ON THE USE OF SEMANTIC FEEDBACK IN RECOMMENDER SYSTEMS

Matthew Garden

School of Computer Science
McGill University, Montréal

August 2004

A Thesis submitted to McGill University
in partial fulfilment of the requirements for the degree of
Master of Science

Abstract

This thesis presents a new approach to recommender systems. Previous recommender systems based on collaborative filtering typically solicit user feedback on domain items as overall ratings which are then recorded as numeric values. This paradigm limits the semantic richness of the user's interaction with the system and the depth to which the system can understand user preferences. We propose a new recommender system, Recommendz, which allows the user to comment not only about the overall quality of the item but also about the *quantity* and *quality* of features of the item. This allows the user to justify his or her ratings and allows the system to compare users not only with respect to overall preference, but also to compare the reasons behind those preferences.

We have developed an implmentation of our approach, and have collected extensive empirical data based on movie ratings. We demonstrate the effectiveness of our approach, and describe the details of the implementation.

Résumé

Cette thèse présente une nouvelle approche aux systèmes de recommender. Les systèmes de recommender précédents basés sur filtrer en collaboration sollicitent typiquement les réactions d'utilisateur sur les articles de domaine comme les classements généraux qui sont alors enregistré comme les valuers numériques. Ce paradigme limite la richesse sémantique de l'interaction de l'utilisateur avec le système et la profondeur à que le système peut comprendre les préférences d'utilisateur. Nous proposons un nouveau système de recommender, Recommendz, qui permet l'utilisateur de commenter pas seulement de la qualité générale de l'article mais aussi de la *quantité* et la *qualité* des caracteristiques de l'article. Ceci permet à l'utilisateur de justifier son ou ses classements et permet au système de comparer des utilisateurs pas seulement dans la référence à la préférence générale, mais aussi de comparer les raisons qui ont causé les préférences.

Nous avons développé une implementation de notre approche, et par il a recueilli la base des données empiriques étendues basées sur les classements de film. Nous démontrons l'efficacité de notre approche, et décrirons les détails de l'implémentation.

Acknowledgements

I owe a great deal of thanks to supervisor and advisor, Greg Dudek, for his direction, suggestions, criticism, openness to my ideas, and for implementing many interesting features for the website. I could not ask for a better supervisor.

Thanks to everyone in the lab, for encouragement and many enthusiastic ideas for the direction of this work, and for making the Mobile Robotics Laborartory a great place to do research in.

Thanks to my family for their encouragement and support, even when they had no idea what I was talking about. Thanks to my friends, scattered across Canada, who have always been there when I need them. Over all these years I really could not have accomplished so much without them.

Thanks to Alexandre Elias for proofreading and in implementing several of the features on the website, especially in making it more visually interesting.

I would also like to thank the National Science and Engineering Research Council of Canada for having provided me with the scholarship which has made it possible for me to pursue this research.

Glossary

This section is intended as a concise reference of acronyms and mathematical notation for the reader.

CF	Collaborative filtering.
MAE	Mean Absolute Error.
NMAE	Normalized Mean Absolute Error.
PCA	Principal Component Analysis.
POP	Prediction method in which the global mean overall rating for an item is taken as the predicted rating for all users for that item.
SFA	Sparse Factor Analysis.
SVD	Singular Value Decomposition.
F_u	The set of all features used by user u in creating ratings.
I_u	The set of all items rated by user u.
$I_u^{\tilde{r}}$	The set of items for which the system could compute predicted ratings for user u.
o_{ui}^f	The feature opinion rating of user u on feature f in item i.
$o_{\min}, o_{\max}$	The minimum and maximum values, respectively, of a feature opinion rating, that the user may enter.
q_{ui}^f	The feature quantity rating of user u on feature f in item i.
$q_{\min}, q_{\max}$	The minimum and maximum values, respectively, of a feature quantity rating, that the user may enter.
r_{ui}	The overall rating of item i by user u.
$\tilde{r}_{ui}$	The predicted overall rating of user u on item i.
$r_{\min}, r_{\max}$	The minimum and maximum overall rating, respectively, that the user may enter.

s_r, s_o, s_q	Three measures of the similarity between a pair of users, based on overall ratings, feature opinion ratings, and feature quantity ratings, respectively.
u, w	Users of the system.
U	The set of all users of the system.
$\omega_r, \omega_o, \omega_q$	The weights given to the overall, feature opinion, and feature quantity weights, respectively, in computing user similarity.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

A recommender system is a mechanism providing suggestions regarding items of interest based on knowledge of a user's tastes. Most recommender systems are web-based and base their suggestions on knowledge of a user's existing tastes in the domain of interest (for example, what movies they like and dislike).

This thesis presents a new approach to recommender systems which centres around richer interaction with the user. In most recommender systems, the model of a user is generated from user feedback specified as a numeric rating reflecting the user's overall opinion of an item; however, this affords the user no oppourtunity to provide feedback on the basis and peculiarities of their judgment on the items. In our system, we allow the user to describe his or her opinion of the presence and quality of features that characterize the item, and so our preference data is more nuanced and has a different structure from that of other recommender systems. As a result, our approach to recommendation must be as unique as the preference data we collect.

This work is exemplified in a system we have developed called *Recommendz*. This recommender system is web-based and contains a database of items from multiple domains of interest (such as movies). Visitors to the site provide feedback on particular items and are then provided with predicted ratings on items they have not yet rated.

1. Motivation

The Internet has grown greatly over the past decade, and this growth has meant a great increase in the availability of information. This information takes on many different forms,

and any item may not be equally interesting to all users. In part, this is due to differing user interests in the content of the items. If we know the sorts of content in which a user is interested, we can then filter items in order to eliminate those the user will find irrelevant. This process is referred to as *content-based filtering*.

It is important to recognize that people's preferences are intrinsically subjective, and that items containing similar content can vary in (perceived) quality. To address this, *collaborative filtering* systems were developed in which users provide subjective ratings of items, and this preference data is then taken into account to try to understand the user's tastes and to make appropriate recommendations.

In addition, some content may not lend itself to content-based filtering as readily. For instance, we may not have a method for automatically extracting useful content information from the items (*e.g.* images). In such a case it will not be possible to use a traditional content-based filtering scheme, where content is assumed to be objectively known. Good examples of items which work well with content-based filtering include documents such as web pages in which content, in the form of text, can be extracted easily and automatically. We wish our system to be generic enough to work with either sort of item, and so we assume no preexisting or easily-extractable content information. Instead, the content information of an item is specified by the users as they provide feedback about it.

A benefit of collaborative filtering are so-called *serendipitous recommendations* [**GSK+99**], recommendations which may have little to do with the content the user is accustomed to, but which nonetheless he or she is likely to enjoy based on the experience of otherwise similar users. One alternative to collaborative filtering which may offer this benefit to a user are traditional "expert" reviews, for instance Roger Ebert [1] for movies or Allmusic Guide [2] for music. In this system each review is written by a single expert, who may have preferences wildly divergent from those of the user, resulting in disagreement regarding the reviewed item. In addition, the experience of the reviewer may be quite different from the user in question, potentially leading to serendipitous recommendations. Meta-review sites also exist, which combine the reviews of many experts into a single review, essentially based on a

[1] http://www.suntimes.com/index/ebert.html
[2] http://www.allmusic.com

consensus of experts rather than the opinion of one [3]. Still, any given user may typically disagree with consensus, especially in a domain where preference is extremely subjective. In fact such a global consensus-based recommendation is used later as a baseline against which to compare the performance of personalized recommender systems, and we show we can substantially outperform it (see Chapter 5, Section 3).

Content-based and collaborative filtering methods are by no means mutually exclusive. In fact, some combination of the two approaches may be necessary. In a content-based filtering system the user can still be overwhelmed by too many matches, and those recommended items may not match the user's tastes. In a collaborative filtering system the user may prefer familiar types of content to serendipitous recommendations. There has been substantial research into ways of combining collaborative and content-based filtering (see Chapter 2, Section 6). At the most basic level we can apply the two sorts of filters sequentially, either using content-based filtering to find items with content in which the user might be interested, and then using collaborative filtering to find which of these items the user might prefer; or using collaborative filtering to find items the user will prefer and then filtering out items in which the user is most likely not interested. One could also calculate the content-based and collaborative filtering predictions for a given item and then combine the two in a weighted average. Our approach is not simply sequential and is more complicated than a simple weighted average. We introduce a novel format for providing feedback on items which combines both content and preference information, and present a method for providing recommendations from this enriched data set.

In a collaborative filtering system, it is necessary to record user preferences in some form. There are different approaches to this (see the discussion in Chapter 2, Section 2) but the most common approach [**RIS$^+$94, HKB$^+$99, MLR03**] is to have the user indicate overall attitude toward the item as a number on some scale, for instance from 1 to 10, with a lower value indicating less preference for the item. The result is that the system is aware of user preferences toward items, but the reasons behind those preferences are hidden. Several authors have developed techniques to infer these reasons behind preferences in order to make better recommendations [**Hof01, Can02**], but these approaches do not explicitly *identify* the reasons being inferred, which limits their usefulness in explaining recommendations

[3]Examples of such systems include "RottenTomatoes" at http://www.rottentomatoes.com, and "Meta-Critic" at http://www.metacritic.com

to the user. Explaining recommendations has been identified as an important aspect of recommender systems [**HKR00**].

It is possible that two users may assign a similar rating to a given item but have different or even contradictory reasons for choosing that particular rating value. If we lack information about the reasons behind the users' preferences then we will not know whether it is appropriate to consider such a pair of users as similar or not. Burke [**Bur00**] created a recommender system which used *semantic ratings* (short phrases describing some characteristic of the item), but the set of semantic ratings in this system is static and predetermined by manual knowledge-engineering. In contrast, *all* users in our system are free to add new features to describe content if none of the features suggested are felt to be adequate. There are no restrictions on the character of features, and in practice we have found that the features range from high level review-like descriptions, to very low-level descriptions of minor details of the item (some examples are "directed beautifully", "comedy", and"dark"). In Burke's system overall opinion is inferred from user behaviour, whereas in Recommendz users explicitly provide overall feedback on items. In addition, our feature feedback includes both numeric and textual components, allowing the system to reason with the feature mathematically while still presenting easily-understandable information to the user. Also, as described above, in a system which only collects overall feedback regarding items, the justification behind any given opinion must be inferred, typically in only a rudimentary form. Preference data is regarded as a noisy data space [$\mathbf{YXT^{+}02}$], so we believe that actually asking the user about the reasoning behind his or her preferences can sometimes be preferable to trying to infer it from noisy data.

By combining preference and content information our system gains a greater insight into the underlying reasons behind user preferences. We allow users to add new features to the system to describe items as required, rather than manually knowledge-engineering a limited set of features. By doing so, we ensure that users can describe items in whatever terms they feel are appropriate, meaning that the content information will not only be flexible and extensible, but also rich and appropriate as well. It is also worth noting that in some domains it may be harder to manually engineer an appropriate and effective set of features than in others, either because it is hard to predict which features users will find useful, which features will be relevant to new items introduced into the system, and which

new features (*e.g.* specialized jargon) will be developed for describing the items in question after the system's launch. In our solution these problems do not arise: any sort of item which people can discuss can be easily added.

We also suggest that our feedback system provides a more social and personally gratifying method of reviewing items. In our experience recommendations, for instance at the social level, are commonly made with both an overall opinion of the item as well as opinions about various aspects of the content of the item. This both provides more information and justifies the overall opinion.

This approach certainly requires an increased amount of interaction with the system on the part of the user; however, we believe that this requirement is outweighed by all of the benefits outlined above. This belief is mirrored in positive empirical results, both in terms of the accuracy of recommendations and user response (see Chapter 5).

