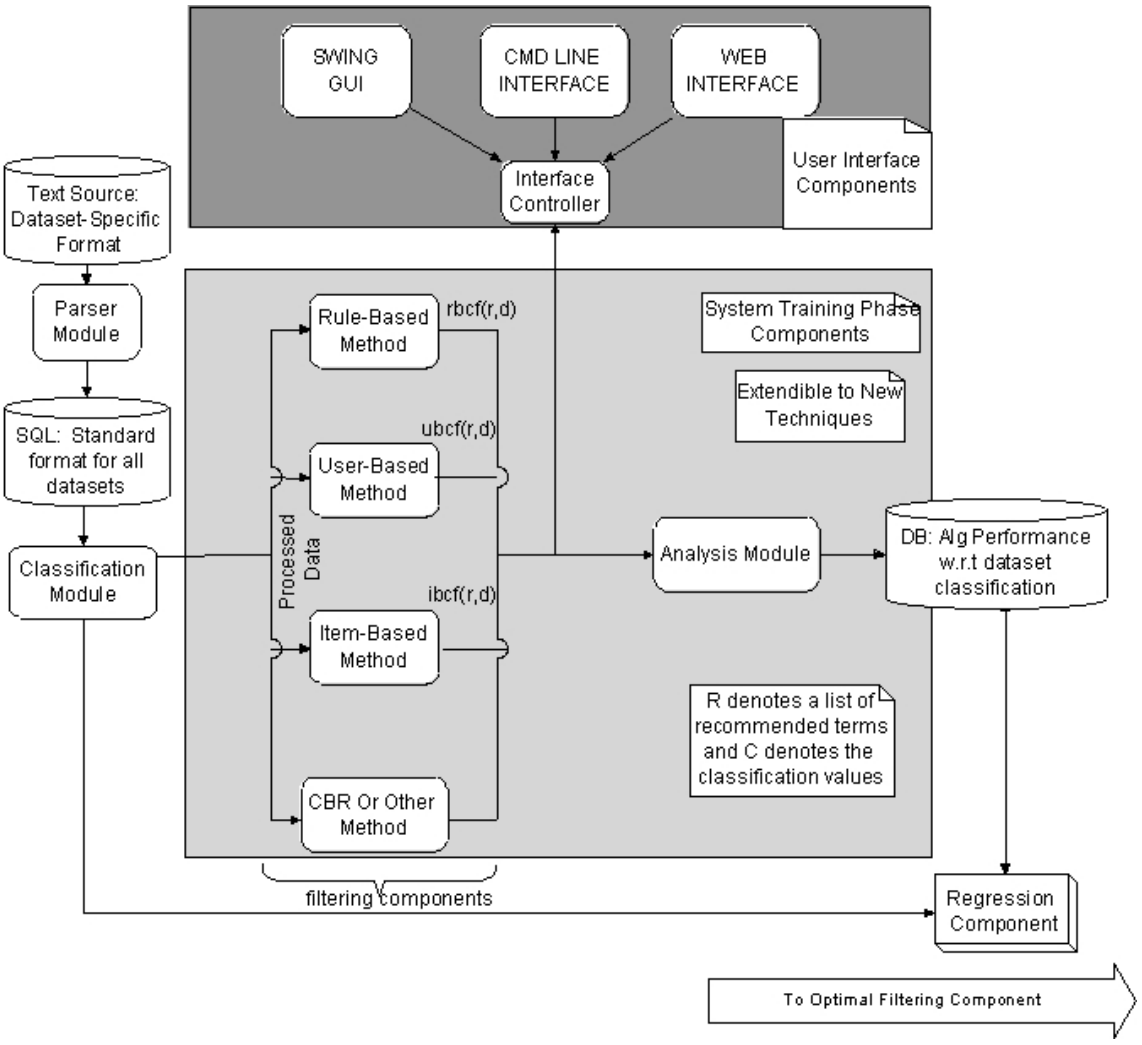


Figure 4.1: Detailed view of System Architecture.

4.2 System Design

This chapter will take us through each component listed in Figure 4.1 and explain its role in the system as well as the technologies used in its implementation.

In the design stage of this system there were several full design patterns considered, namely the *Observer* [46] pattern in the form of the Model-View-Controller (MVC) architecture, as seen in [58] and [27]. In such a system the three components are maintained as completely separate modules: the model, the view and the controller. MVC is a classic design pattern often used by applications that need the ability to maintain multiple views of the same data. The MVC pattern hinges on a clean separation of objects into one of three categories: models for maintaining data, views for displaying all or a portion of the data, and controllers

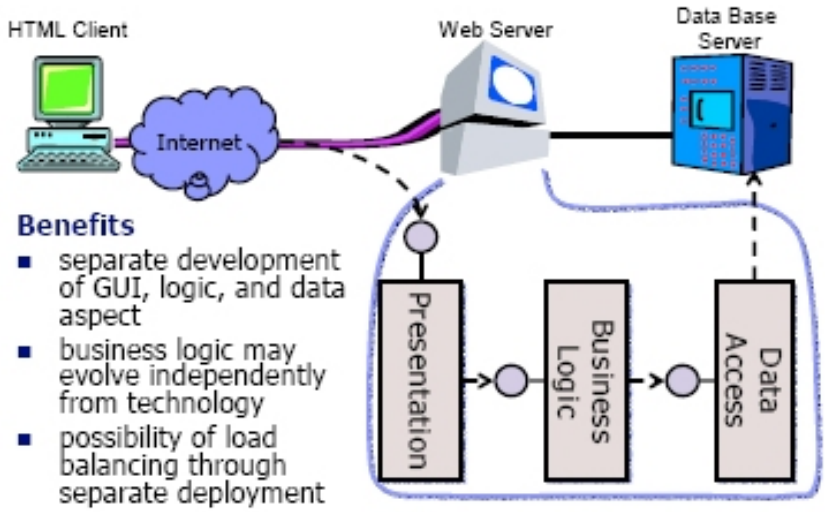


Figure 4.2: Standard Three Tier Architecture

for handling events that affect the model or view(s).

Because of this separation, multiple views and controllers can interface with the same model. Even new types of views and controllers that never existed before can interface with a model without forcing a change in the model design.

In all software design, there is a trade-off between developing quickly and getting the product working on the one hand, and using detailed and rigid design patterns to allow for easy extensibility and maintainability on the other. For the AdRec system it was decided not to stick rigidly to the MVC pattern, as it would too time-consuming and largely unnecessary for this 'once-off' experimental system, but instead to model the system from the operation diagram in Figure 4.1. This can be reduced down to a standard three-tier software development architecture shown in Figure 4.2 by taking the dark shaded portion of Figure 4.1 as the front-end, the lighter shaded section as the middleware, and the non-shaded section as the back-end.

4.3 Data Issues

The biggest obstacle to the success of this system was the lack of datasets available on which to test the system. For the regression functions to be really reliable, many more datasets would need to be incorporated, in order that a smoother, and more reliable graph could be obtained, without the need for interpolation of data points.

Each of the datasets that were used in the system were received in completely different formats (as varying format text files). A parsing component had to be

developed to integrate each set into the system.

4.3.1 The Parser Module: Integrating Each Dataset

The five datasets used were all received as text files, and they were parsed according to their individual formats using the Java StringTokenizer class (Example code shown below) and a script written in PERL.

```
for (int i = 0; i < max; i++) {
    String line = in.readLine();
    //tab as a token
    StringTokenizer temp =
         new StringTokenizer(line, "\t");
    while(temp.hasMoreTokens()){
        next = temp.nextToken();
        }
```

4.3.2 Persistent Storage: MySQL Database

It was decided that MySQL 4.3.2 would be used for persistent storage in the system. Reasons for this include the fact that MySQL is a very fast, multi-threaded, multi-user and robust SQL (Structured Query Language) database server. MySQL is a relational database management system. MySQL software is Open Source and the licensing is free. The MySQL query language is also easy to use and very well documented.*

All of our datasets were stored in a very simple, single table format of the form: Userid, ItemId, Rating. (In the PTV dataset, which just contained a list of TV programmes as a user profile was given a 1 if the TV programme was present in the profile, and a 0 otherwise.)

JDBC Technology

The JDBC (Java Data-Base Connect) API is the industry standard for database-independent connectivity between Java programs and a wide range of databases. The JDBC API provides a call-level API for SQL-based database access. JDBC technology allows you to use the Java programming language to exploit "Write

*http://dev.mysql.com/doc/

Once, Run Anywhere" capabilities for applications that require access to enterprise data.

The JDBC API makes it possible to do three things: establish a connection with a database or access any tabular data source; send SQL statements; and process the results. It is shown in the UML of Figure 4.3 how the database is interfaced with the rest of the system via a database manager class that handles all of the JDBC details for the system. Some example java code from this class is shown below.

```
public class DBaseManager {
    ResultSet rs;
    ResultSetMetaData mdata;
    int numrows, numcols;
    Connection connection;
    DBConnect dbconn;

    public DBaseManager(){
        dbconn = new DBConnect();
        connection = dbconn.getCon();
    }
```

4.3.3 Classification Module

Once the datasets were parsed into the required SQL format, we performed simple counting operations for the number of users, items and total ratings in the dataset. From these values we could calculate the rating density and sparsity values using Equation 2.4, and it was straightforward division to calculate the user-item ratios. This classification data is used to calculate the regression function (which models the relationship between algorithm performance and the dataset classification values of the training sets). This function is then used to predict the performance of the recommendation algorithms on a new dataset, based only on a lightweight classification of that data.

4.4 Software Issues

This section deals with the software issues in the development of the AdRec system from both design and implementation perspectives, looking at the design technologies and programming languages used.

4.4.1 Unified Modelling Language (UML)

UML is a formalised and unified manner of modelling the design and implementation of an object-oriented (OO) software application. There are a number of different model types which address the different issues that confront an OO designer during the course of a project lifecycle. This section details the most important UML diagrams.

Use-Case Diagrams

Use-case diagrams identify the core functionality of a system by placing it within the context of various external entities which interact with the system. In most cases, these entities are called "actors", and are represented diagrammatically by stick-figure characters. The main goal of a use-case model is to identify all of the system's functionality. These models are of most importance during the requirements analysis stage of a project's lifecycle. One important feature of use-case diagrams is that they can be understood easily by both professionals and lay people.

Class Diagrams

Class diagrams (Figure 4.3) are the component of UML that shows the internal structure of classes and objects in the system and the external relationships that exist between them. Traditionally, class diagrams have been used in some manner or other in all OO applications. Because of this, they constitute a central component of UML and support a wide range of modelling concepts. A common result of this broad range, however, is that over-detailed, elaborated modelling, and hence confusion arises with these diagrams.

Pooley and Stevens [80] forward two simple objectives for class diagrams. Firstly, "Build, as cheaply as possible, a system which satisfies our current requirements", and, secondly, "Build a system which will be easy to maintain and will easily adapt to future requirements". In order to meet the first requirement, we should build a class model in which every piece of behaviour which is required of the system is provided for, in a sensible manner, by the classes we design.

Interaction Diagrams

These diagrams illustrate the the dynamic actions of the system. Use-cases, statecharts and activity diagrams also model dynamic features of the system, but interaction diagrams specifically model the messages which can be sent between the various components in a system. There are two main types:

- Sequence Diagrams, which illustrate the temporal ordering of messages;
- Collaboration Diagrams, which emphasize the structural organisation of the objects that send and receive these messages.

Packages

As systems expand, so too do the number of classes, interfaces, diagrams, etc. In order to handle this increasing complexity, it is necessary to break the system up into manageable subsystems called packages. This is UML's general-purpose mechanism for organising the system in a coherent manner. The package concept can be applied to any modelling element supporting UML. In the AdRec system, there are four defined packages, one each for the filtering system, front-end, data components and demonstration system.