One standard approach to collaborative filtering is to use nearest neighborhood interpolation to make predictions [**SM95, RIS$^+$94**]. In this approach, to make predictions for a given user the system first computes the user's similarity to all other users. "Similarity" is some measure of the degree of correspondence between the preferences the users have expressed through their ratings. Given this measure, a neighbourhood of the most similar users can be computed, and well-liked items within the neighborhood would be recommended to the user. Essentially this approach recommends the preferred items of users who have appeared to be similar to the target user in the past.

As personalized recommender systems become more and more common, on the Internet and mobile devices, it will be important to explore alternate methods of user feedback for human-computer interaction purposes, increased explanatory power for the system, and an enriched understanding of user preferences, all of which are important aspects of a recommender system.

2. Applications

Many websites have begun soliciting user feedback on various items. News sites ask for user feedback on articles and images [**Yah**], for instance. Other websites collect expert reviews and use them to calculate consensus opinions [**rot, met**]. However, individual users often disagree with a population average and have more in common with smaller groups

of like-minded users. As a result, personalized recommender systems have been applied to many item domains (*e.g.* movies, music, books, websites, research documents) and on many websites, some of which are informative [**GSHJ01, GRGP01, RIS+94, Dud**] and some of which are specifically based on advertising and e-commerce [**ama, SKKR00b, SKR01**].

3. Thesis Outline

The following is an outline of the remainder of this thesis:

Chapter 2 provides an overview of relevant research in the field of recommender systems, with particular attention to methods of feedback, collaborative filtering systems, the combination of content-based and collaborative filtering techniques in hybrid systems, and the problem of preference elicitation.

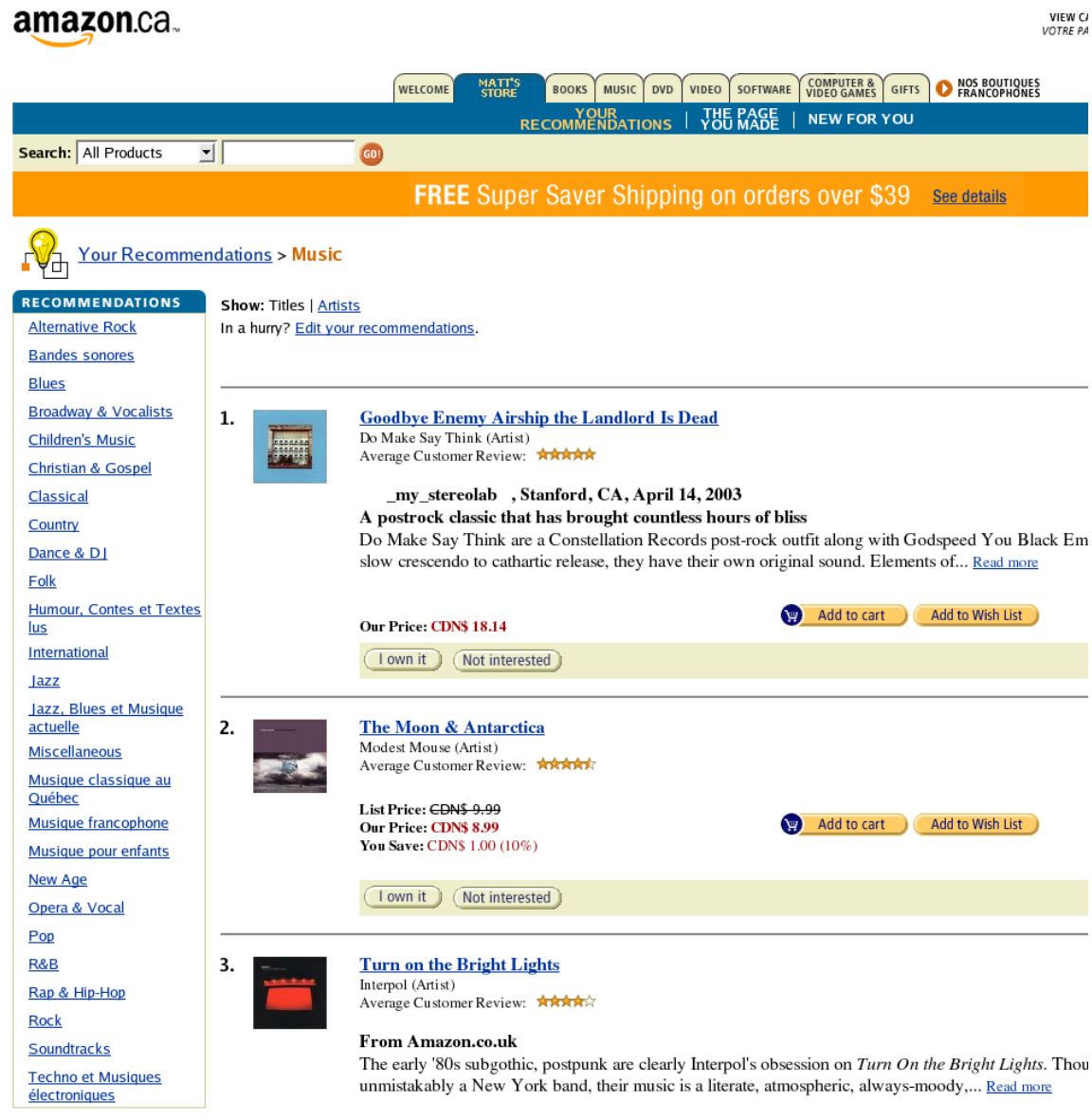
Chapter 3 discusses the approach to recommender systems we have taken in this research. Our approach includes item recommendation through a hybrid of collaborative filtering and content-based filtering, and intelligent preference elicitation to encourage more useful feedback from users.

Chapter 4 describes the implementation of our approach as the Recommendz [4] recommender system. We discuss the structure of the software infrastructure, the user interface, and the design decisions involved.

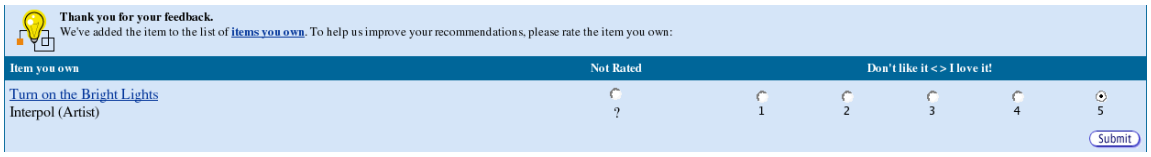
Chapter 5 discusses the experimental results we have obtained for the performance of our system with respect to predictive accuracy. These results are compared with those of several other approaches. This chapter also discusses the apparent user reaction to our system, as demonstrated in usage statistics as well as explicit user feedback on the system.

Chapter 6 contains a discussion of the results observed, presents conclusions, and discusses directions for future work.

[4] http://www.recommendz.com



(a) Recommendations calculated by amazon.com based on purchases and explicit feedback on items.



(b) The explicit feedback interface on amazon.com

FIGURE 1.1. Websites such as amazon.com have incorporated recommender systems into the e-commerce experience.

CHAPTER 2

Background

1. From manual to automated Collaborative Filtering

Attempts to gain an understanding of a subject's personality by extracting information from feedback has been a topic of interest in psychology research for some time. For instance the Minnesota Multiphasic Personality Inventory (MMPI) was first developed in 1940 as a standardized questionnaire intended to implicitly classify the subject in several personality dimensions [**HMB$^+$89**]. The goal of this system is not to make recommendations to a user but to understand the user's psychological makeup. MMPI involves a very large number of detailed psychological questions but is popular despite its intrusiveness.

The earliest collaborative filtering (CF) systems were primarily tools to facilitate social information filtering among the users, and were not automated.

In the *Tapestry* system [**GNOT92**], users annotate documents, creating a database of annotated documents. Users of the system can then query this database by making searches using the annotation values, allowing them to search not only with respect to the content of the documents, but also with respect to user comments regarding the documents. This is a manual form of collaborative filtering.

Maltz and Ehrlich [**ME95**] developed a similar collaborative filtering system which facilitated and supported pre-existing recommending behaviour. In their system, which was a website recommender, a recommendation is made by sending a *pointer* to a group of users, or a database (a repository for recommendations). A pointer consists of a hyperlink to the resource being recommended, an annotation containing contextual information about

the resource, and, optionally, any comments on the resource that the recommender feels are relevant. This system is primarily a formalized method for sending hyperlinks to other people, and is, like *Tapestry*, a manual collaborative filtering system.

As researchers recognized the potential in collaborative filtering, and the need for recommender systems in general, automated collaborative filtering systems began to be developed in which users provided feedback that would be analyzed and used to compute predicted ratings in order to automatically provide the user with recommendations. Good early examples of such systems include *GroupLens* [**RIS+94**], *PHOAKS* [**THA+97**], and *Ringo* [**SM95**].

Eventually the term *recommender system* became popular for describing systems which provide users with items predicted to be of interest, whether the prediction was made through collaborative filtering, traditional information retrieval techniques, or some combination of these and other approaches [**RV97**].

2. Feedback

User feedback is crucial to any recommender system. A name has been given to the lack of user ratings in a recommender system: The *Cold Start* problem, or the *Early Rater* problem. The Cold Start problem is the state of a new recommender system which has no rating information. At this state it is impossible to perform traditional collaborative filtering to provide recommendations. A standard way of alleviating this problem is to incorporate any known information about the content of items into the recommendation process [**CGM+99, SPUP02**] so that items can be matched based on user interest. Of course such an approach assumes that some sort of content information is available for the items involved. The "Early Rater" problem refers to being a user in a recommender system suffering from Cold Start. For such a user there are few items which can be recommended, if any at all, and the accuracy of these recommendations may be impaired due to lack of information. These problems also arise in relation to items which have recently been added to the database and thus have received no feedback.

Even systems which do not suffer from the Cold Start problem must be able to make accurate predictions based on sparse preference data. Recommender systems are often used in systems with large item sets (*e.g.* in e-commerce or in a domain such as movies) where

each user will only ever be able to provide feedback on a small number of items. As a result, there is a great deal of sparsity in the ratings data, and a recommender system must be able to function accurately despite this sparsity.

Researchers have attempted to address this sparsity issue in a few ways. One approach is to use a statistical measure which is designed to process sparse data [**BHK98, HKB+99**]. Pearson correlation (see Chapter 3, Section 2.1.1) is one example of such a measure. Another approach is to fill in the gaps in a each user's ratings and then make similarity comparisons between these dense versions of the ratings. There are many ways the gaps can be filled in, including item mean ratings, user mean ratings, or a more intelligently personalized value. It has been found that this approach does not perform very well without a large degree of manual intervention [**BHC98**]. Canny argues that filling in gaps in the data and then making predictions based on the resulting data is incorrect because the known ratings data induce a probability distribution over possible ratings, not a single value [**Can02**]. Preference data is highly dimensional, typically with one dimension per possible rating, *i.e.* one for each combination of user and item. Several methods can be used to compress such highly-dimensional, sparse data into a lower-dimensional, dense space, in which similarity between users or items can then be calculated [**GRGP01, Can02, SKKR00a**]. Section 5 contains a survey of several techniques related to dimensionality reduction.

One way in which we can classify recommender systems is with respect to how the user provides feedback regarding their preferences for the set of items being recommended. The mechanism by which the user interacts with the system has an impact on the character of the data which is available to the system in making recommendations. For instance, in e-commerce applications, user interaction with the system may consist of buying items, and as a result the system may only know whether a given item was purchased (preferred) or not. In this case the system will have to work with binary preference data.

In general, a system can collect preference data through an *implicit* mechanism such as monitoring user behaviour, or through an *explicit* mechanism where the user is prompted for a reaction to items in the database. Although most recommendation systems represent feedback as numeric values corresponding to the strength of preference or degree of relevance, some systems [**Bur00**] have used "semantic" feedback which describes content information or reflects human judgments on the quality or character of the rated item.

2.1. Implicit feedback. In a system using an implicit feedback scheme, the actions a user takes are monitored and preferences are inferred from these monitored actions. The inferred preferences are then used to make predictions about future preferences.

User preferences can be inferred from many behaviours. Which behaviours are relevant typically depends on the context of the recommender system. In a recommender system for documents or web pages, a system can monitor navigation history, time spent reading documents, whether or not a document was bookmarked or saved, and so on [**CLWB01, KOR01, Nic97**]. A music program might monitor the frequency with which certain songs, artists, or albums, are listened to [**Aud**]. Some systems combine both explicit and implicit feedback [**HC00**].

In some systems, different behaviours are assigned different values reflecting the relative importance (for instance clicking a link to view a webpage indicates interest but less than bookmarking that page), and others use binary values (for instance in e-commerce, where all the information the system may have is whether a user has purchased an item or not).

2.2. Explicit feedback. In a system using an explicit feedback scheme, the user will be presented with an item and prompted to provide an opinion on that item. This opinion can take any of a number of forms. A typical approach is having the user specify an overall opinion of the item as a rating on some numeric scale, for instance in the range $[1, 5]$ where a low score indicates distaste and a high score indicates preference. Some systems use a binary feedback scheme (e.g. "like"/"dislike" or "relevant"/"irrelevant") [**CMZF00, BP99**], others use rating scales with more or fewer choices [**GRGP01, HKR00**], and some systems use a rating scale which is presented in such a way as to (appear to) provide a continuous range of choice rather than just a few distinct options [**GRGP01**].

A variation on this approach is in a content-filtering based recommender system where the user would not provide feedback directly on items, but rather on topics or features of items, and recommendation is achieved by filtering the database of items according to the likes and dislikes expressed by the user through the filter.

2.3. Semantic feedback. In many systems, user feedback is provided as a numeric rating on a scale such as 1 to 10, or a binary value indicating affinity or dislike. We might describe such a rating system as *numeric* or *abstract*, or *holistic* since it aggregates all

properties of an item and reduces them to a single number indicating an attitude. In our system we combine such ratings with *semantic* ratings [**Bur00**], which are in part text annotations of features of the items.

The document annotation method used by the *Tapestry* system [**GNOT92**] is another good example of semantic feedback; however, while in *Tapestry* this semantic feedback is only used in manual queries, in Recommendz semantic feedback is incorporated into an automatic recommender system. While users are able to manually search items according to semantic feedback, the system automatically suggests items whenever the user logs in.

In Burke's *Entree* restaurant recommender system, knowledge engineering is used to provide a set of semantic rather than numeric rating options [**Bur00, Bur02**]. The user begins with some restaurant, and then can select from a list of attributes which indicate what is desired in a restaurant (e.g. "nicer", "less expensive", "livelier", "quieter", etc.). Choosing an attribute in order to browse to a new restaurant in effect enters a rating for that user and restaurant: for instance browsing away from a particular restaurant by selecting "less expensive" indicates to the system that you think that restaurant is overly expensive. On the other hand, if a user stops browsing at a restaurant, then that restaurant is assigned a positive rating under the questionable assumption that the user has found what he or she was looking for. User similarity is then based on not only on similarity of preference for items, but also on the reasons behind those preferences, allowing the system to find truly like-minded users. For instance if two users dislike the same restaurant but one dislikes it because it is too lively and the other because it is too quiet, there is little reason to consider the users truly similar. On the other hand if two users "browse away" from a restaurant because they feel it is too expensive the system can become more confident that the users are looking for the same thing in a restaurant. The approach taken in *Entree* was compared to several other approaches and found to be a very good predictor of restaurant preference.

In a variation on this approach employed by the RACOFI music recommender system, Anderson *et al.* defined five dimensions of music feedback (*impression*, *lyrics*, *music*, *originality*, and *production*) [**ABB$^+$03**]. To provide a rating for a given item, the user must provide a rating in each of these categories. Of course in this system, the feedback dimensions must be determined via manual knowledge-engineering, and are fixed once decided upon.

The *CoFIND* [**Dro99**] system also uses a semantic rating system. In this system, the user provides feedback on the "qualities" of the item and qualities are suggested for use if they have been used repeatedly. Users are able to add new qualities to the system. The emphasis in CoFIND is in organizing resources to aid learning as opposed to predicting preferences.

3. Preference elicitation

The problem of how to efficiently prompt the user to provide useful feedback, known as *preference elicitation*, is important to recommender systems. It is worth noting, as Boutilier points out [**Bou02**], that preference elicitation is important in a number of areas of research, not just recommender systems. In the context of a recommender system, the issue is to identify those items which will be useful to the system in predicting preferences, so that the user can be guided to provide as much information as possible with as little burden as possible. For instance, we may find that some item or group of items is a particularly good predictor of preferences on other items (*e.g.* maybe users who enjoyed a romance film are typically much less likely to enjoy slasher horror). In this case, having the user provide feedback upon a romance film will be useful because based on that feedback we can confidently infer preferences toward another group of items. On the other hand there may be items which will not be so useful, and it may even be the case that there are no items which would significantly increase our understanding of the user's preferences.

We wish to consider all possible queries which could be asked of the user to elicit additional preference information, and actually select and ask only those which provide the most useful information. Research which considers preference elicitation stresses the desirability of providing an online and interactive experience for users of recommender systems [**BZM03, CSP03**].

Carenini *et al.* identify preference elicitation as an underdeveloped aspect of recommender system research, and argue that research should focus both on improving the accuracy of recommender algorithms as well as devising methods which elicit as many ratings as possible from the users [**CSP03**]. These researchers also argue that the focus on preference elicitation as a technique relating to new users should be altered to create recommender systems in which intelligent preference elicitation plays an integral role, not just at user

registration time but throughout the user's interaction with the system, so that we can continually strive to learn as much as possible about the user, leading to better recommendations not only for that user but for others. The authors refer to such systems as adhering to a *Conversational and Collaborative* model. The authors identify four situations in which the system should prompt the user for more ratings:

(i) The users asks for a rating of an item but the system doesn't have enough information to provide a confident prediction for that user and item.

(ii) The system has a small amount of information on the user, and the prediction on a requested item is average. More information may provide a better recommendation.

(iii) The user is puzzled or surprised by a recommendation. In this case more information might help the system better understand the user's preferences and not make such surprising recommendations in the future.

(iv) Another user may have asked for a recommendation on an item which the system knows little about. By asking other users about this item, recommendations for the community as a whole will be improved.

The authors present several measures of the usefulness of a given item in understanding the preferences of a user. These methods include popularity, entropy, a combination of entropy and popularity, items similar to items the user has already rated (*item-item personalized*). These measures were compared against a random item selector. The combination of popularity and entropy was found to provide the best improvement in predictive accuracy without requiring a large user effort. The authors also refer to research which observed that users are willing to provide more feedback to the system in order to receive more accurate predictions [**SS01**].

Boutilier has formulated the preference elicitation problem as a *partially observable Markov decision process* (POMDP) [**Bou02**]. In this formulation, the recommendation process is seen as a decision problem in which the recommendation problem is broken into a series of steps; at each step, the system can either query the user for more preference information, or provide a recommendation. The gathering of information through such queries provides a greater understanding of the user's preferences (referred to as the user's *utility function* in POMDP terminology [**Bou02**]). Queries have an associated cost which

represents the fact that providing feedback requires some effort. This approach allows the system to balance the cost of elicitation with the gain in useful information achieved through the queries. In this POMDP formulation, the set of system states is the set of possible user attitudes and the POMDP's states are densities over the set of system states. The system can make a decision to elicit a query from the user, or to make a prediction of the user's underlying attitude towards items (these predictions are terminal). Rewards are given by the expected utility of the prediction made, and the observations of the system are the probabilities of the user giving a particular response to the given query.

In [**BZM03**], Boutilier *et al.* discuss an approach to preference elicitation based on the *expected value of information* (EVOI) of new user ratings. This particular measure fits well within systems that learn an explicit probabilistic model of the domain (such as [**PHLG00, CG99**]). The details presented in this research can be applied to any such system, and is demonstrated using the *multiple-cause vector quantization* (MCVQ) model [**RZ02**] and with a Naive Bayes classifier. The essential idea in this approach is to compute an approximation of the (myopic) EVOI for each possible query which could be asked of the user, and to choose the query with the maximal EVOI (or to make a prediction if the maximal EVOI is below some threshold or the system feels that too many queries have been asked and a prediction should be made). This process is quite computationally intensive for each user, since a large number of costly calculations must be made. Thus, the authors present an approach for pruning potential query items, and for pre-computing sets of prototype queries. The authors demonstrate that their approach leads to an improvement in both MCVQ and Naive Bayes performance as compared to entropy-based and random measures for preference elicitation. The improvement in performance is most dramatic when only a few user ratings have been observed, indicating that if a user should only provide a small amount of information, a well-designed preference elicitation scheme can be very helpful.

Another group of researchers [**DLL03**] has performed a theoretical analysis of the computational properties of a general collaborative filtering and preference elicitation problem. In this work, the problem is defined as using queries to elicit preference information from the user in order to determine the expert which is closest to the user in terms of preferences. The authors outline a polynomial-time approximation algorithm to solve this problem in

a bounded number of queries, and show that this upper bound on the number of queries cannot be beaten by a polynomial-time algorithm unless all problems in NP have $n^{O(\log\log n)}$ time algorithms.

4. Memory and model-based systems

Recommender systems can be categorized according to how they handle the ratings data which has been collected, into *memory-* and *model-based* systems [**BHK98**], which use the entire data set or a model of the data set to make recommendations, respectively. Lemire suggests a third category of recommender systems, the *learning-free* systems, which do not employ a model and do not compare users in order to make recommendations [**Lem04**].

Some systems, known as "memory-based", use the entire data set in order to compute a prediction. A typical memory-based algorithm would be one which used an inter-user similarity measure to find users who are similar to each other, and to then draw preferred items from that group as recommendations [**RIS$^+$94**]. This classical CF approach has been shown to provide a good level of accuracy and is easy to maintain [**BHK98, PHLG00**]. Another advantage is that memory-based systems always have a clear interpretation: each user is being matched to like-minded users [**YXS$^+$02**]. Lemire characterizes memory-based systems as those in which updating the ratings database requires constant time (since no model needs to be updated), but recommendation requires time proportional to the number of users (since users must be compared to make predictions) [**Lem04**]. As a result, with extremely large data sets, memory-based methods can be computationally expensive. Yu *et al.* have developed methods to identify "relevant" and "irrelevant" instances in the ratings data set, allowing the size of the data set to be reduced, increasing efficiency without degrading accuracy [**YXS$^+$02**]. In related work Yu *et al.* developed instance selection algorithms for memory-based recommendation which actually improved accuracy as well as computational speed by reducing noise and overfitting [**YXT$^+$02**]. Other authors have also attempted to improve the performance of memory-based algorithms, including Chee *et al.*, who developed a tree structure tailored specifically for recommendation algorithms [**CHW01**].