Activity Diagrams

Activity diagrams indicate the flow of control from activity to activity (as distinct from object to object). An activity is defined as the state of actually doing something, and ultimately results in some action which usually changes the state of the system. These diagrams are essentially flowcharts, and are most useful when illustrating workflow or parallel processing behaviour.

State Diagrams

These diagrams illustrate the series of events which an object goes through in its lifetime. In contrast to interaction diagrams, which model behaviour of a number of objects, state diagrams are only concerned with one specific object and how its state changes in response to specific events.

Implementation Diagrams

There are two implementation diagrams in UML:

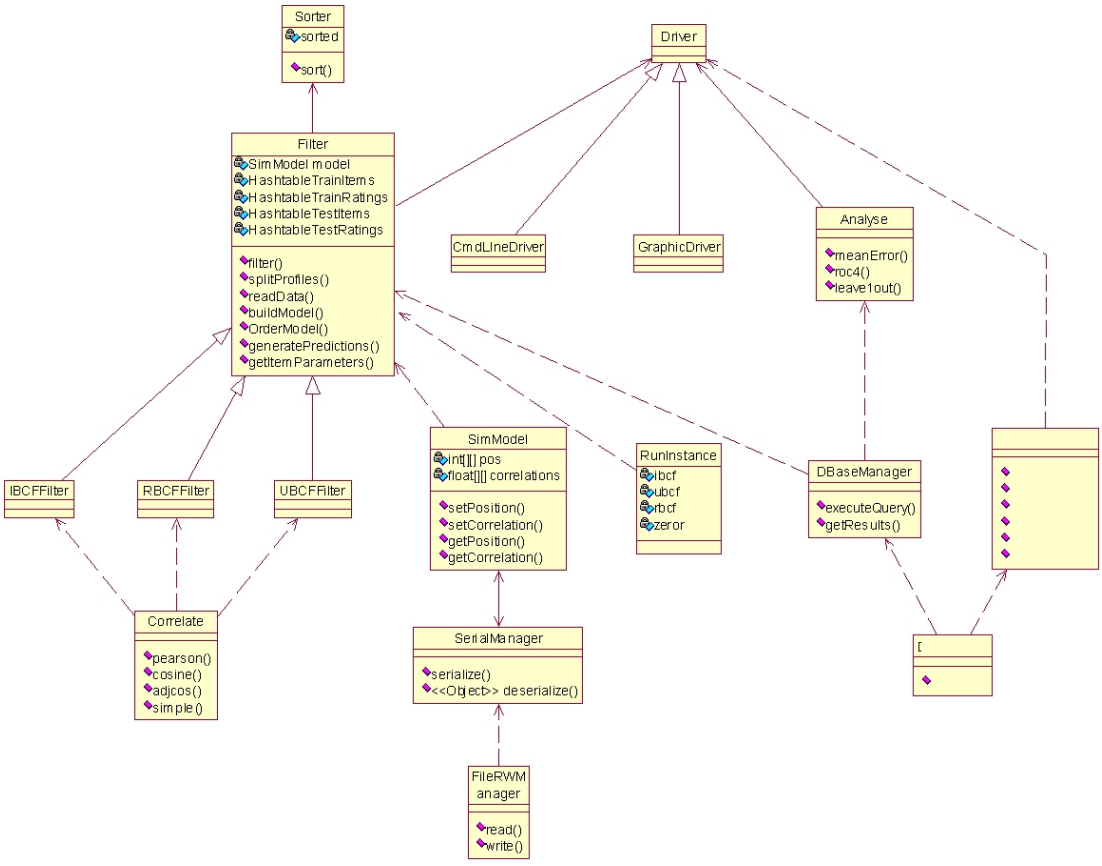


Figure 4.3: A Basic UML Class Diagram of the AdRec System.

- Deployment Diagrams: These illustrate the physical relationships between hardware and software components in the system. Each node in an deployment diagram represents a computational unit, usually a hardware component.

- Component Diagrams: These illustrate the various componenets in the system and their various dependencies. A component usually corresponds to a module of code, or possibly a package.

4.4.2 Programming Language

The majority of the AdRec system is coded in the Java programming language. Java is a clever combination of features that have been taken from older languages and adapted to its own needs. The following is a list of some of these features and where they originated:

- Basic object-orientation and many syntactic features from C++;
- Interfaces (as opposed to implementations) from Modula-2, Ada, IDL (used in DCE, COM, CORBA);
- Byte code/virtual machine from Pascal P-code;
- Absence of an explicit pointer notion from Oberon;
- Garbage collection from LISP/Smalltalk/Oberon;
- Threads from existing C/C++ libraries.

Java is easier to use than C++, and above all, it is portable, Web-oriented, and provides an ever growing wealth of class libraries for new application domains, including, and importantly for this work, the collaborative filtering domain. The platform independent nature of Java was an important feature in its selection, as much of the AdRec system design and implementation was to be performed under a Microsoft Windows environment, and testing the system was to be carried out in a laboratory on multiple machines with various UNIX-based operating systems. Java saved time that would have been spent recompiling source files for each different environment.

Java has a minimal support for mathematics functionality, and does have some other disadvantages for application programming, such as the memory and runtime issues dealt with in the following sections.

4.5 Development Environments

Some of Java GUI implementation for this system was carried out on a machine running a Red Hat Linux operating system, using XWindows to support Java SWING on a Sun Microsystems server running Solaris. However, the majority of coding for this system was carried out using the NetBeans Java IDE[†] on a Windows 2000-based PC with a 1.8GHz processor and 512MB of RAM.

4.6 Middleware: Algorithm Design and Implementation

This section gives details on implementation for each specific recommendation algorithm, including the baseline *Zero-R* algorithm and the WEKA suite of clas-

[†]www.netbeans.org

sification tools from which we acquired the source code for the *Zero-R* algorithm.

IBCF Implementation

The item-based CF algorithm was implemented in Java, and we followed exactly the implementation carried out in [65], and detailed in Section 2.5.4 of this thesis.

UBCF Implementation

The following is the pseudocode illustrating the important parameters in our implementation of the User-Based CF algorithm. These include which database to use, what percentage split between the test and training data, and the number of profiles and items to use from the database.

```
public class UserFilter extends Filter {
    public UserFilter(String s, int u,
                      int i, int ttr, int k) {
            users = u;
            items = i;
            ttRatio = ttr;
            knn = k;
            dbName = s;
        }
```

RBCF Implementation

We used a version of the Apriori data mining algorithm from Christian Borgelt's website[‡]. This algorithm is discussed in detail in [7]. Output from this algorithm is shown in Figure 4.6, to be of the form

$$< ItemName > \rightarrow < ItemName > \Rightarrow < \%confidence, \%support > .$$

We built a similarity model between all the items in the system by multiplying the support by the confidence for each rule generated by the Apriori algorithm. This technique is taken from O'Sullivan's work in [56].

[‡]http://fuzzy.cs.uni-magdeburg.de/borgelt/apriori.html

```
public void runApriori(){
        System.out.println("Starting Apriori...");
        try{
            String cmd = "apriori.bat";
            Process p = Runtime.getRuntime().exec(cmd);
            }
            ...
```

The Zero-r Algorithm

One of the most primitive classification schemes is called Zero-r: it simply predicts the majority class in the training data if the class is categorical and the average class value if it is numeric. Although it makes little sense to use this scheme for prediction, it can be useful for determining a baseline performance as a benchmark for other learning schemes. Sometimes other schemes actually perform worse than Zero-r indicating serious overfitting, (ie modelling the training data to too great an extent). We will see an example of this in Figure 5.3 with the Movielens dataset. The Java code used in AdRec for the Zero-r algorithm was taken from an open-source implementation (part of the WEKA system described in the next section).

The WEKA Classification System

WEKA[§] [3] is an open source collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, generating association rules and visualisation. It is also well-suited for developing new machine learning schemes. The WEKA software is issued under the GNU General Public Licence.

4.7 The AdRec System Interfaces

Considering the research and findings of Swearingen and Sinha [78] regarding the importance of all aspects of recommender systems, the user interface is designed

[§]http://www.cs.waikato.ac.nz/ml/weka/

with ease of use and overall user-friendliness in mind.

The interface to the AdRec system was developed in three phases. Each of these is outlined in the darker shaded area of Figure 4.1. Firstly, a basic command line interface was implemented, then the system took advantage of Java SWING components and a GUI was developed. For demonstration/publicity purposes we also developed a Web-based front end using JSPs and Java Servlets.

4.7.1 Command-Line Interface

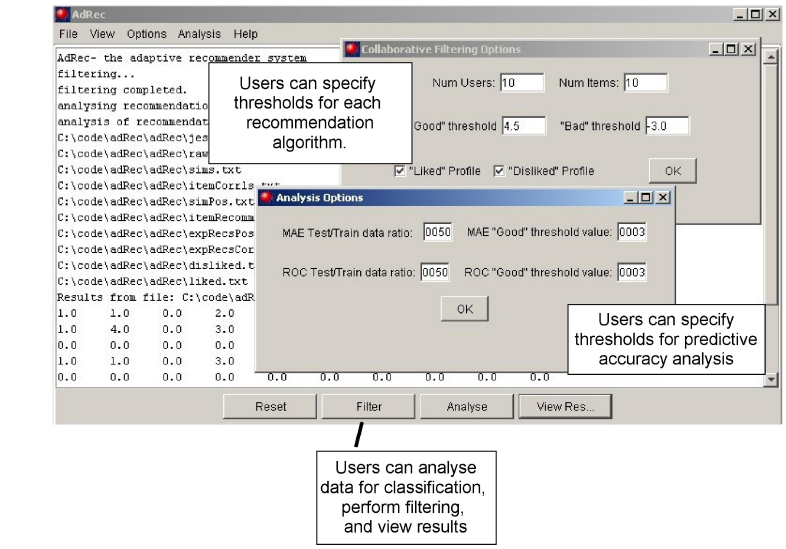
This simple interface provided all the basic functionality, but required complicated parameter lists as input. This interface was adequate at first, but as the system got more complicated, batch files/shell scripts were required. This prompted the development of the graphical interface to the system.