"Model-based" recommender systems develop a model of the rating data which is collected and then make predictions by manipulating the model rather than the raw rating data. The model is typically created in an offline phase, allowing online predictions to

be made quickly. For example model-based systems may use a dimensionality reduction algorithm such as Singular Value Decomposition (SVD) or the related Principal Component Analysis (PCA) [**GRGP01, SKKR00a, Can02**], a probabilistic model such as a Bayesian network [**BHK98, CG99**], clustering [**GRGP01, UF98**], or latent semantic models [**Hof01, Hof04**]. Some of the models used by recommender systems have meaningful semantic interpretations, which can help users understand how recommendations are generated [**PHLG00**]. Lemire characterizes model-based systems as those in which recommendation requires periodic updates of a potentially complicated model. These updates require linear or worse time in the number of users in the system but recommendations are based on a query of the model and likely fast; however, Yu *et al.* argue that the learning phase of model-based approaches can become prohibitively long for large data sets [**YXS+02**].

The *Personality Diagnosis* system combines some of the benefits of both memory and model-based methods [**PHLG00**]. In Personality Diagnosis, a set of "personality types" (stereotypes in terms of ratings on the item set) are determined, and users are matched to the nearest personality type. The system uses all data without a model-building phase, but at the same time it provides a model with a clear semantic interpretation.

5. Dimensionality reduction

Several researchers have investigated the application of dimensionality reduction methods to recommender systems. These approaches all work by first creating a model of the rating data by reducing the dimensionality of the rating data space. These methods are often created to reduce the problems caused by sparsity in the ratings data, scalability problems that can exist in naïve memory-based recommender systems, and to find latent connections between items [**SKKR00a**].

Singular Value Decomposition (SVD) results in data which is less noisy and also captures latent associations between items (Berry *et al.* [**BDO95**] quoted in [**SKKR00a**]). As discussed in Section 7, SVD has also been used in conjunction with supervised learning techniques [**BP98**].

In the *Eigentaste* method, which has been implemented as the *Jester* joke recommendation system by Goldberg *et al.* [**GRGP01**], Principal Component Analysis (PCA) is used

to reduce the rating space to two dimensions. All users are required to rate a *gauge set* of 10 items, and are then projected into the lower-dimensional rating space based on this gauge set. Within the lower-dimensional space, groups of similar users of the system are divided into clusters, and recommendations are computed as the preferred jokes among the users of each cluster. In terms of accuracy, this method was shown to perform very favourably compared to several other algorithms. Prediction with the algorithm is constant-time because the model can be maintained in an offline phase: each user is projected into the low-dimensional rating space based on the gauge set, and recommendations are then looked up from the best cluster.

Sarwar *et al.* have also applied dimensionality reduction to recommender systems [**SKKR00a**]. These researchers actually first addressed the sparsity problem in earlier work by using semi-intelligent filtering agents to make predictions in order to fill in some of the sparsity in the ratings set [**SKB+98**]. This approach uses SVD to do two things: capture the latent relationships between users and products, and then use that to directly predict for a given item; and to compute a low-dimensional version of the customer-product space and use this to compute neighbourhoods and within these neighbourhoods produce *Top-N* recommendation lists. Note that SVD assumes complete data, meaning that any missing values (of which there are many) must be filled in with some sort of default values. To fill in the gaps, Sarwar *et al.* tried using average ratings per customer and average ratings per item (and found product average to work better). Ratings are normalized before computing the SVD as well. Normalization by subtracting user average ratings was found to work better than conversion to z-scores.

In Canny's *'Mender* system [**Can02**], dimensionality reduction is performed with an SVD-related technique called *sparse factor analysis*. With this approach Canny avoids the problem encountered above by Sarwar *et al.* of having to fill in the sparse ratings data before being able to analyze it to build a model. In sparse factor analysis missing elements are effectively ignored rather than replaced with default values. The author argues that this method is more appropriate than making guesses to reduce sparsity because it does not introduce additional uncertainty. In the resulting lower-dimensional model, items and users are characterized in terms of hidden factors which depend on content, and preference for these factors, respectively. Predicted ratings can then be quickly calculated from the model.

Very good results were reported with this method on several data sets, in comparison with other methods.

Sarwar *et al.* have investigated collaborative filtering algorithms which rely on computing similarities between items ("item-item similarities") rather than similarities between users ("user-user similarities") in order to make recommendations [**SKKR01**]. They argue that the bottleneck in many collaborative filtering algorithms is the search for similar users in a large user population. In the approach presented in this work, the collaborative filtering algorithm finds items which are similar to items the user in question has enjoyed in the past. Unlike in content-based systems, the similarities between items are still computed based on preference data rather than on content information, so the benefit of serendipitous recommendations is not diminished. The authors show that these item-based approaches lead to slightly improved quality of predictions over a user-user similarity-based k-nearest-neighbour scheme. More importantly, a simple model can be adopted which allows pre-computation of item similarities, so that the system can avoid the bottleneck of computing inter-user similarity. The authors assume that item neighbourhoods are "fairly static", which means that item similarities can be pre-computed and subsets of the most similar items can be stored as a simple model.

Since the feedback in recommender systems is intrinsically subjective, it is not only possible but probable for different users to use different rating values to indicate similar degrees of preference. This effect manifests itself in differing biases in rating values and in the amplitude of ratings (*i.e.* the range of ratings used) between users. A recommender system must translate ratings from one user to another to make accurate predictions and therefore it is common to normalize ratings according to user biases. Lemire presents a detailed analysis of this phenomenon [**Lem04**] and develops scale- and translation-invariant versions of several learning-free, memory-, and model-based recommendation algorithms, demonstrating that the invariant versions outperform the standard versions.

Clustering algorithms have been applied to recommender systems. The users of the system are divided into groups of similar users based on preference data, and predictions are made within these groups. Ungar and Foster present a statistical model of collaborative filtering and compare several different methods for clustering users [**UF98**]. Goldberg *et al.* cluster users after reducing the dimensionality of the ratings space using PCA [**GRGP01**].

O'Connor and Herlocker evaluated several algorithms for clustering items based on similarity in ratings data [**HO99**]. The suggestion is that items which receive similar ratings from the same users are similar in some sense, and that only items within a given class are relevant to making predictions within that class. To achieve a more scalable system, the authors suggest that predictions for a given class of item can still be effectively performed using a single subset of items.

6. Hybrid recommendation

It has been noted in Information Retrieval research that a combination of the scores from various retrieval agents can outperform the score of any of the individual agents (Vogt *et al.* [**VCBB96**] quoted in [**CGM$^+$99**]).

The *Fab* system by Balabanovíc and Shoham [**BS97, Bal97**] is a hybrid content and collaborative filtering-based recommendation system for web sites. Users provide feedback on sites on a scale which is presented as terms such as "Neutral" and "Terrible" but which correspond to, and within the machinery of the recommendation algorithm are treated as, numeric values. In addition to this feedback, the system analyzes the words occurring in the web pages and uses this information combined with the user's feedback to build profiles both of the web pages and of user interests. A set of agents are then trained to make recommendations to the user. Fab is designed to support so-called *parasitic users*, users who wish to provide no feedback but still receive recommendations. Such users are provided with global mean recommendations.

Claypool *et al.* developed *P-Tango*, a recommender system for online newspaper articles [**CGM$^+$99**]. *P-Tango* operates by combining predictions made by a collaborative filter and a content-based filter. In this system users explicitly provide preference feedback in the form of ratings of news articles, and also explicitly specify content in which they are interested. In addition, the system finds keywords in articles the user has rated highly and includes them in an implicit user profile in the hope that those keywords reflect what the user found appealing about the article. The collaborative filter finds similar users based on preference data and draws recommendations from this neighborhood. The content-based filter evaluates the correspondence between keywords in the user's profile and each article, in order to find potentially interesting articles. The two predictions made for each item are

then unified via a linear combination. The weights used in the combination are tuned on a per-user basis, over time, by finding values which minimize the predictive error on past ratings.

Burke has presented a survey of existing and possible hybrid approaches [**Bur02**]. First, many different methods a recommender system can use are identified, including collaborative filtering, content-based filtering, demographics, and other methods. Then, existing hybrid systems are classified according to which techniques they combine. Several of the approaches discussed have not yet been attempted.

Montaner *et al.* present a taxonomy of recommender systems [**MLR03**]. They identify eight dimensions by which recommender systems can be classified:

- User profile representation
- Initial profile generation
- Profile learning technique
- Relevance feedback
- Information filtering method
- User profile-item matching technique
- User profile matching technique
- Profile adaptation technique

The paper also lists possible approaches to each of these dimensions and discusses many recommender systems within this taxonomy.

7. Learning-based methods

Billsus and Pazzani suggested that it might be useful to find a way to apply some of the more mature and well-developed techniques of machine learning to the field of recommender systems and collaborative filtering in specific [**BP98**]. The approach presented involves using SVD to reduce the sparse preference rating data space into a dense lower-dimensional space representing some set of features. The idea is that this feature set can then be used as input to a traditional supervised machine learning algorithm, with the set of actual ratings as the label. As an example, the authors used a portion of the feature set to train an artificial neural network.

Rule-based learning algorithms have also been used in recommender systems. Sarwar *et al.* provided information about movies, with training rating data, to the Ripper rule-learning algorithm [**BHC98**]. The authors found that simply making all movie data available to Ripper did not lead to better results than either collaborative or content-based filtering alone. Better performance was only achieved when the authors manually selected a subset of the movie information to use. Another rule-learning collaborative filtering system is RACOFI, a Canadian music recommender, by Anderson *et al.* [**ABB$^+$03**]. Kim and Kim present another approach which learns rules relating items and also takes advantage of hierarchical structure among the items (for instance genres in music or movies) to learn rules relating categories of items [**KK03**].

Another supervised learning method which has been used successfully in the context of recommender systems is the Bayesian network. In one version of this approach, a Bayesian network is constructed with a node for each item in the domain. The possible values for each node correspond to the possible ratings on the item, including no rating at all. Some method is then used to learn structure from dependencies in the rating data [**BHK98**]. A similar approach was presented by Chen and George [**CG99**]. Robles *et al.* use a naive Bayes classifier [**RnM$^+$03**]. Robles *et al.* create a number of naive Bayes classifiers using confidence intervals and choose the best one [**RnM$^+$03**]. Yu *et al.* use a hierarchical Bayesian approach to combine content and preference information [**YST$^+$03**]. Bayesian approaches such as these have been found to perform well, but both inference and structure learning in a Bayesian network are hard problems, and in a domain with a large number of items a large number of nodes may be required.

CHAPTER 3

Approach

1. Ratings data

In preliminary experiments, we determined that having the user provide detailed feedback to substantiate their rating was effective in improving the mean absolute error of predictions [**DG03**].

Initially in our research, we wished to have users attempt to explicitly identify the normally hidden reasons behind their preferences. To this end, we allowed the user to specify one or more features which were important to his or her opinion of the item. In addition, we recognized that an item can be viewed positively on the whole but that some features of the item might be viewed negatively (and vice versa), so the user would also specify whether each feature was positive or negative, and how much so.

Since that time, we have observed that the extent to which a feature is applicable is an important criterion for both user satisfaction and performance. In addition, a user's reaction to a feature can depend significantly on the strength of its observed "presence" (*i.e.* a little bit of violence may be good, but a large amount may be negative, or a little romance might be somewhat positive but a large amount of romance might be very positive). Based on these observations we now permit a user to:

(i) specify an overall opinion of the item,

(ii) select a relevant feature of the item,

(iii) specify the quantity of that feature in the item (or applicability of the feature to the item),

(iv) specify the degree to which the presence of this feature was a positive or negative factor.

This provides a more natural transliteration of the form of a typical interpersonal dialogue regarding the review of an item, but it does impose much more overhead on the user. For example,

> *I thought that movie was pretty good. There was a lot of action and special effects, which is great. It's just too bad that the romantic subplot was underdeveloped.*

becomes

Overall: 8

action	quantity 8	opinion 5
romantic subplot	quantity 2	opinion -4

Precisely, a rating by user u on item i is of the following form: Exactly one *overall* rating, $r_{ui} \in [r_{\min}, r_{\max}]$. This represents the user's opinion of i on the whole, where a rating of $r_{\min}$ indicates extreme dislike and a $r_{\max}$ indicates extreme preference. In our system, $r_{\min} = 1$ and $r_{\max} = 10$.

For this item, u must select a minimum of one feature which was important to his or her overall opinion. We represent this set of features used by user u on item i as F_{ui}. Suppose feature $f \in F_{ui}$ is chosen. Then the user specifies the *feature quantity* of f perceived to be in the item, $q_{ui}^f \in [q_{\min}, q_{\max}]$, where $q_{\min}$ indicates the complete absence of this feature while $q_{\max}$ indicates a very large amount. The quantities of the features selected are not required to sum to a particular value. The idea is not for the user to completely and in detail characterize the item in terms of features, but rather to indicate a few features which are important to his or her overall opinion of the item. Indeed, we do not present the quantity specification as an absolute and exact rating on a numeric scale but rather graphically, as an indication of quantity. In our system, $q_{\min} = 0$ and $q_{\max} = 10$. To complete this feature rating, the user must specify his or her *feature opinion* of this presence of f in the item, as $o_{ui}^f \in [o_{\min}, o_{\max}]$, where a rating of $o_{\min}$ should be used for extremely negative features, and $o_{\max}$ should be used for extremely positive features. In our system, $o_{\min} = -5$ and $o_{\max} = 5$.

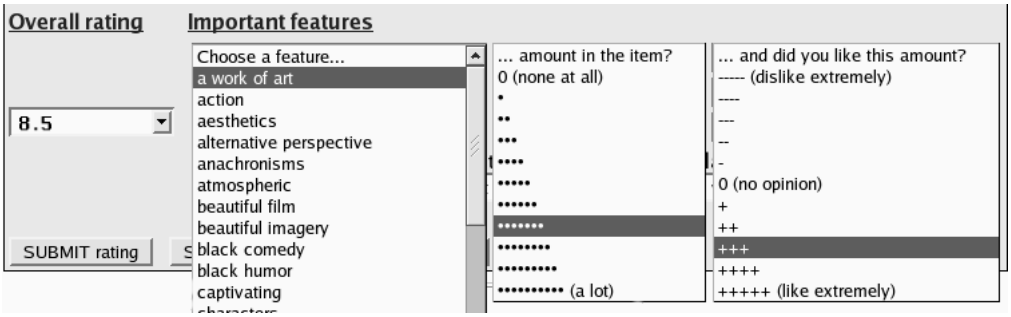


FIGURE 3.1. Screenshot of providing feature feedback for a movie in Recommendz.

We will denote the set of all *overall* ratings by user u as R_u, all *feature quantity* ratings by user u on item i as Q_{ui}, and all *feature opinion* ratings by user u on item i as O_{ui}. A *complete* rating by user u on item i then is (r_{ui}, Q_{ui}, O_{ui}).

The set of all users is U, the set of all items I, and the set of all features in the system, F. For convenience, we denote the set of all items rated by user u as I_u, and the set of all features used by user u as F_u.

We discuss the reaction of users to this more detailed rating system in Chapter 5, Section 2.

2. Nearest Neighborhood recommendation

In Recommendz we have implemented a nearest neighbour interpolation method. Creating recommendations for a given user requires the following steps:

(i) We compute the similarity between the user in question and each other user in the system. Any of a variety of similarity measures can be used in this step. We discuss the issue of similarity measures in Section 2.1. In Chapter 4, Section 1.4 we discuss how the system has been designed to reduce the computational burden of this step to improve scalability.

(ii) Sort the users by similarity and return a neighbourhood of k users.

(iii) For each item that has been rated by one or more of the users in the nearest neighbourhood, compute the average of reported overall ratings, with each rating weighted by the inverse of the user's similarity.

(iv) Report the resulting list of predicted ratings, or some subset thereof.

Users are given the option to ignore predictions of items that have been rated by only one user. Unless the user who has rated the item is very similar to the user receiving the prediction, a recommendation on an item rated by only one user should generally not be given much confidence.

2.1. Similarity measure. Perhaps the most important part of a nearest neighborhood-based collaborative filtering scheme is the similarity measure used. Different measures will, in general, lead to different groups of neighbours, and different sets of neighbours will usually have had different experiences – so different neighbourhoods will, in general, lead to different recommendations.

It is important to recognize that for many of the domains to which we want to apply collaborative filtering, the number of items to choose from is very large; much larger than either the number of users, or more importantly, the number of items each user has rated. The implication of this is that the number of items rated in common between any pair of users will often be small. As a result, it is important that any similarity measure between a pair of users handle sparsity effectively.

In addition, preference data tends to be noisy and so a good similarity measure should not be too sensitive to noise.

2.1.1. *Pearson correlation.* The Pearson Correlation coefficient has been widely used in collaborative filtering. Its design addresses the issues of sparsity and normalizing for user rating biases. First, Pearson correlation between two users is calculated based only on the items rated in common. Gaps in the rating data do not need to be filled in in order to compute the similarity, although we would tend to have more confidence in similarities computed on more data.

Second, each rating is normalized by subtracting the user's mean rating. Users may have different internal sensitivities to quality in movies. One user might rate all of his or her favourite items with relatively low values compared to the values given to favourite items by another user. This illustrates that the absolute values of rating values are not as important as the *relationships* between the values. By normalizing for user biases in rating, we hope to isolate the relationships between rating values.

In a system in which users have specified ratings over a set of items, we can compute the Pearson correlation between two users, u and v, with the formula

$$\text{correlation}(u,v) = \frac{\sum_i (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_i (r_{ui} - \bar{r}_u)^2 \sum_i (r_{vi} - \bar{r}_v)^2}} \tag{3.1}$$

where r_{ui} is the rating of user u on item i, and $\bar{r}_u$ is the mean rating of user u over all items. Note that this equation is a generic version of Pearson correlation, and versions of this equation specific to our rating system is presented in sections 3.1 and 3.2 (and specifically in Equations 3.2, 3.3, 3.4, 3.7, 3.8).

2.1.2. *Cosine Vector similarity.* Vector similarity comes from the field of Information Retrieval, where it is often used for document similarity comparisons. We would construct a vector for each document under consideration, with a position for each word occurring in the set of documents, and with the value at each position in the vector being the frequency with which the word occurs in the document. The distance between a pair of such vectors is then computed and interpreted as the distance between the documents, *i.e.* in terms of word similarity. This similarity is then a measure of how similar the composition of the documents are, in terms of word frequency. Since this measure has been successful in IR it was suggested as a possible similarity measure in neighborhood-based collaborative filtering systems, by researchers including Resnick *et al.* [**RIS+94**], Herlocker *et al.* [**HKB+99**], and Breese *et al.* [**BHK98**] who found Pearson Correlation to perform better. We initially investigated whether vector similarity would be more effective in our system than Pearson Correlation but found Pearson performed better.

Herlocker *et al.* discuss other potential similarity measures [**HKB+99**].

3. Hybrid recommendation

Our approach to hybrid recommendations uses both preference information and the content information which is supplied by the users in the form of features. We have developed two variations on this central theme.

3.1. Feature bias. The system uses a nearest-neighbor interpolation scheme to recommend items that are preferred by similar users. Typically, user similarity is computed as the Pearson correlation [**BHK98**] of items rated in common between the users. In our system the ratings are more complicated so while we also use Pearson correlation,

we actually compute three different similarity measures (defined as Eq. 3.2, Eq. 3.3, and Eq. 3.4) and then take a weighted average of the three to arrive at one similarity value.

Suppose we wish to compute the similarity between users u and w.

First, we have the similarity in overall ratings, which is just the Pearson correlation between the overall ratings of the items rated in common.

$$s_r(u,w) = \frac{\sum_i (r_{ui} - \bar{r}_u)(r_{wi} - \bar{r}_w)}{\sqrt{\sum_i (r_{ui} - \bar{r}_u)^2 \sum_i (r_{wi} - \bar{r}_w)^2}} \tag{3.2}$$

where in each summation, $i \in I_u \cap I_w$, the items common to both user u and w; and $\bar{r}_u$ is the mean overall item rating for user u.

In the first of two alternative approaches, we calculate statistics for feature ratings. In effect we compare user biases toward features, rather than comparing feature usage on an item-by-item basis (this is done in Section 3.2). Thus, the next similarity measure we compute will be the similarity in feature bias between the two users. For each user u, for each feature used, we calculate the mean feature opinion for feature f over all items, $\bar{o}_u^f$, and the mean opinion over all features, $\bar{o}_u$. Then, using the Pearson correlation:

$$s_o(u,w) = \frac{\sum_f (\bar{o}_u^f - \bar{o}_u)(\bar{o}_w^f - \bar{o}_w)}{\sqrt{\sum_f (\bar{o}_u^f - \bar{o}_u)^2 \sum_f (\bar{o}_w^f - \bar{o}_w)^2}} \tag{3.3}$$

where in each summation, $f \in F_u \cap F_w$, the features which have been used by both user u and w.

We wish to compute a third measure based on the similarity in quantity values assigned by the two users. The following equation uses Pearson Correlation to compare mean quantity ratings assigned to the features rated in common by the two users. This similarity can be interpreted as similarity in interest or experience (for instance two users will be more similar if both have scored many items having high or low levels of particular features).

$$s_q(u,w) = \frac{\sum_f (\bar{q}_u^f - \bar{q}_u)(\bar{q}_w^f - \bar{q}_w)}{\sqrt{\sum_f (\bar{q}_u^f - \bar{q}_u)^2 \sum_f (\bar{q}_w^f - \bar{q}_w)^2}} \tag{3.4}$$

where in each summation $f \in F_u \cap F_w$, the features which have been used by both user u and w; $\bar{q}_u$ is the mean quantity rating by user u over all features and items; and $\bar{q}_u^f$ is the mean quantity rating from user u for feature f, over all items.

We now want to combine these three similarity measures into one. To do this we compute a weighted sum of s_r, s_o, and s_q using weights ω_r, ω_o, and ω_q, respectively. The final similarity between two users u and w then is:

$$s(u,w) = \frac{\omega_r s_r + \omega_o s_o + \omega_q s_q}{\omega_r + \omega_o + \omega_q} \tag{3.5}$$

With the ability to calculate this similarity s between any two users, we can find the k nearest neighbors of user u, which we denote as N_u^k. We will be interested in the subset of N_u^k who have rated item i, denoted N_{ui}^k. With this information, we can predict the rating of user u on item i:

$$\tilde{r}_{ui} = \frac{\sum_{w \in N_{ui}^k} s(u,w)(r_{wi} - \bar{r}_{wi})}{\sum_{w \in N_{ui}^k} s(u,w)} + \bar{r}_u \tag{3.6}$$

Note that we normalize the predicted rating according to the users' overall rating biases, and then make the actual prediction by adding the target user's overall mean rating.

3.2. Item-by-item feature usage comparison. Another approach is to compare feature usage on an item-by-item basis, rather than comparing the statistics of feature usage as was done in Section 3.1. The basic approach is the same as that taken above. We use the same overall similarity s_r but we will redefine s_o and s_q with the s'_o and s'_q functions below.