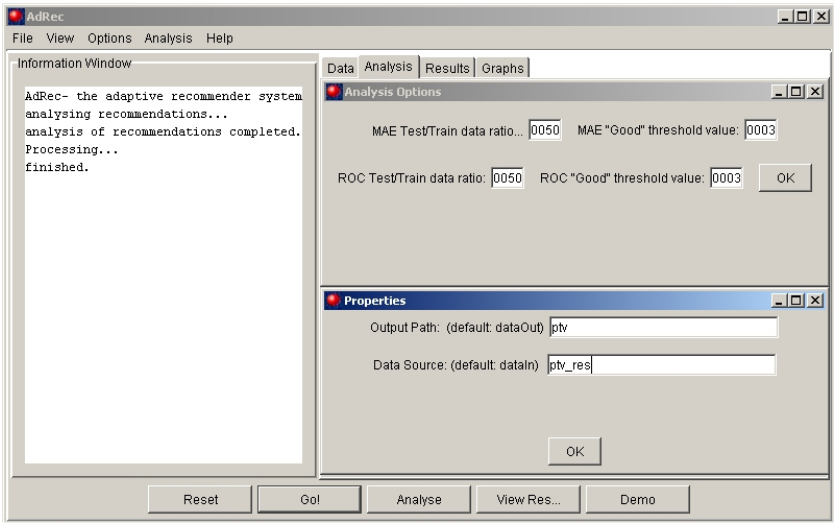
4.7.2 The AdRec Swing GUI

Figure 4.4 below shows the Graphical User Interface for the training phase of the system. A user can control all the variable parameters from this interface. For example, changing from Pearson correlation to Overlap coefficient in the collaborative filtering options or select the database the system operates on. The automated testing parameters can also be varied from here. A user can change between precision/recall and MAE metrics etc. The GUI is implemented using Java's SWING components. Note: All system functionality can also be controlled from a command line.

```
public GUI1() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);

try {
    jbInit();
    }
catch(Exception e){
        e.printStackTrace();
}
}
```





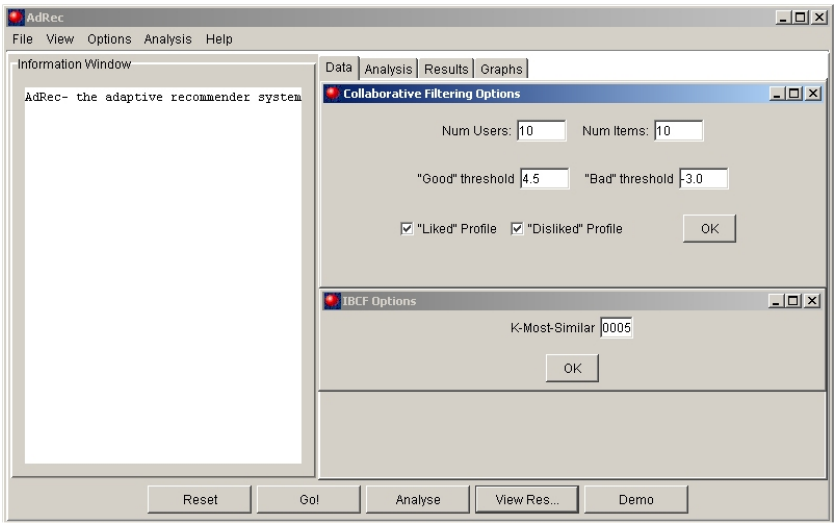


Figure 4.4: The AdRec Front-End

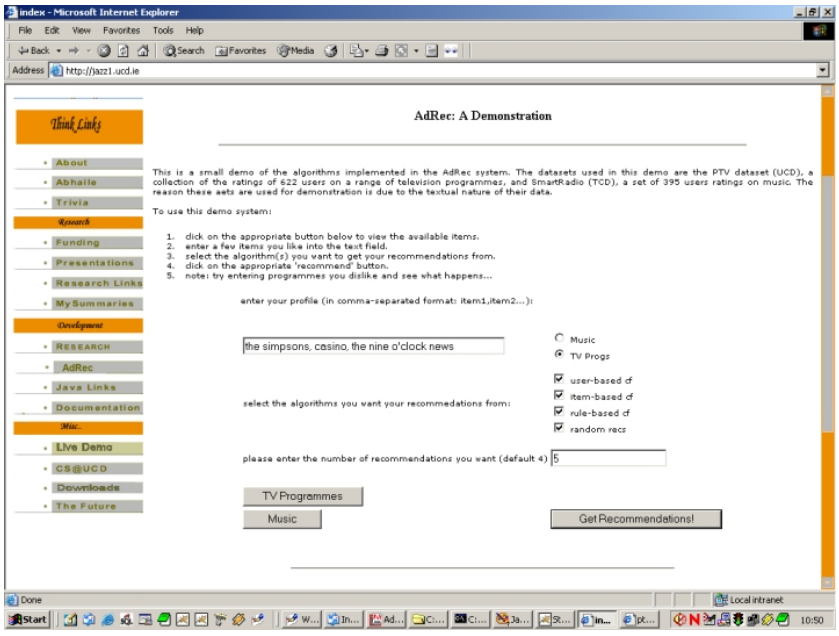


Figure 4.5: The AdRec Web Application: Algorithm/Profile/Dataset Selection

4.8 The AdRec Web Application

In this section we introduce the Web-based version of AdRec, looking at both design and implementation and the technologies used in developing the system. We also look at some screen-shots of the live system, explaining each important feature.

4.8.1 Motivation and Design

For the purpose of demonstrating the AdRec system, an online version was created, showing the end product recommendations of the algorithms in AdRec. The Web demo works on two datasets, PTV and SmartRadio.

In short, users can click on the appropriate button to view the available items. then enter a few items as their profile. There is an option to select the algorithm(s) used to produce recommendations, and an option to specify a number of other features, for example, the number of recommendations required from each algorithm.

4.8.2 Implementation

The demo system is implemented using a standard three tier architecture, illustrated in Figure 4.2. The data is stored in a MySQL database, and pre-built

collaborative filtering similarity models are stored in serialised Java files. The middleware of the demonstration system is simply the online sections of the filtering algorithms described in Sections 2.5.3, 2.5.4 and 2.5.5.

4.8.3 Hosting the Application

AdRec was hosted on an Apache-Tomcat Web server and is available on the intranet at University College Dublin.[¶].

Apache-Tomcat version 4.1 was used in the system. The main motivations behind this server choice was that the NetBeans IDE[‖] contained a version of this server pre-installed, with added functionality for ease of deployment, etc. This made the Web-development task much easier, since deployment of JSPs and Servlets could be handled in just one click, as opposed to the usual lengthly manual deployment process.

4.8.4 Java Servlet Technology

Servlets are the Java platform technology of choice for extending and enhancing Web servers. Servlets provide a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs. And unlike proprietary server extension mechanisms (such as the Netscape Server API or Apache modules), Servlets are server and platform independent.

Servlets have access to the entire family of Java APIs, including the JDBC API for accessing enterprise databases. Servlets can also access a library of HTTP-specific calls and receive all the benefits of the mature Java language, including portability, performance, reusability and crash protection.

Today, Servlets are a popular choice for building interactive Web applications. Third-party Servlet containers are available for Apache Web Server, Microsoft IIS and others. Servlet containers are usually a component of Web and application servers, such as IBM WebSphere, Sun Java System Web Server, Sun Java System Application Server and others.

4.8.5 Java Server Pages (JSPs)

The purpose of JSP is to provide a declarative, presentation-centric method of developing Servlets. The JSP specification itself is defined as a standard extension

[¶]http://jazz1.ucd.ie/

[‖]http://www.netbeans.org/

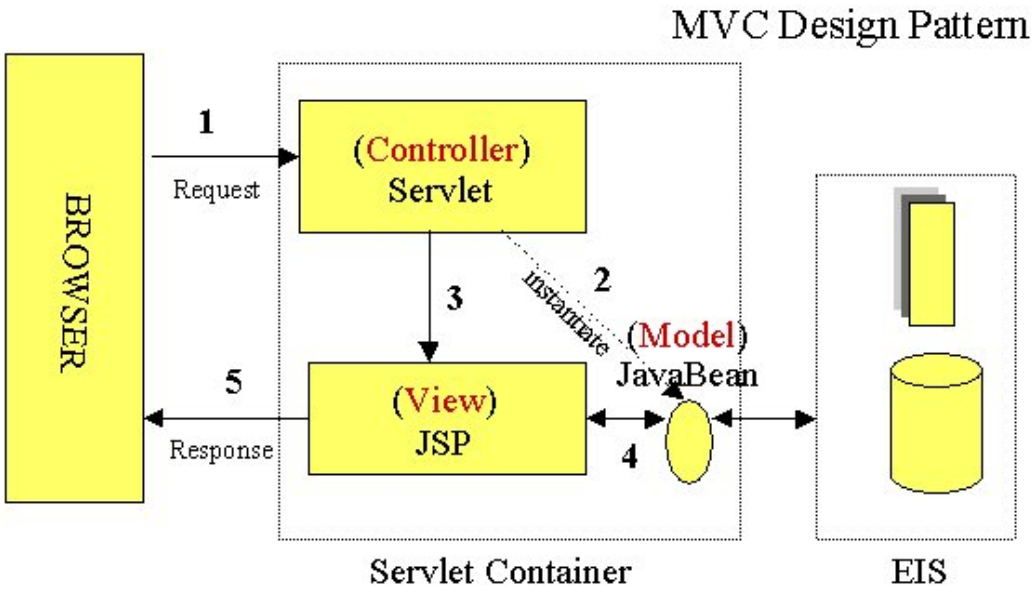


Figure 4.6: The JSP-Servlet Implementation of the Model View Controller Design Pattern.

on top of the Servlet API. Consequently, under the covers Servlets and JSP pages have a lot in common. Essentially, JSPs are HTML pages with functionality for embedded Java code. This allows easier abstraction of presentation details from business logic in the system. Separation of these keeps the system more modular and allows for easier maintenance. In the Web deployment of the AdRec system, JSPs are used in a loose interpretation of the Model View Controller architecture** presented in Figure 4.6.

The advantage of this architecture is that there is no processing logic within the presentation component itself; it is simply responsible for retrieving any objects that may have been previously created by the controller, and extracting the dynamic content within for insertion within its static templates. Consequently, this clean separation of presentation from content leads to a clear distinction between the roles and responsibilities of front-end and middleware design. Another benefit of this approach is that the front components present a single point of entry into the application (Marked as *1* and *5* in Figure 4.6), thus making the management of application state, security and presentation uniform and easier to maintain.

**http://java.sun.com/developer/onlineTraining/JSPIntro/contents.html#JSPIntro2

4.8.6 Other Technologies Used

Other technologies that were used in the Web-application were JavaScript and Cascading Style Sheets. Both of these technologies can be introduced from this documentation[††]. JavaScript is a lightweight client-side programming language, run in the browser. In our system this was used mainly for form validation.

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g. fonts, colors, spacing, etc) to Web documents. Essentially, this technology allows all the style information for every HTML page in the system to be maintained through one external `.css` file. This allows much simpler modification of the appearance of the AdRec system.

4.8.7 The Web-Application Interface

Using the Web-development technologies discussed in the previous sections the application in Figures **??**, 4.5 and 4.9 was developed. Figure **??** shows the introductory screen to the application Web site, providing various links to research, project information, resources and the demonstration system. The online system provides data from two of the experimental sets, PTV and SmartRadio. The reason for this is that the other sets are numeric, whereas items have nominal values in these sets. Figure **??** also shows an example of the data selection from the list of available items in the SmartRadio dataset.

The top screen-shot in figure 4.9 shows some of the options available to the user. On this screen, the user can select the algorithm from which the recommendations are made. In the web application we have made all four of our algorithms available. A user can also select the number of recommendations to receive from each algorithm (the default is four); select which of the available datasets to receive recommendations from; and select a profile of items from these datasets. The bottom screen-shot in figure 4.9 shows the algorithm output being presented to the user. In this example there are five recommendations of TV programs recommended by each of the four algorithms. As an addition to the recommendations section, users have the option to rebuild the similarity models for each algorithm upon which their recommendations are based.