In Section 3.1, we considered two users to be more similar to one another if they had similar biases in their feature ratings, i.e. if they used features in a similar way *in general*. In this approach we only consider two users to be more similar if they used features in a similar way *on the same items*.

In this version of the feature opinion similarity measure, we compare the individual feature opinion ratings for features both users used in items they rated in common. Before giving the equation, we will define $C_{uw} = \{f : f \in F_{ui} \cap F_{wi}, \forall i \in I_u \cap I_w\}$, *i.e.* for users u and w, C_{uw} is the set of all features that were used by both users on commonly rated items.

$$s'_o(u,w) = \frac{\sum_{f \in C_{uw}} (o_{ui}^f - \bar{o}_u^f)(o_{wi}^f - \bar{o}_w^f)}{\sqrt{\sum_{f \in C_{uw}} (o_{ui}^f - \bar{o}_u^f)^2 \sum_{f \in C_{uw}} (o_{wi}^f - \bar{o}_{wi}^f)}} \tag{3.7}$$

The feature quantity similarity measure is computed similarly:

$$s'_q(u,w) = \frac{\sum_{f \in C_{uw}} (q^f_{ui} - \bar{q}^f_u)(q^f_{wi} - \bar{q}^f_w)}{\sqrt{\sum_{f \in C_{uw}} (q^f_{ui} - \bar{q}^f_u)^2 \sum_{f \in C_{uw}} (q^f_{wi} - \bar{q}^f_{wi})}} \tag{3.8}$$

Once these two similarity values are computed, the predicted rating for item i is computed using Eq. 3.5 with s_o and s_q replaced with s'_o and s'_q, respectively, and then equation 3.6 is used to calculate the predicted overall rating $\tilde{r}_{ui}$.

4. Feature Suggestion

In our system, it is up to the users to contribute features which can then be used by all. Because of this, the number of features in the system is constantly growing. In order to encourage feedback and to be better able to compare user similarity, we would like users to use preexisting features to rate items. At the same time we cannot simply present the users with a list of all of the features in the database, as that would be overwhelming. What we wish to do is suggest relevant features. This notion of relevant features is itself a subjective issue, however. The suggested features should be relevant to the item to be rated and should provide useful information to the system. This is a preference elicitation problem, since we wish to intelligently determine a subset of features about which feedback from the user would be useful in determining the user's preferences (see Chapter 2, Section 3 for prior work in this field).

Recommendz suggests features using a combination of the following six techniques:

4.1. Controversy. Controversial features are those about which users differ widely in opinion. If users have, in the past, disagreed strongly in their opinions of such features, then knowing the opinion of future users on the same features should help discriminate between users with very different underlying reasons for item preferences.

We consider two methods of determining the controversiality of a feature: entropy and variance.

4.1.1. *Entropy.* One suggested measure of the controversiality of ratings is a high entropy [**CSP03**]. The authors showed that entropy is a useful measure for determining the usefulness of an item to a preference elicitation scheme (in fact it is even more useful

when combined with popularity). In our system, to find controversial features, we would be looking for those features with a high entropy in terms of feature opinion.

We can compute the entropy $H(f,i)$ of the feature f for item i. This entropy can be computed with respect to the feature's opinion value or quantity value and in the equations below we indicate these as H_o and H_q, respectively.

$$H_o(f,i) = -\sum_{x \in [-5,5]} P(o_i^f = x) \log_2[P(o_i^f = x)] \tag{3.9}$$

where $P(o_i^f = x)$ is the probability that the opinion rating given by any user for feature f on item i has the value x.

$$H_q(f,i) = -\sum_{x \in [0,10]} P(q_i^f = x) \log_2[P(q_i^f = x)] \tag{3.10}$$

where $P(q_i^f = x)$ is the probability that the quantity rating given by any user for feature f on item i has the value x.

We can also have a combination of the feature's popularity for this item (in terms of the number of ratings, not preference) and the entropy, as Carenini *et al.* found to be the most effective metric for preference elicitation. Those researchers used the product of entropy and the logarithm of the item's popularity.

4.1.2. *Variance.* As discussed in the preceding section, entropy has been suggested by other researchers as a useful measure of the controversiality of items in a recommender system. Other researchers have used variance, instead, as a measure of controversy [**GRGP01**].

We should also make the observation that an item (or feature, in our case) can have a high entropy but a low variance, meaning users tend to disagree on the rating to assign to the item, but not by much. On the other hand, a high variance and low entropy might indicate a few outlying ratings but something close to consensus aside from that.

In Recommendz, we have used variance in feature opinion value as a measure of how useful a feature will be. Variance is easily and quickly calculated, unlike entropy, and we have found, that variance produces qualitatively good results.

4.2. Correlated Features. As of writing, the system has a database of nearly seven hundred features. While we need to select relevant features from the list, note that

the *Controversy* method described above will only suggest features which have already been used on the item by other users. It is possible, especially when only a few ratings have been entered for the item, that many features which are relevant to the item will not have been used yet. We would like to explore the feature space to find these unused yet relevant features. We can randomly choose features to suggest and hope to find something relevant (in fact the system can and does do this, as discussed in section 4.6), but we can approach this task in a more intelligent way, by examining some measure of similarity between features.

We can calculate the correlation in usage of any two features, over all items. The idea is that if two features are often used in similar quantities on the same items, then the features may be related in some way. Given these correlation values, if we know that one or more features are relevant to an item because they have been used to provide feedback on that item, then we can calculate a list of features which are highly correlated to the original features. Correlated features are not necessarily synonyms (in fact they may have very different meanings) but they are similar in some sense, and it is possible that within a set of correlated features, any user may find some more applicable to a particular item than others. Therefore, by suggesting correlated features, the system has a better chance of covering more of the features which the user will find useful in rating the item. This would result in a more accurate representation of the user's true opinion of the item, and less frustration and effort on the user's part. Additionally, in the case of seldomly-rated items, there may be a very limited number of features which are known conclusively to be relevant (*i.e.* by previously being used in ratings of the item). If we do not consider correlated features then the system will have *no* chance to suggest any of these likely-relevant features, and either the user's rating will be less accurate than it might otherwise be, or the amount of effort required to complete the rating to the user's satisfaction will be much greater than it might otherwise be.

To calculate the correlation between two features f and g, we examine their values on items for which both have been used:

$$\text{correlation}(f,g) = \frac{\sum_{i \in I^f \cap I^g} \frac{\bar{q}_i^f \bar{q}_i^g}{(q_{\max})^2}}{|I^f \cap I^g|} \tag{3.11}$$

where I^f is the set of all items which feature f has been used to rate, and $q_{\max}$ is the maximum quantity rating, in our case 10.

Other formulae for computing the correlation between the two features could be substituted for Eq. 3.11. One alternative might be the mutual information between the quantities of the two features:

$$I_q(f;g) = \sum_{x\in[0,10]} \sum_{y\in[0,10]} P(q_i^f = x, q_i^g = y)\log_2(\frac{P(q_i^f = x, q_i^g = y)}{P(q_i^f = x)P(q_i^g = y)}) \tag{3.12}$$

As with entropy we can define mutual information of in terms of the feature opinion ratings rather than the quantity ratings:

$$I_o(f;g) = \sum_{x\in[-5,5]} \sum_{y\in[-0,10]} P(o_i^f = x, o_i^g = y)\log_2(\frac{P(o_i^f = x, o_i^g = y)}{P(o_i^f = x)P(o_i^g = y)}) \tag{3.13}$$

The system maintains a table of correlation values between pairs of features, and from the *Controversy* suggestion method we will have a list of features we already have decided to suggest. For each feature with high controversy we can find a list of correlated features. From this list we can choose a number of features to suggest, either according to the strength of correlation (i.e. we choose the n features most highly correlated to the features which are have already been chosen for selection), or probabilistically. In the case of probabilistic selection, the likelihood of choosing a correlated feature is proportional to the strength of the correlation.

In practice, we currently suggest several of the most highly correlated features, and then we choose several of the less well-correlated features probabilistically. The idea behind this is to balance the exploitation of features which we suspect will be relevant with the exploration of features which *might* be relevant but about which we are less confident.

4.3. Indirectly correlated features. We can also suggest several features which are not directly correlated to features which have actually been used on the item, but which are correlated to those features which are directly correlated to the features used on the item. It is also possible to iteratively suggest features which are correlated to features which in turn are indirectly correlated to the original features of the item.

Suggesting these indirectly correlated features allows us to explore the feature space more than focusing entirely on features known to be relevant. In practice, we currently

iterate to two levels of indirectly correlated features. We have not yet performed a detailed quantitative analysis to see how well this procedure works, but our initial qualitative observations suggest that it is quite useful.

4.4. Favourite features. There may be features which are not controversial, but have been used on the item in question very often, and therefore must be applicable to the item in some sense. Since users have a tendency to use these features on the item, they may find it convenient to have these features easily available, and so we can suggest several of these favourites. We refer to these features as *item favourites*.

So far, the techniques for suggesting features depend on having access to at least some features which have been used on the item in question. Items newly added to the system will have few ratings (and initially no ratings), and thus there will be very little knowledge about which features are actually relevant. One option would be to present few or no features; however, this places a burden on the user to come up with one or more features for the item from scratch. Users might find this intimidating or annoying. For that reason, we would like to provide the user with a list of features, even when we don't know which are relevant to the item. We can use several methods to find features which will hopefully be useful.

A given user may have a feature or set of features to which he or she pays a great deal of attention in the items under consideration. In this case the user may find it convenient to always have these features easily available for providing feedback, as the user may find them useful in discriminating the quality of items. For instance if a user often rates movies using the *action* feature, then it may make sense to suggest this feature to this user, even when the movie in question contains no action (this absence may in fact be important if action is a favourite feature). We refer to these features as *user favourites*.

Similarly, there may be features which are favourites over all users in the system. Since users find these features useful, we can suggest some of them. We refer to these as *global favourites*.

4.5. Globally controversial features. We can suggest features which are controversial, i.e. have been considered controversial across ratings by all users.

4.6. Random features. We can suggest some features completely at random, in order to explore the space and hopefully find some relevant yet seldomly used features which escaped the notice of our other suggestion methods.

CHAPTER 4

Implementation details

In order to evaluate the effectiveness of our approach, and to determine whether a community of users would be enthusiastic about our system and its unique feedback method, we implemented our approach as the Recommendz recommender system and made it available through the internet.

Since we have developed what is essentially a package to create a recommender website for Zope on Unix, we also discuss the structure of the software to some extent.

The system discussed in this chapter was designed and written over approximately a year, and consists of many thousand lines of html, C, and Python code.

1. Software infrastructure

The overall architecture is based on dynamically generated web pages which display the results from a "back end" computational subsystem. Data for these computational layers is stored in a database. This structure is described below and illustrated by Fig. 4.1.