4.9 Summary

In this chapter we have dealt with the design and implementation issues of the AdRec system. We discussed system design from a UML perspective, and we ex-

[††]www.w3schools.com

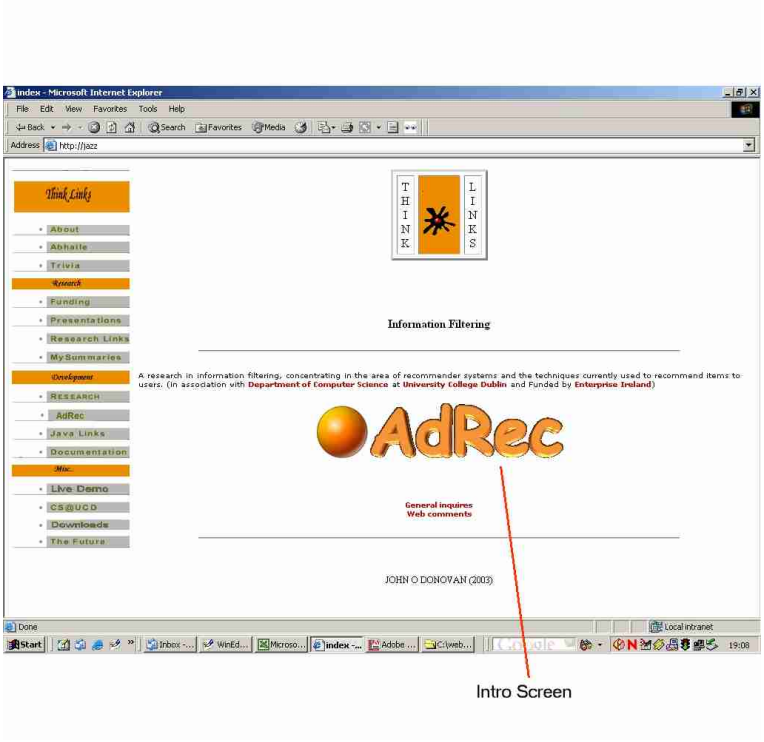


Figure 4.7: The AdRec Web Application: Introductory Screen.

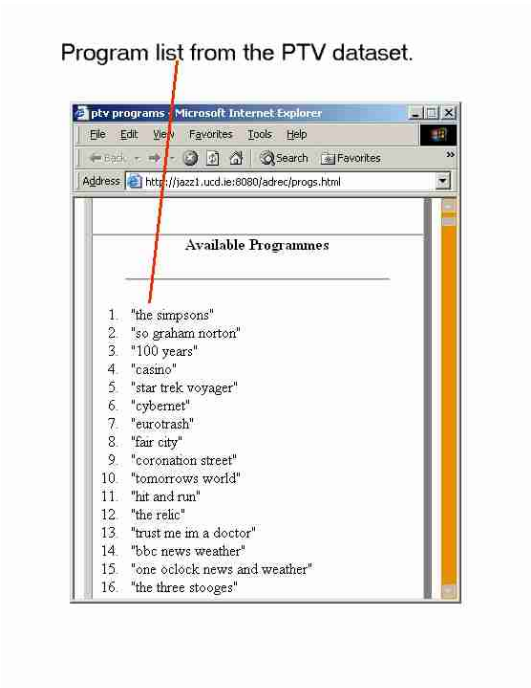


Figure 4.8: The AdRec Web Application: PTV Data.

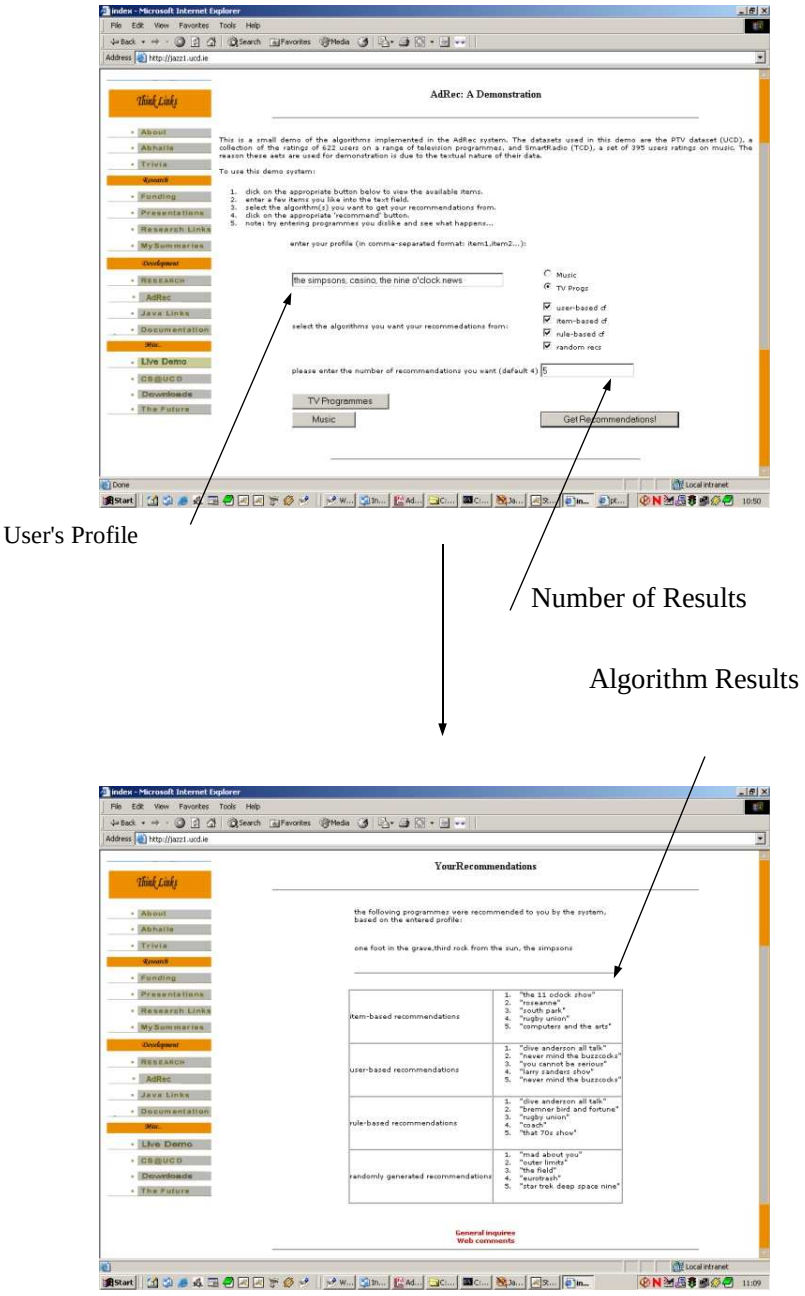


Figure 4.9: The AdRec Web Application: Presentation of Results.

amined the various technologies which were used in the implementation of the core system, the persistent storage, the GUI, and the Web-based application. Screenshots of both the GUI and the Web application were presented, corresponding explanations and discussions.

Chapter five presents the experimental analysis and evaluation of the AdRec system, including explanation of each experiment undertaken, and presentation of the resulting data.

Chapter 5

Experimental Evaluation

5.1 Introduction

In the previous chapter we outlined our implementation goals, and showed how they were met using various techniques and technologies. This chapter deals exclusively with the testing procedures for the system, and the empirical results of that testing. Specifically, we give a background on the testing metrics used for our experiments, describe some of the data issues that were noted, and introduce two separate experiments for testing individual algorithm performance and exploring the effects of varying density and neighbourhood size on predictive accuracy. We then present the results of these experiments with a discussion in respect to each dataset in turn. Finally, we describe a third experiment to test the validity of our approach to adaptive recommendation, and present our primary results and discussion.

5.1.1 Statistical Accuracy Metrics

Common metrics for evaluating the accuracy of a prediction algorithm can be categorised under two main headings: Statistical Accuracy and Decision-Support metrics [45].

Statistical Accuracy is obtained by comparing a set of predicted values with a set of user-provided values (Test Data). Mean Absolute Error (MAE) is the average absolute difference between predicted ratings and actual ratings. Precision/Recall is also used for statistical accuracy testing. In the AdRec system, decision support functionality for Precision/Recall and for Receiver Operating Characteristic (see Section 5.1.2) were added but never finalised. The experimental analysis in this thesis therefore focusses on statistical accuracy in the form of

Mean Absolute Error (MAE) metrics.

5.1.2 Decision Support Metrics

Decision-Support metrics measure how well a system helps users select high-quality items. Receiver Operating Characteristic (ROC) is used to measure decision support. ROC sensitivity is found by calculating the area under a ROC curve. A ROC curve is a plot of sensitivity versus specificity for a predictor, where sensitivity is the probability that a good item is accepted by the predictor, and specificity is the probability that a bad item is rejected by the filter. There is a level of subjectivity involved, in that the area under this curve is dependant on what is considered to be a "good" recommendation. To make the ROC graph, the X-axis is 1 minus the specificity (the false positive rate) and the Y-axis is the sensitivity (the true positive rate).

5.1.3 Data Issues/Comparison with Existing Systems

Good et al. [25] analyse the predictive ability of collaborative filtering and information filtering. Information filtering focusses on the analysis of item content and the development of a personal user interest profile. They find that the combination of both methods leads to the most useful recommendations. The literature mentioned in [25] shows that various approaches have been proposed and compared. Several contributions use the mean absolute error as a performance measure. However, in our opinion a fair comparison of all of the experimental results presented in the literature is elusive, since there are many different variables in CF systems and different subsets of the original data sets. Although the literature considered indicates that the choice of the methodology adopted significantly influences the quality of recommendations, we suppose that some of the results presented in recommender system studies might be a result of the specific design (data selection) chosen. For this reason we did not attempt an empirical comparison with existing collaborative recommendation systems.

5.2 Experiment 1: Predictive Accuracy with Varying Neighbourhood Sizes (k-nn)

This experiment has two motivations: The first motivation is to find optimal values for k for our final experiment; and secondly, to record individual algorithm performance for the purposes of developing our regression functions. It has been

recorded [65] that there are generally peaks in the curves for the user-based and item-based algorithms at a neighbourhood size of around 30–40. We vary this neighbourhood size between 10 and 100 in intervals of 10 in order to ascertain an optimal value for the number of neighbours. We perform a standard 'leave one out' analysis in which we hide the test portion of the user's profile and try to predict the hidden test set ratings by training our algorithms on the rest of the profile.

To assess the predictive accuracy, we then compare our predicted values for the hidden set with the actual values. We calculate the mean absolute error between the predicted ratings and the actual ratings.

Results of this experiment are presented in Figures 5.1, 5.2, 5.3 and 5.4, for Eachmovie, Jester, Movielens and the PTV dataset, respectively. It is clear from the results in these graphs that varying the number of neighbours in a collaborative filtering algorithm has a direct impact on accuracy. In general, the effect is that % accuracy of the algorithms increases up to some threshold value, and then starts to level off. This is most likely due to the fact that as the size of the neighbourhood increases, the similarity of the peers to the user requesting the recommendation is reduced. For example, if we were to allow the entire user-base to make up the peer group (without using similarity), then we would simply be recommending the most popular items to every user in the system.

Keeping processing time and throughput (number of recommendations per second) in mind, we arrived at optimal values for k for each recommendation algorithm. These values are the first "reasonably" high local maxima from the graphs. In some cases there were higher values, but these would have involved computing very large neighbourhoods, which is computationally expensive. The optimal values for each algorithm on each dataset are presented in Table 5.1, and are in the range of 20–50.

We find also that there is a difference in the relative performance of our algorithms over the datasets, which is needed to make the idea of an adaptive recommender system work. The optimal k-values for each algorithm from this experiment are used in experiment 3. As expected, our trained recommendation algorithms generally perform better than the baseline Zero-r algorithm, which, obviously has a straight line in each of the graphs, since it has no dependency on k. The performance information from this experiment is also used in calculation of the linear regression function in Appendix A. This is the function upon which the system bases it's algorithm performance predictions.