For the presentation of the site on the internet, we chose the Z Object Publishing Environment (Zope) [**Zop**]. The characteristics of a recommender system require some sort of ability of dynamic presentation of content. New items may be added at any time, so it is more convenient to be able to pull item information out of a database and automatically generate the necessary web pages rather than have static pages written for each item. In addition, Recommendz has been designed to handle arbitrary domains, so the actual item information to be presented may not be constant, requiring custom scripts or templates to recognize what should be displayed, and to display it appropriately. Recommender

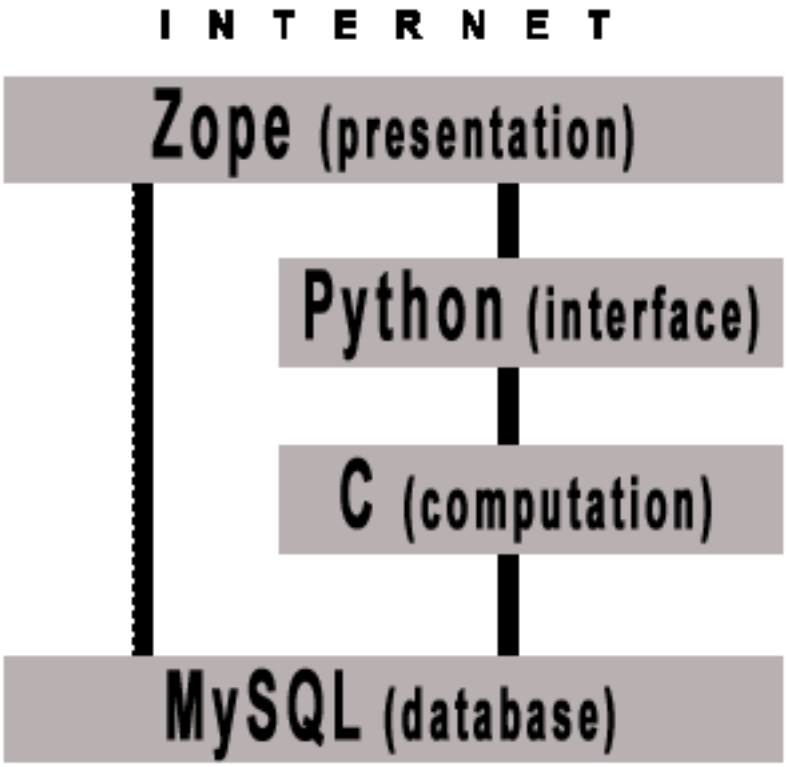


FIGURE 4.1. An overview of the software infrastructure used in Recommendz.

systems also require some amount of computation at the presentation level, for instance in searching through the database and in calling lower level functionality which will compute recommendations. An application server such as Zope is a good choice for a system with the requirements discussed above. Zope is an open-source system for the presentation of dynamic web pages. Zope can present regular html, or its own combination markup and scripting language which is based on html and Python [**Pyt**].

Zope was selected because it is freely available, reasonably fast in its presentation, and has a large community of users. Another option might have been html pages using php to organize and present dynamic content.

Zope is not actually used to compute recommendations or any of the other computationally intense aspects of the system. For this, there is a collection of back-end code written mainly in C, with some of the less time-critical functions written in Python. To

allow communication between the layers of the system, an interface, written in Python, was established between Zope and the computational back-end.

Any recommender system also requires a database in order to store information about items, users, and the ratings of users on items. Each of these sets of data are either likely or certain to grow during the life of the system, and to grow to potentially very large sizes. A database provides a dynamic storage space and should be optimized for efficient queries. For the database, MySQL [**mys**] was chosen, since it is freely available, fast, and well-documented. Zope has built-in support for communicating with MySQL databases, which was also a positive factor.

1.1. Database structure. An important goal in the design of the database, as in the design of the rest of the software, was genericity with respect to the items being used. We wished to have one database, but within it to potentially have several distinct item types. We also wished to be able to make recommendations from within a particular item type, or, as a future direction, potentially be able to treat all items as one domain and make recommendations within all, for instance measuring similarity between users based on ratings on movies *and* books, rather than one or the other, and then make recommendations.

This genericity we wished to achieve meant simple things such as calling the table for storing items "`item`" rather than, say, "`movie`"; and slightly more complicated decisions such as having a domain identification number as a field in the `item` table, and providing fields for the storage of any description of the item as might be available. For this last set of fields, we decided on three: one for a basic description of the item, another for any additional information which might not be of general interest but good to know nonetheless, and a third to store any other names by which the item might be known.

It is also worth noting that the database structure used here can be used for traditional collaborative filtering schemes which do not involve feature ratings, as well as for content-based filtering schemes since the item table can store item information which is as detailed as desired.

1.2. Content. For the movie domain discussed in this thesis, the database was initially populated with the movies contained in the EachMovie data set [**Eac**]. The EachMovie data set is actually a collaborative filtering data set from a now-defunct recommendation

site; however, since EachMovie used a different rating format than we do, we took the item information, which consisted of 1628 movies, and did not use the rest.

1.3. Computational code. The code which actually uses the database to compute recommendations is written in C. C was chosen for its speed and because the version of Python currently being used by Zope does not support extensions written in C++.

One of the main goals in writing the computational core was to make it modular at all levels. For instance, the functions which compute neighbourhoods use inter-user similarity functions which share a common interface and are abstracted so that the neighbourhood could be computed based on any similarity measure. Ideally, the interface of the recommendation engine with Zope would also be modular enough that Zope that different recommendation algorithms could be substituted transparently.

1.4. Online and offline computation. Inter-user similarity and predicted ratings based on neighbourhoods of users are computed online as the user views the pages which display recommendations, or as the user requests a recommendation.

Aside from recommendations, the other major computational burden on the system is the calculation of inter-feature correlation, which is then used in the feature suggestion process (described in Chapter 3, Section 4). A table of inter-feature correlations is computed each night and used in feature suggestion for the next day, avoiding a great deal of computation which could slow the system down.

2. Presentation

A highly accurate recommender system has little use if no one enjoys using it. Further, with few users little data can be collected. In this section we discuss the website through which the user interacts with the computational core of our recommender system.

2.1. Login and preferences. In our system we need to connect user identity with ratings, so each user must have an account on Recommendz in order to use the system. Creating an account consists of specifying a username, an email address, and password. Optionally, the user can specify demographic information including age and country of residence. This information is currently collected solely for interest and is not used in calculations; however, in his survey of hybrid recommendation systems, Burke [**Bur02**]

included demographics as a piece of information which might be profitably incorporated. Thus, collecting this data could lead to further hybridization of the recommendation engine in the future.

Once the account is created the user has the option of performing a few customizations of the site. The user is able to control the number of feature rating boxes that appear for new items (see Section 2.3).

Since we recognize that a user may use the same username and password combination on other sites, the password is stored on our system encrypted as an MD5 hash for the protection of privacy.

2.2. Main page. The main page presents the user with several options. These options are described in the following list (the options are numbered in reference to Fig. 4.2).

(i) *Recommendations.* In this area, the user is presented with a list of recommended items. The user can specify the number of recommendations to be presented, from the best 10 to 50. Alternatively, the user can choose to see the 10 items the system thinks he or she will dislike the most. The recommendation list is refreshed when the user returns to the main page.

(ii) *Domain.* If the database contains items from more than one domain then the user will have the option to choose which of these domains to work in and receive recommendations on.

(iii) *Items.* The items in the current domain are organized into tabs alphabetically by name. In addition, there is a tab which contains a list of some of the most often-rated items. This is done for convenience, reasoning that if users in the past have wanted to rate these items, new users may want to rate them in the future.

(iv) *Search.* Some domains may have far too many items to just navigate through the lists by name, so we have provided a search function. Users can search by name or by a feature which has been used to describe the item.

(v) *Item addition.* Users who have rated 10 or more items are considered trusted in some sense and are given access to a form through which they can suggest items to be added to the database. The information taken by the form can be

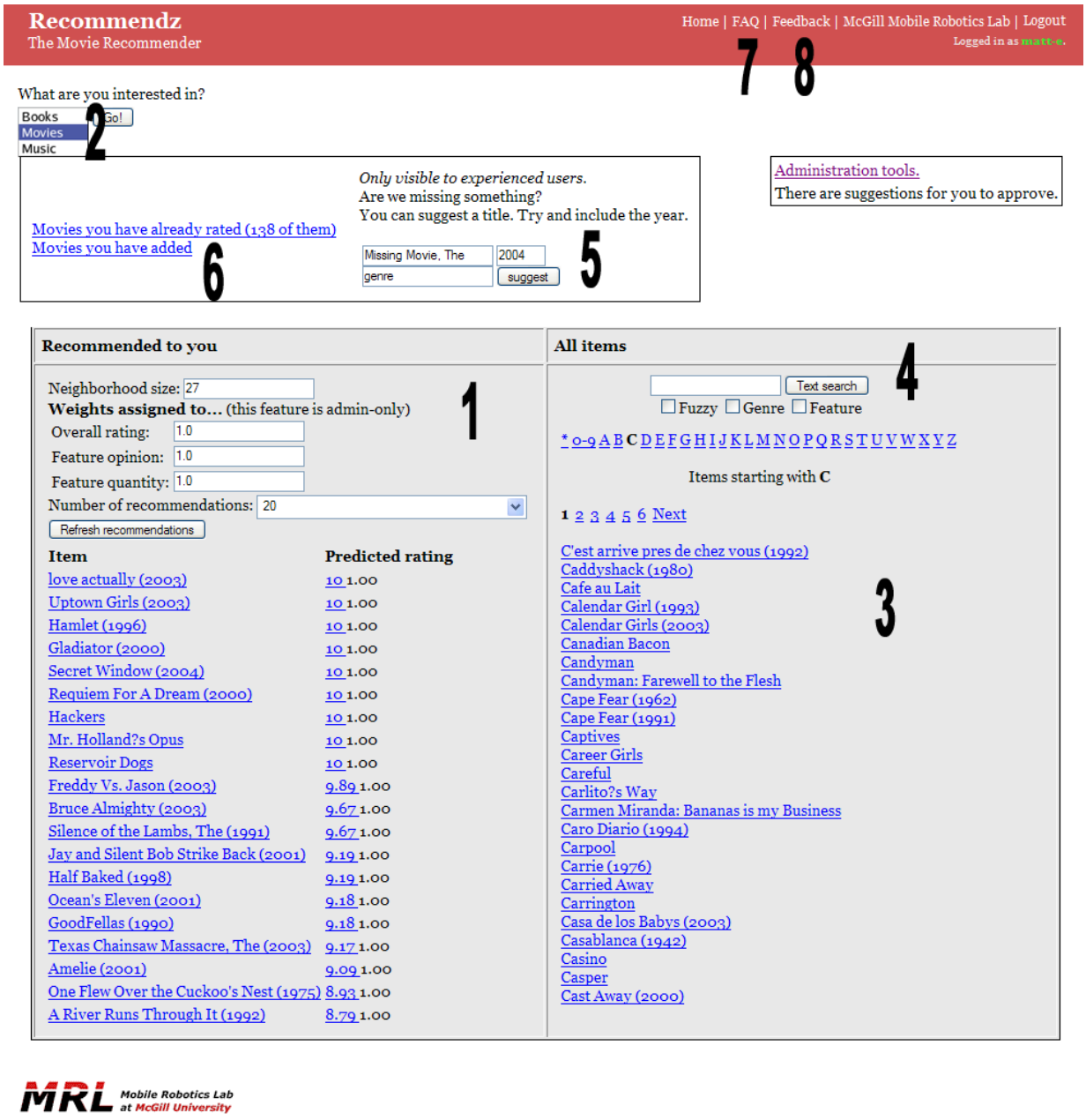


FIGURE 4.2. The main page of Recommendz. Various features of the system have been labeled and these features are described in Section 2.2.

customized to the item domain, if desired. For instance in suggesting movies the user is asked to enter the title, the year of release, and a description. The year and title are then combined into one string which is used as the item's name.