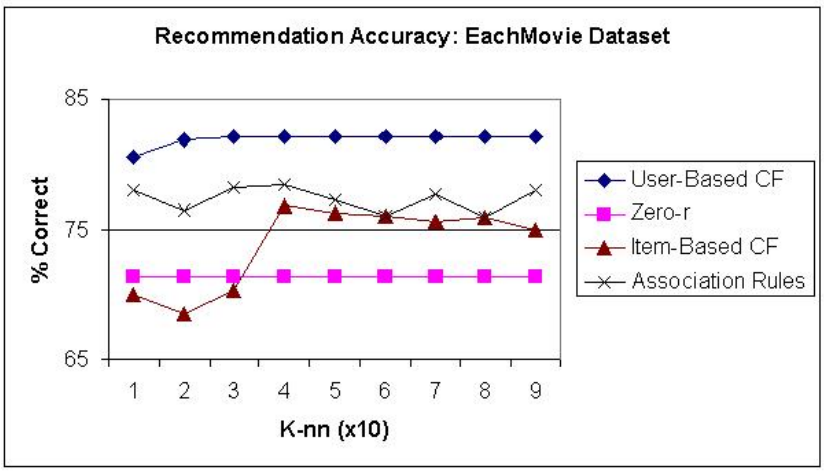


Figure 5.1: Testing neighbourhood sizes in EachMovie.

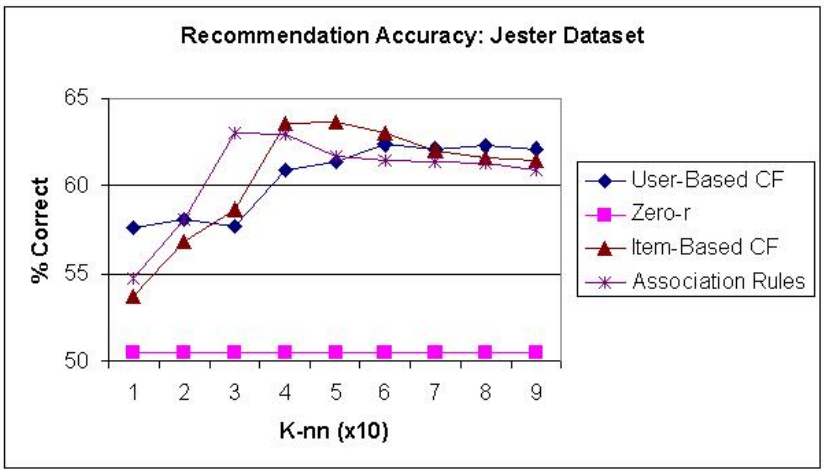


Figure 5.2: Testing neighbourhood sizes in Jester.

5.3 Experiment 2: Predictive Accuracy with Varying Density

Density of the training data can be experimentally controlled by varying the test-train ratio of the experimental data. In Figures 5.5, 5.7, 5.8 and 5.6 we can see the results of this variation. It is clear that across all of our datasets there an increase in predictive accuracy as the percentage of training data increases (with

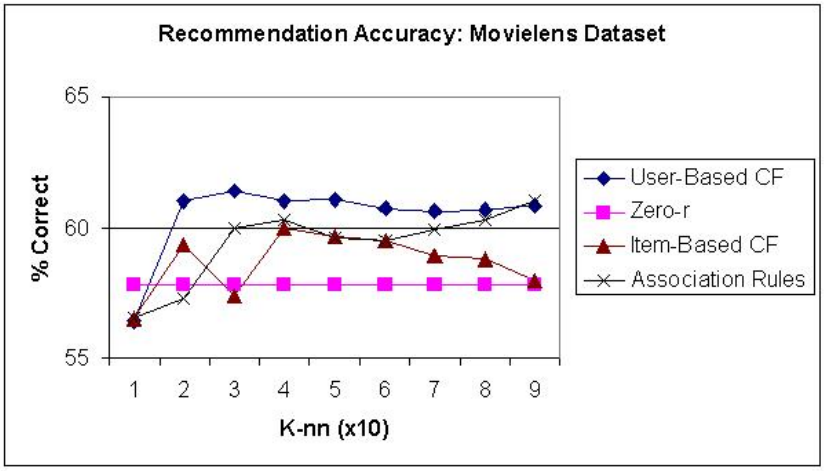


Figure 5.3: Testing neighbourhood sizes in MovieLens.

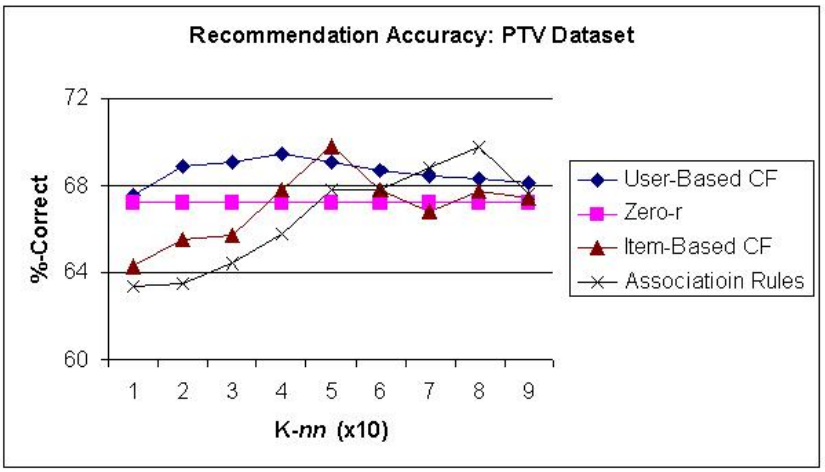


Figure 5.4: Testing neighbourhood sizes in PTV.

the exception of the RBCF algorithm on the Jester dataset). This indicates that the algorithms are in fact learning well.

Ideally, we would like to have shown that the Item-Based or Rule-Based algorithms performed relatively better at lower test-train ratios and were less affected overall by the change in the training set size. This would have told us something about the quality of the information captured by the Item/Rule-Based algorithms. This expected good performance at low test-train ratios was to indicate that this

Algorithm	*MovieLens*	*Jester*	*Eachmovie*	*PTV*
IBCF	40	40	40	50
UBCF	30	40	30	50
RBCF	20	30	40	40

Table 5.1: Optimal Values Discovered for Neighbourhood Size (k).

filtering strategy could have been employed as a method to solve the sparsity problem with User-Based collaborative filtering. Also, since better predictions can be made with less training data, it saves on memory (small model sizes) and will generate faster recommendations. This was our expected result, but the empirical evidence has shown that this is not the case.

Apart from the obvious increase in accuracy with training set size, the only other notable feature is the drop-off in performance accuracy at extremely high train-test ratios. It has been suggested in [3] that this phenomenon may be due to serious *overfitting*: modelling of the training data so closely that the algorithm loses sight of the real picture.

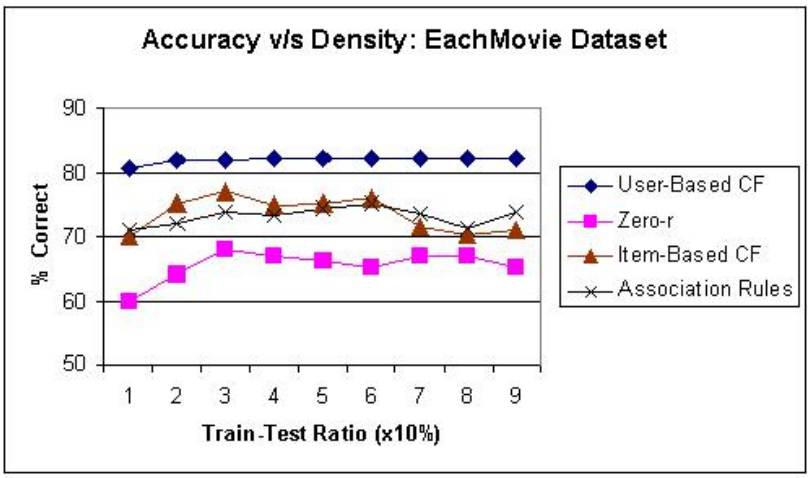


Figure 5.5: Density Testing in EachMovie

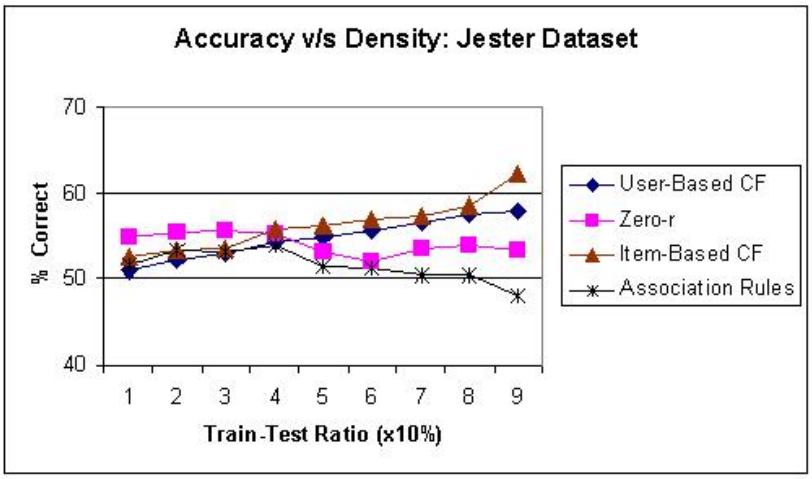


Figure 5.6: Density Testing in Jester

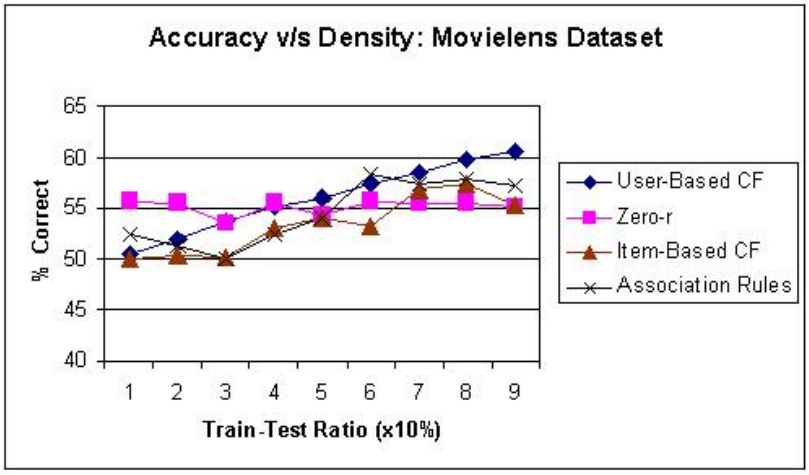


Figure 5.7: Density Testings in MovieLens

5.4 The Zero-r Algorithm

For comparison purposes we also test a primitive algorithm known as Zero-r. This simply predicts the majority class in the training data. This algorithm itself is not

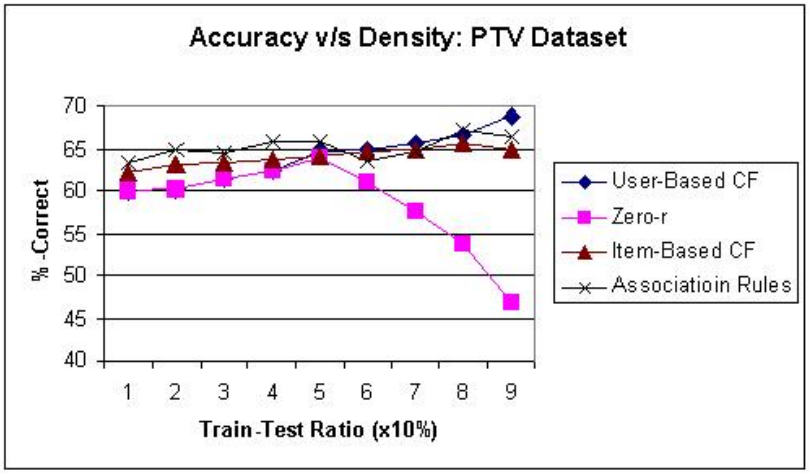


Figure 5.8: Density Testing in PTV

much use as a functional predictor, but serves well as a benchmark for our other prediction algorithms. In one experimental case, at very low train-test ratios on the Jester dataset, this algorithm does perform better than the learning algorithms, but generally it performs much worse.

5.5 Movielens

On the Movielens dataset, the UBCF algorithm is strongly affected by changes in neighbourhood size. At low k values, UBCF has a predictive accuracy of around 50%, which is a very poor performance. This predictive accuracy increases steadily to well over 60% as the training set increases in size. The Item-based algorithm starts with a similarly poor performance, and does undergo a performance increase as the size or the training set is increased. This increase is not nearly as steady however and there are several dips in the Item-based performance curve for MovieLens. The Rule-based algorithm has a similar performance to that of the Item-based approach. Overall, on the MovieLens dataset, the best performing algorithm was the User-based algorithm with a 90% training set. There is not a huge difference in the performances of the recommendation algorithms on this dataset, in contrast to the EachMovie set, for example. This could be due to a number of factors, such as dataset sizes or densities for example.

5.6 EachMovie

In the EachMovie dataset, there are further relative differences in performance between our algorithms with respect to data. Unlike all of the other test sets, the accuracy of IBCF and RBCF follow roughly the same trends, and have only a marginal difference of approximately 1%. The optimal recommender configuration for this set was found to be at a density of 30 and neighbourhood size of 30, using the UBCF algorithm, which produced very high accuracy (approx. 81%) even at low train-test ratios. The user-based algorithm did not display much learning, apart from an initial jump of about 2% up to a training set size of 20%. The Zero-R algorithm was outperformed across all k values by around 10% by IBCF and RBCF, and 20% by UBCF.

We can see from the dataset classification table in Chapter 3 (Table 3.1) that EachMovie is highly sparse (97.6%). From this information we would expect the performance values for our recommendation algorithms to be very poor, when in fact they are the best results from all our datasets. To explain this, we need to look at the size of the dataset. EachMovie is *huge* compared to the other datasets. (with 118 million possible ratings). This means that the algorithms have more data to train on than the other sets, and consequently, have a better performance. In the EachMovie dataset, the Zero-r algorithm performed much worse than all the other algorithms.

5.7 Jester

Jester shows more interesting results for our algorithms. The best performance here is given by the IBCF algorithm at high k values, and marginally beats the User-based approach over all train-test values. This may be due to the fact that the clusters of people with similar tastes in jokes are larger than those with similar tastes in movies, or TV programmes, since there are more types of movies than jokes. Jester has a sparsity value of 46.41 (see Table 3.1 in Chapter 3), making it the least sparse of all. The Rule-based algorithm has a notable drop-off in accuracy on the Jester dataset at high train-test values. It is explained in [56] that collaborative filtering algorithms based on association rules perform best over sparse data. The evidence in Figure 5.6 correlates with this theory, since the Rule based algorithm has performed much better on our more sparse datasets.

5.8 PTV

Our results do indicate that there is a relative difference in the performance of our algorithms with different data, but not as strongly as the results on the other datasets. On the PTV dataset, at neighbourhood size k of 10, UBCF performs better than IBCF by 2%. We can draw some similarities between the PTV dataset and the MovieLens dataset, in that the learning curve for all of the recommendation algorithms seems to be very consistent. On the PTV data (Figure 5.8), we can see that there is an increase in accuracy of about 10% on average over all of the algorithms as we increase the size of the training set from 10% to 9%. This once again reenforces the fact that the algorithms are learning well from the training data. Another point of similarity between these sets is that the difference in sparsity levels between these two sets is less than 1%. (Table 3.1)

5.9 Experiment 3: Evaluation of the Adaptive System's Predictive Performance

Our final experiment is designed to empirically test the performance of the adaptive recommendation system as a whole. Re-iterating the main goal of this thesis, we aimed to identify and classify the salient features of a number of collaborative filtering ratings datasets, and use this classification to compute regression functions to predict the performance of each of our three CF algorithms and a baseline prediction algorithm on a previously unseen dataset based only on it's classification values.

In this experiment we assess the performance of each individual algorithm on each of our four training datasets. For this experiment we maintain the optimal values for the algorithm parameters which we detected in our previous two experiments. For example, we use a neighbourhood size of 37 in the new dataset, which is the average from our previous experiments (since we found average optimal k-values of 37, 30, 37, and 46 for EachMovie, MovieLens, Jester and PTV respectively, cf Table 3.1 in Chapter 3). Further work could include adjusting the k-values in the new dataset based on the linear regression analysis, instead of simply using the average optimal value.

As in our previous experiments, we use MAE as a predictive accuracy metric and we use the previously described 'leave one out' analysis, in which we predict hidden items from the training profiles. We again take the MAE between the actual ratings and the ratings predicted by each algorithm. The results of this for each of our training algorithms are presented in figure 5.9, our best performer being

the User-based algorithm on the EachMovie dataset, with a predictive accuracy of 82%. The worst performing algorithm on our training datasets was the baseline predictor, Zero-r.

Next now we build our regression functions for each algorithm from this performance data. Actual calculation details of these functions is shown in full in Appendix A. We wish to show that the AdRec system can automatically apply the best performing algorithm to a new, previously unseen dataset.

In this relatively simple experiment, we classify the previously described SmartRadio dataset according to the metrics outlined in Table 3.1. It is important to note that this is a very lightweight classification and requires minimal knowledge of the actual data (the number of items, users and ratings).

We input the SmartRadio classification values into the predictive functions and get the following performance predictions in terms of predictive accuracy (Details of calculation are in Appendix A):

1. RBCF = 66.4%
2. UBCF = 65.2%
3. IBCF = 64.01%
4. Zero-r = 58%

The graph in Figure 5.9 clearly shows that the Rule-based CF algorithm (predicted as the best-performer by our regression function) does in fact have a better predictive accuracy than all of its competitors on the SmartRadio dataset. We note that there is a general overestimate of the performance of all of our algorithms on the SmartRadio dataset by our regression function. It is our theory that this is due to the extreme sparsity of the data. (99.98%) It has been shown in [56] that increasing sparsity has the direct effect of reducing the predictive accuracy of collaborative filtering algorithms.

On the upside, however, we have managed to predict not only the best performing algorithm for the new dataset, but we have correctly predicted the relative performance of *all* of our test algorithms. This is the main contribution of our work.

5.10 Summary

In this chapter we have provided an experimental evaluation of each of the algorithms in the AdRec system. We performed three experiments, one to ascertain

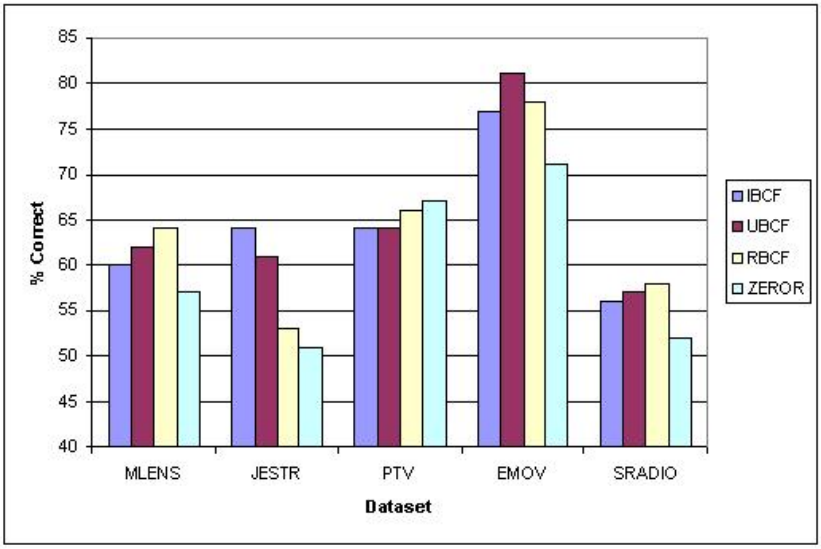


Figure 5.9: Performance Results

optimal values for neighbourhood sizes for our recommendation algorithms, another to examine optimal test-train ratios, and a final experiment to analyse the performance of the adaptive recommender system as a whole. We have found that the system can predict the relative order of performance of each algorithm on the unseen dataset based on it's classification values. We have presented results for all of our experiments provided a discussion and analysis of the results. The following chapter concludes this thesis and provides ideas on possible avenues for future work on the AdRec system.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Collaborative recommendation techniques are powerful tools which enable the delivery of relevant information without users being forced to trawl through irrelevant material to find what they require. Such techniques are relied on more and more as technology takes a stronger foothold in everyday living. These techniques also have great value to business as they can present a user/customer with items/products they are interested in, thereby generating more sales for the business with less hassle for the customer. Recommendation techniques are not without their flaws, however, especially collaborative techniques. Problems such as sparsity of data, latency and the early-rater problem lead to varying performances of recommendation techniques. In this research we have evaluated three recommendation strategies on different data. Our results indicate that there is a relative performance difference of these techniques over the datasets they were tested on.

We have harnessed these performance differences in a system which is enabled to adapt the best performing algorithm to a new dataset based on a relatively slight classification of the data.

The major contributions of this thesis are:

- We have defined a new regression model-based approach to the development of an adaptive recommender system;
- Empirical results show that the adaptive system has predicted the most appropriate algorithm in our tests;
- We have shown a comparison and evaluation of four recommendation algorithms across five collaborative filtering datasets.

6.2 Future Work

There are many directions in which to continue this research. One challenge to this area of research is the lack of experimental datasets of user ratings. Oddly enough, the majority of research into collaborative filtering techniques has been carried out in the movie domain, EachMovie and Movielens being two of the major publicly available CF datasets. This lack of dataset diversity does not favour our dataset classification experiments. A possible solution to this problem is to introduce data automatically generated by agents with specific 'tastes' (Tatsumi '99). Another, more interesting data source comes from an online jokes database know as 'punderland' that has been collecting user ratings on jokes for the past 18 months. In this system, users can enter new items (jokes), and provide ratings on the existing ones. This system contains more items than the popular Jester jokes dataset, and has an average of 15 user-ratings per item. One advantage of this system, from a research perspective, is that the ratings data is fresh, having never been used before for testing recommendation algorithms. It would be interesting to develop the AdRec system further, increase its user-base, and deploy it's learned recommendations to its users as a means of acquiring implicit/explicit user feedback. Such a dataset of ratings would be a significant contribution to the Information Retrieval community, as they seem to be in short supply.

In AdRec, the regression function which makes the final algorithm prediction, is calculated manually, (see Appendix A). This suits our small system well, but for any more detailed calculations, eg with an increased number of datasets, or using polynomial or cubic regression equations, it would be better to write code to automate the calculation of this function.

The recommendation algorithms in AdRec can be modified/reinvented to provide faster and more reliable recommendations. Two major factors during implementation and testing were memory usage and processing speeds. A non-object-based language such as C or PERL would be more suited to AdRec's number crunching than the current Java implementation. Hybrids of filtering techniques could be greatly improved by moving from the meta-level approach to a feature combination (e.g: case-based recommendation) or a cascade approach (where one recommendation algorithm can refine the recommendations of another). With a sufficiently fast implementation, it would be possible to introduce weighted hybrid techniques, as defined by the following algorithm:

```
for each prediction required{
   for each recommendation technique{
       getTopKRecommendations(user u);

       assignPoints();

           //the item with the nth highest
           // rating gets (k + 1)-n points )
           //The total rating of an item is
           // determined by the sum of its points.
   }
}
```

Individual techniques and hybrids can be included in this algorithm, if processing time can be significantly reduced.

6.2.1 Global Application of n-Fold Cross Validation

If the calculation of the regression functions in our system could be automated, it would become feasible to apply n-Fold Cross Validation to the AdRec system in a global sense. This essentially involves taking different train-test ratios from the system training datasets and, for each of n runs, develop a regression function for each algorithm. In this manner, we would be in a much better position from which to assess the performance of the regression-based adaptive recommendation engine.

6.3 Potential Applications of AdRec

There is plenty of scope for practical applications of an adaptive recommender system. Examples of these everyday applications might be seen in E-Commerce, for example, providing the best available recommendation algorithm to a Web-application developer automatically, based on a lightweight analysis of the available data. This would eliminate the need for a recommender systems expert to be part of the programming team for an E-Commerce application. A similar argument can be applied to areas such as Shopping, Stock Control, Tourist Trade, etc. In general, AdRec could be deployed anywhere that a conventional recommender

system is used, bringing the advantage of the most suitable and state of the art recommendation algorithms available.

An adaptive system enables application of the most suitable recommendation technique to an application with minimal analysis of the domain. The AdRec system provides a structured framework for assessing the performance of different recommendation techniques over a range of domains. The system has a scalable architecture to allow for new techniques and technologies to be added as they arrive.

Appendix A

Regression Function Calculation

In this Appendix, we show the calculations for the multiple variable regression functions used in the AdRec system. For the following calculations, we are only interested in the probabilities of the relative performance of each algorithm, so the scalar values by themselves are of no real significance. The following is a standard multiple variable linear regression function:

$$E\{Y\} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \tag{A.1}$$

where β_1 = Sparsity and β_2 = User-Item Ratio.

Let a = Rule-Based CF, b = Item-Based CF, and c = User-Based CF. We need to calculate a unique regression function for each algorithm, based on the experimental results in Table A.2 and Table A.1 We do this by inserting values from the tables into equation A.1 and solving the resulting system of simultaneous equations for β_1 and β_2.

We will begin by letting Y represent the Item-Based algorithm:

$$E\{IBCF\} = 63 + \beta_1(16.6) + \beta_2(94.25)\ldots(ptv) \tag{A.2}$$

Dataset	UBCF	IBCF	RBCF	Zero-r	Best Performing Algorithm
PTV	63	65	68	42	Rule-based
MovieLens	62	60	70	56	Rule-Based
Jester	63	63	60	44	User-Based
EachMovie	65	75	76	50	Rule-Based

Table A.1: Performance Results for each algorithm

Dataset	*Users as a % of Items*	*Sparsity (%)*	*Type*	*total possible ratings*
PTV	16.6	94.25	TV Programmes	2.3 Million
MovieLens	69.2	93.6	Movie Ratings	1.6 Million
Jester	900	46.41	Jokes Ratings	7.4 Million
EachMovie	52.9	97.6	Movie Ratings	118 Million
SmartRadio	11.1	99.98	Music Ratings	4 Million

Table A.2: Classification of Experimental Datasets.

$$E\{IBCF\} = 62 + \beta_1(69.2) + \beta_2(37.0) \ldots (movielens) \tag{A.3}$$

$$E\{IBCF\} = 63 + \beta_1(900) + \beta_2(46.41) \ldots (jester) \tag{A.4}$$

$$E\{IBCF\} = 65 + \beta_1(52.9) + \beta_2(97.6) \ldots (eachmovie) \tag{A.5}$$

These give the IBCF regression function:

$$E\{IBCF\} = 10.38\beta_1 + 2.75\beta_2 + 2.55 \tag{A.6}$$

Substituting in values for the SmartRadio dataset (where β_1 represents user-item ratio, and β_2 represents sparsity): $\beta_1 = 11.1$ and $\beta_2 = 99.98$

$$E\{IBCF\} = 10.38(11.1) + 2.75(99.98) + 255 = 64.016\% \tag{A.7}$$

Similarly for the User-Based Algorithm:

$$E\{UBCF\} = 65 + \beta_1(16.6) + \beta_2(94.25) \ldots (ptv) \tag{A.8}$$

$$E\{UBCF\} = 60 + \beta_1(69.2) + \beta_2(37.0) \ldots (movielens) \tag{A.9}$$

$$E\{UBCF\} = 63 + \beta_1(900) + \beta_2(46.41) \ldots (jester) \tag{A.10}$$

$$E\{UBCF\} = 75 + \beta_1(52.9) + \beta_2(97.6) \ldots (eachmovie) \tag{A.11}$$

Solving the simultaneous equations...

$$E\{UBCF\} = 263 + 10.4\beta_1 + 2.75\beta_2 \tag{A.12}$$

Substituting in the SmartRadio dataset values: $\beta_1 = 11.1$ and $\beta_2 = 99.98$

$$E\{UBCF\} = 262 + 10.387(11.1) + 2.752(99.98) = 65.2\% \tag{A.13}$$

and for the rule-based algorithm

$$E\{RBCF\} = 68 + \beta_1(16.6) + \beta_2(94.25) \ldots (ptv) \tag{A.14}$$

$$E\{RBCF\} = 70 + \beta_1(69.2) + \beta_2(37.0) \ldots (movielens) \quad \text{(A.15)}$$

$$E\{RBCF\} = 60 + \beta_1(900) + \beta_2(46.41) \ldots (jester) \quad \text{(A.16)}$$

$$E\{RBCF\} = 76 + \beta_1(52.9) + \beta_2(97.6) \ldots (eachmovie) \quad \text{(A.17)}$$

Solving the simultaneous equations:

$$E\{RBCF\} = 274 + 10.387\beta_1 + 2.752\beta_2 \quad \text{(A.18)}$$

substituting in the SmartRadio dataset values: $\beta_1 = 11.1$ and $\beta_2 = 99.98$:

$$E\{RBCF\} = 274 + 10.387(11.1) + 2.752(99.98) = 66.44\% \quad \text{(A.19)}$$

Finally for the Zero-r Algorithm:

$$E\{ZeroR\} = 42 + \beta_1(16.6) + \beta_2(94.25) \ldots (ptv) \quad \text{(A.20)}$$

$$E\{ZeroR\} = 56 + \beta_1(69.2) + \beta_2(37.0) \ldots (movielens) \quad \text{(A.21)}$$

$$E\{ZeroR\} = 44 + \beta_1(900) + \beta_2(46.41) \ldots (jester) \quad \text{(A.22)}$$

$$E\{ZeroR\} = 50 + \beta_1(52.9) + \beta_2(97.6) \ldots (eachmovie) \quad \text{(A.23)}$$

Solving the simultaneous equations:

$$E\{ZeroR\} = 274 + 10.387\beta_1 + 2.752\beta_2 \quad \text{(A.24)}$$

Substituting in the SmartRadio dataset values: $\beta_1 = 11.1$ and $\beta_2 = 99.98$

$$E\{ZeroR\} = 192 + 10.387(11.1) + 2.752(99.98) = 58.20\% \quad \text{(A.25)}$$

So our regression analysis over user-item ratio, sparsity and algorithm performance in 4 other datasets generates predicted scores for each algorithm on the SmartRadio dataset as follows:

1. RBCF = 66.44%
2. UBCF = 65.2%
3. IBCF = 64.016%
4. Zero-r = 58.20%

If we examine the actual predictive accuracy results for our algorithms on the SmartRadio dataset, we can see that although the regression function has overestimated the performance, it does predict *every* algorithm in the correct order. A possible explanation for this overestimate is that the importance of the sparsity metric may not have been given enough weight in the regression function. The SmartRadio dataset is *extremely* sparse at 99.98% sparsity, much more sparse than any of the other datasets. This is our main theory to explain the overestimates in our predictions.

The actual predictive accuracy scores on the SmartRadio dataset are:

1. RBCF = 57%
2. UBCF = 56%
3. IBCF = 55%
4. Zero-r = 53%

Appendix B

Code: Illustrating Parameters in the AdRec System

```
/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: UCD</p>
 * @author John O'Donovan
 * @version 1.0
 */
public class RunInstance{

  private int users;
  private int items;
  private String dbname;
  private int ttratio;
  private int knn;
  private int filtertype;
  private int arg7;
  private boolean ibcf;
  private boolean ubcf;
  private boolean cbr;

  /**
   * default constr. (test constr.)
   */
```

```
public RunInstance() {
 dbname = "Movielens";
  users = 10;
  items = 10;
  ttratio = 10;
  knn = 1;
  filtertype = 1;
  arg7 = 0;
  ibcf = false;
  ubcf = false;
  cbr = false;
}

/**
 * Explicit Constr.
 */
public RunInstance(String db, int u, int i,
           int ttr, int k, int f, int a7){     //explicit constr.
  dbname = db;
  users = u;
  items = i;
  ttratio = ttr;
  knn = k;
  filtertype = f;
  arg7 = a7;
}
public int getUsers(){
return users;
}
public void setUsers(int i){
users = i;
}
public int getItems(){
return items;
}
public void setItems(int i){
items = i;
}
```

```
public String getDBName(){
return dbname;
}
public void setDBName(String n){
dbname = n;
}
public void setTTRatio(int i){
ttratio = i;
}
public int getTTRatio(){
return ttratio;
}
public void setKNN(int k){
knn =k;
}
public int getKNN(){
return knn;
}
public int getFiltertype(){
return filtertype;
}
public void setFilterType(int i){
filtertype = i;
}
public boolean getIBCF(){
return ibcf;
}
public void setIBCF(boolean b){
ibcf = b;
}
 public boolean getUBCF(){
return ubcf;
}
public void setUBCF(boolean b){
ubcf = b;
}
 public boolean getCBR(){
return cbr;
```

```
}
public void setCBR(boolean b){
cbr = b;
}
}
```

Bibliography

[1] Fabio Abbattista, Marco Degemmis, Pasquale Lops, Giovanni Semeraro, and Fabio Zambetta. Improving the usability of an e-commerce web site through personalization. In *Proceedings of the Workshop on Recommendation and Personalisation in E-Commerce, Malaga, Spain*, pages 20–29, May 27–31 2002.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, September 12–15 1994.

[3] Syed S. Ali and Susan McRoy. Links: Java resource for artificial intelligence. *intelligence*, 11(2):15–16, 2000.

[4] Suhail Ansari, Ron Kohavi, Llew Mason, , and Zijian Zheng. Integrating e-commerce and data mining: Architecture and challenges. In *ICDM'01: The 2001 IEEE International Conference on Data Mining*, pages 27–34, 2001.

[5] Nicholas J. Belkin and Bruce B. Croft. Information filtering and information retrieval: two sides of the same coin? *Commun. ACM*, 35(12):29–38, December 1992.

[6] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning*, pages 46–54. Morgan Kaufmann, San Francisco, CA, 1998.

[7] Christian Borgelt and Rudolf Kruse. Induction of association rules: Apriori implementation. In *Proceedings of the 15th International Conference on Computational Statistics (Compstat 2002, Berlin, Germany)*, Heidelberg, Germany, 2002. Physika Verlag.

[8] Keith Bradley, Rachael Rafter, and Barry Smyth. Case-based user profiling for content personalisation. In *AH '00: Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 62–72. Springer-Verlag, 2000.

[9] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, San Francisco, July 24–26 1998. Morgan Kaufmann.

[10] R. Burke. Hybrid recommender systems: Survey and experiments. In *Proceedings of the User Modeling and User-Adapted Interaction Conference (UAI-00)*, pages 53–62, 2000.

[11] Sonny H. S. Chee. Rectree: A linear collaborative filtering algorithm. Master's thesis, Simon Fraser University, Toronto, Canada, Nov 2000.

[12] Sonny Han Seng Chee, Jiawei Han, and Ke Wang. Rectree: An efficient collaborative filtering method. In *DaWaK '01: Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery*, pages 141–151. Springer-Verlag, 2001.

[13] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper, 1999.

[14] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, and John Riedl. Is seeing believing?: How recommender system interfaces affect users' opinions. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 585–592. ACM Press, 2003.

[15] Paul Cotter and Barry Smyth. Personalisation technologies for the digital TV world. In *Proceedings of the European Conference on Artificial Intelligence, (ECAI 2000)*, pages 701–705, 2000.

[16] Paul Cotter and Barry Smyth. PTV: Intelligent personalised TV guides. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 957–964, Menlo Park, CA, 2000. AAAI Press.

[17] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew K. McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pages 509–516, Madison, US, 1998. AAAI Press.

[18] Chrysanthos Dellarocas. Building trust on-line: The design of reliable reputation reporting mechanisms for online trading communities. *eBusiness@MIT, July 2001*, 2001.

[19] D. Fisk. An application of social filtering to movie recommendation. In *Software Agents and Soft Computing, Lecture Notes in Computer Science*, pages 116–131, 1997.

[20] Andreas Geyer-Schulz, Michael Hahsler, and Maximillian Jahn. Educational and scientific recommender systems: Designing the information channels of the virtual university. *International Journal of Engineering Education*, 17(2):153–163, 2001.

[21] Anjua Gokhale. Improvements to collaborative filters algorithms. Master's thesis, Worchester Ploytechnic Institute, 2000.

[22] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[23] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[24] Andrew R. Golding and Dan Roth. A winnow-based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130, 1999.

[25] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 11th Conference on Innovative Applications of Artificial Intelligence*, pages 439–446, Menlo Park, Cal., July 18–22 1999. AAAI/MIT Press.

[26] Marco Grimaldi and Pádraig Cunningham. Experimenting with music taste prediction by user profiling. In *MIR '04: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 173–180. ACM Press, 2004.

[27] Markku Hakala, Juha Hautmki, Kai Koskimies, Jukka Paakki, Antti Viljamaa, and Jukka Viljamaa. Task-driven specialization support for object-oriented frameworks. Technical report, University of Tampere, Finland, 1997.

[28] Conor Hayes, Pádraig Cunningham, Patrick Clerkin, and Marco Grimaldi. Programme-driven music radio. In *Proceedings of the 15th European Conference on Artificial Intelligence*, 2002.

[29] Ralph Herbrich, Thore Graepel, and Klaus Obermayer. Regression models for ordinal data: A machine learning approach. In *Technical report, TU Berlin, TR-99/03.*, 1999.

[30] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of ACM CSCW'00 Conference on Computer-Supported Cooperative Work*, pages 241–250, 2000.

[31] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

[32] A. Jameson, J. Konstan, and J. Riedl. AItechniques for personalised recommendation. In *Anthony Jameson, Joseph A. Konstan and John Riedl. Tutorial SA1, Eighteenth International Conference on Artificial Intelligence, August 10th, 2003*, 2003.

[33] George Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA*, pages 247–254, 2001.

[34] Judy Kay. Lies, damned lies and stereotypes: Pragmatic approximations of users. In *Proceedings of the 4th International Conference on User Modeling*, pages 175–184, Bedford, MA, USA, August 1994. MITRE Corporation.

[35] Liadh Kelly. An intelligent navigational aid for the world wide web. Master's thesis, University College Dublin, Belfield, Dublin 4. Ireland, February 2002.

[36] Liadh Kelly and John Dunnion. Invaid: A personalised navigational aid system for the web. In *Proceedings of the 11th Irish Conference on Artificial Intelligence and Cognitive Science*, August 23–25 2000.

[37] Liadh Kelly and John Dunnion. Personalised web navigation using combination filtering. In *Proceedings of the 14th Irish Conference on Artificial Intelligence and Cognitive Science*, Dublin, September 2003.

[38] B. Krulwich. Lifestyle finder. *AI magazine*, 18:37–46, 1997.

[39] N. Kushmerick. Robustness analyses of instance-based collaborative recommendation. In H. Toivonen T. Elomaa, H. Mannila, editor, *Proceedings of the European Conference on Machine Learning, Helsinki, Finland.*, volume 2430, pages 232–244. Lecture Notes in Computer Science Springer-Verlag Heidelberg, 2002.

[40] Nick Kushmerick, Fergus Toolan, and James McKee. Towards zero-input personalization: Referrer-based page prediction. In C. Strapparava P. Brusilovsky, O. Stock, editor, *Adaptive Hypermedia and Adaptive Web-Based Systems International Conference*, pages 133–143, Trento, Italy, August 25–30 2000. Lecture Notes in Computer Science Springer-Verlag Heidelberg.

[41] Henry Lieberman. Letizia: An agent that assists web browsing. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[42] Henry Lieberman. Letizia: An agent that assists web browsing. In C. S. Mellish, editor, *Proceedings of 14th International Joint Conference on Artificial Intelligence*. AAAI Press, 1995.

[43] Greg Linden, Steve Hanks, and Neal Lesh. Interactive assessment of user preference models: The automated travel assistant. In *Proceedings, User Modeling '97*, 1997.

[44] Sean Luke, Lee Spector, David Rager, and James Handler. Ontology-based web agents. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 59–68, Marina del Rey, CA, USA, 1997. ACM Press.

[45] P. Melville, R. Mooney, and R. Nagarajan. Content-boosted collaborative filtering. In *In Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.

[46] Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, January 1997.

[47] Neter, J., Wasserman, W. and Kutner, M.H. *Applied Linear Statistical Models*. Irwin, Homewood, IL USA, 2nd edition, 1985.

[48] D. Oard and J. Kim. Implicit feedback for recommender systems. In *Proceedings of the AAAI Workshop on Recommender Systems*. AAAI Press, 1998.

[49] John O'Donovan and John Dunnion. A comparison of collaborative recommendation algorithms over diverse data. In *Proceedings of the National Conference on Artificial Intelligence and Cognitive Science (AICS), Ireland*, pages 101–104, September 17–19 2003.

[50] John O'Donovan and John Dunnion. Adaptive recommendation: Putting the best foot forward. In *Proceedings of 3rd International Symposium on Information and Communication Technologies (ISICT), Las Vegas, Nevada. USA*, pages 502–506, August 23–26 2004.

[51] John O'Donovan and John Dunnion. Evaluating information filtering techniques in an adaptive recommender system. In *Proceedings of 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Eindhoven, The Netherlands*, pages 312–315, June 16–18 2004.

[52] John O'Donovan and John Dunnion. A framework for evaluation of collaborative recommendation algorithms in an adaptive recommender system. In *Proceedings of the International Conference on Computational Linguistics (CICLing-04), Seoul, Korea*, pages 199–203. Springer-Verlag, February 15–21 2004.

[53] John O'Donovan and John Dunnion. Mheitifhoglaim do mholadh dhearadh innill. In John O'Donovan, editor, *Sruth Gaeilge sa 15ú Comhdháil ar an Intelacht Shaorga agus ar na hEolaíochtaí Cognaíochta (AICS 04) Caisleán an Bharraigh, Contae Mhaigh Eo, Éireann.*, pages 18–20, September 9–12 2004.

[54] John O'Donovan and Barry Smyth. Trust in recommender systems. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174. ACM Press, 2005.

[55] Michael P. O'Mahony, Neil Hurley, and Guenole C. M. Silvestre. An attack on collaborative filtering. In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 494–503. Springer-Verlag, 2002.

[56] Derry O'Sullivan, David C. Wilson, and Barry Smyth. Improving case-based recommendation: A collaborative filtering approach. In *Proceedings of the Sixth European Conference on Case Based Reasoning*, pages 278–284, 2002.

[57] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[58] Terence John Parr. Enforcing strict model-view separation in template engines. In *WWW '04: Proceedings of the 13th International Conference on World Wide Web*, pages 224–233. ACM Press, 2004.

[59] David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In C. Boutilier and M. Goldszmidt, editors, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 473–480, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[60] R. Rafter, K. Bradley, and B. Smyth. Passive profiling and collaborative recommendation. In *Proceedings of the 10th Irish Conference on Artificial Intelligence and Cognitive Science, Cork, Ireland*, 1999.

[61] Rachael Rafter, Keith Bradley, and Barry Smyth. Automated collaborative filtering applications for online recruitment services. *In proceedings of the International Conference on Adaptive Hypermedia Lecture Notes in Computer Science*, 1892:363–72, 2000.

[62] Adrian E. Raftery, David Madigan, and Jennifer A. Hoeting. Bayesian model averaging for linear regression models. *Journal of the American Statistical Association*, 92(437):179–191, 1997.

[63] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, pages 175–186, 1994.

[64] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[65] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *World Wide Web*, pages 285–295, 2001.

[66] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC-00)*, pages 158–167, N.Y., October 17–20 2000. ACM.

[67] Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*, Social Filtering, Social Influences, pages 345–354, 1998.

[68] Badrul M. Sarwar, Joseph A. Konstan, Jonathan L. Herlocker, Bradley N. Miller, and John Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Computer Supported Cooperative Work*, pages 345–354, 1998.

[69] J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in E-Commerce. In *Proceedings of the ACM Conference on Electronic Commerce (EC-99)*, pages 158–166, New York, November 3–5 1999. ACM Press.

[70] I. Schwab and W. Pohl. Learning user profiles from positive examples. In *Schwab, I. and Pohl, W. (1999). Learning User Profiles from Positive Examples. In Proceedings of the International Conference on Machine Learning and Applications, pp. 15-20. Chania, Greece*, 1999.

[71] Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1 of *Papers: Using the Information of Others*, pages 210–217, 1995.

[72] B. Smyth, D. Wilson, and D. O'Sullivan. Improving the quality of the personalised electronic programme guide. In *In Proceedings of the TV'02 the 2nd Workshop on Personalisation in Future TV, May 2002.*, pages 42–55, 2002.

[73] Barry Smyth and Paul Cotter. Surfing the digital wave. In *ICCBR '99: Proceedings of the Third International Conference on Case-Based Reasoning and Development*, pages 561–571, London, UK, 1999. Springer-Verlag.

[74] Barry Smyth and Paul Cotter. Surfing the digital wave: Generating personalised TV listings using collaborative, case-based recommendation. *Lecture Notes in Computer Science*, 1650:561–567, 1999.

[75] Barry Smyth and Paul Cotter. Personalized electronic program guides for digital TV. *The AI Magazine*, 22(2):89–95, 2000.

[76] M. Sollenborn and P. Funk. Category-based filtering in recommender systems for improved performance in dynamic domains. In *Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Malaga, Spain. May 2002*, pages 436–439, 2002.

[77] Frank Stowell and John Mingers. *Information Systems: An Emerging Discipline?* McGraw-Hill Berkshire, England, 1997.

[78] K. Swearingen and R. Sinha. Beyond algorithms: An HCI perspective on recommender systems. In *ACM SIGIR Workshop on Recommender Systems, New Orleans, Lousiana*, 2001.

[79] Davor Ubrani and Gail C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, pages 408–418. IEEE Computer Society, 2003.

[80] Perdita Stevens with Rob Pooley. *Using UML: software engineering with objects and components*. Object Technology Series. Addison-Wesley Longman, 1999. Updated edition for UML1.3: first published 1998 (as Pooley and Stevens).

[81] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. Technical Report TR651, URCS, 1997.

[82] Tong Zhang and Vijay S. Iyengar. Recommender systems using linear classifiers. *Journal of Machine Learning*, 2:313–334, 2002.