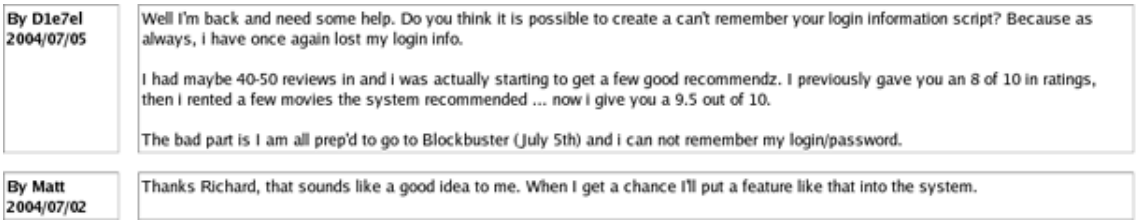


FIGURE 4.3. An example of typical user-administration interaction in the Recommendz feedback forum.

(vi) *Rating information.* The user can click these links to view the ratings and item suggestions that he or she has previously entered into the system.

(vii) *FAQ.* A list of answers to frequently asked questions about Recommendz.

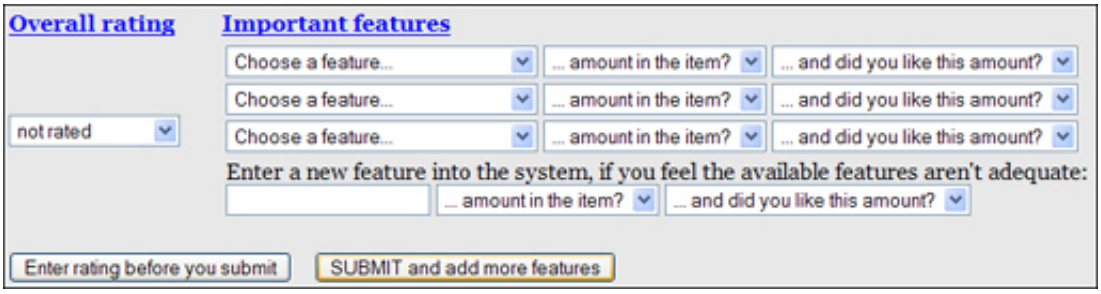
(viii) *Feedback.* We provided users with a forum in which they could quickly and easily post questions, suggestions, and feedback. An example of some typical interaction is illustrated in Fig. 4.3.

2.3. Rating interface. When an item is selected for viewing, the user is taken to a page which presents whatever information the system has about the item, and a form through which the user can specify his or her rating.

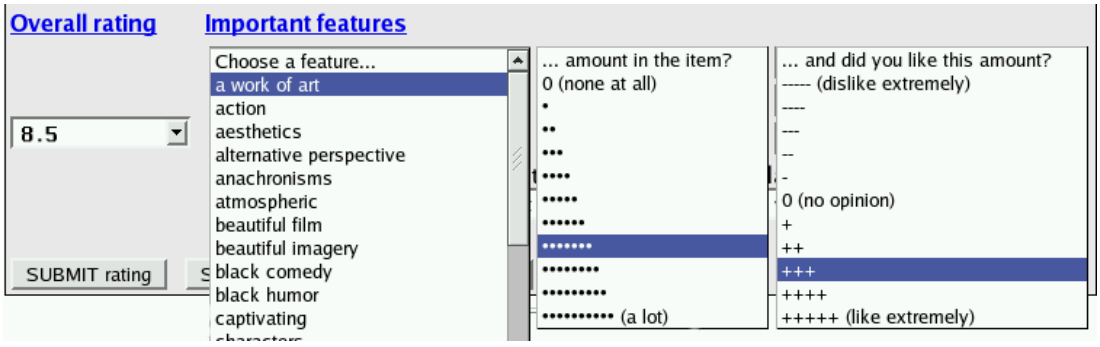
The information presented can be customized to suit the item domain being used and the type of information available for the item. The rating form provides boxes for the overall rating and enough spots for the user to specify as many as four feature ratings. For three of these positions, the user may choose from a drop-down list of preexisting features which have been suggested for use by this user on this item. The fourth of the positions does not contain a suggested feature, and instead is a place where the user can specify some other feature. If the feature specified already exists in the system then that feature will be used; otherwise, the new feature will be added and the feature rating which accompanied it will be registered.

A screenshot of the rating interface and an example of a user entering a rating on a movie is shown in Fig. 4.4.

2.4. Recommendation explanation. Herlocker *et al.* have persuasively argued that recommendations are more effective when accompanied by an explanation [**HKR00**]. For this reason we incorporated a page to justify the predicted rating given by the system.



(a) Rating interface.



(b) User entering a rating on a movie.

FIGURE 4.4. Screenshots of the rating interface of Recommendz.

In the list of recommended items, each predicted rating value is also a hyperlink to a page which provides information on how we arrived at that prediction. Currently, this page consists of a list of features which have been used on the item, and the mean quantity of each reported in the item. The page also gives a rough idea of how many other users have rated the item. When only a very small number of users have rated the item, the user can view the recommendation with a lower amount of confidence.

This is a weak form of explanation. A stronger sort of explanation would be personalized to the user and recommendation in question.

2.5. Recommendation-on-demand. The primary way the system provides recommendations is through the list on the main page (see section 2.2); however, if a user should log in to the system and be curious about a specific item, it would be convenient for him or her to be able to view the system's rating on that item without having to find

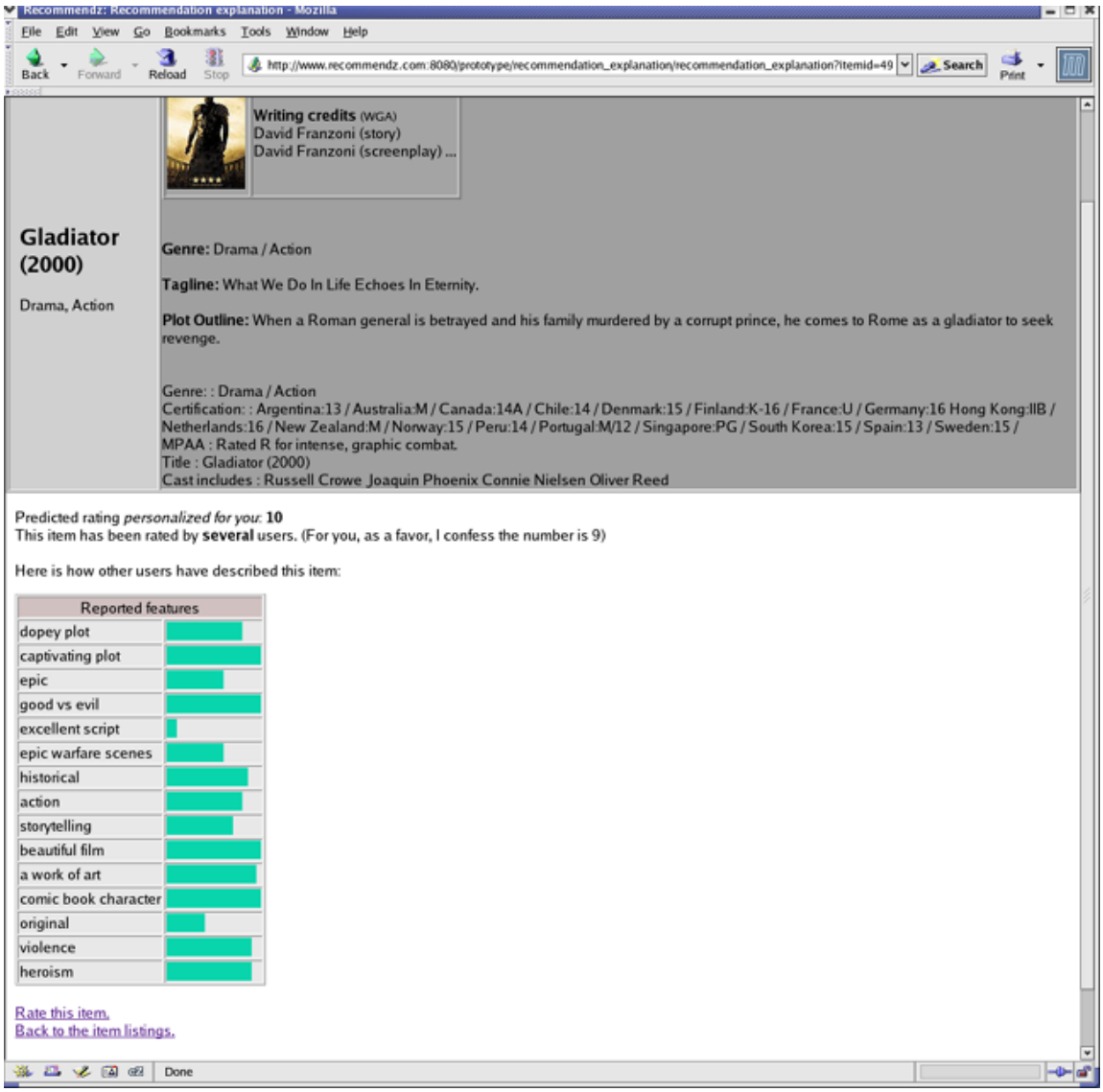


FIGURE 4.5. An example of the explanation for a recommendation of the movie *Gladiator*.

the desired item in the list of all recommendations. Instead, on the item information and rating page we have provided a button which, when clicked, will attempt to find a predicted rating on that item. This prediction is made in either one or two steps, as the case may be:

(i) The system attempts to find the personalized rating for the item as is computed in the main page's list of recommendations, *i.e.* as a personalized nearest neighbourhood-based recommendation (as described in Chapter 3, Section 2).

(ii) If the previous attempt at computing a personalized recommendation has not been successful, then the system computes the mean overall rating given to this item. This is an unpersonalized prediction and likely to be less accurate than a personalized prediction (see Chapter 5), but we believe that having some sort of recommendation will be more useful to the user than nothing. If at least one user has rated the item then at least this attempt at a prediction will be successful.

When a recommendation is made through this method, the system also provides an explanation page which includes the information discussed in section 2.4 as well as an indication of how the recommendation was actually made.

2.6. Item suggestion. In many domains which might be used in our recommender system, the set of relevant items may grow with time. In addition, whatever database is used initially may contain only a relatively small subset of the existing relevant items. Administrative users have the ability to add new items to the system but in the early days of the system's operation we found that some users wanted to be able to add items themselves. Allowing users the ability to add to the database directly would be a problem if a malicious user decided to spam the system with a large number of fake items, so we implemented a mechanism by which users can suggest a movie to the system, and those suggestions are then reviewed by an administrative user before being added into the system. Even with this level of oversight by the administrators we have set the item suggestion form up to only be visible to users who have rated ten or more items, so that those users who have already shown a genuine interest in and commitment to the system can become more involved in it.

One of the design goals of Recommendz is a recommender system which does not have to rely on extensive, preexisting textual descriptions of items, but there are some domains in which we may have access to such information. In these cases we may want to provide a user with a link to this information for his or her convenience, and so in the item suggestion step we can insert a script to check an authoritative information repository for verification that the item suggested actually exists. If it does exist then it is given to an administrative user to approve for addition to the database.

The item suggestion feature of the system has been very successful, with users having contributed over two hundred items within less than 10 months of access to the item suggestion feature.

3. Administrative features

Administrative users of the system are given access to many functions that casual users do not have.

Item control This includes the ability to add and remove items from the database, and to modify existing items. Also administrative users can review and either approve or reject the item additions suggested by casual users of the system (see Section 2.6).

Feature control It is entirely possible that some users of the system will be either less mature than others, or simply malicious. Such users may attempt to abuse the system by entering offensive features into the system. For this reason, administrative users have access to a page on which they can switch features to be *invisible*. An invisible feature will only be seen by the user who added and used it (and by any other users who subsequently try to add the feature to the system). This option is intended for features which were not entered with any conscious malicious intent, but all the same may not be appropriate for all users (*e.g.* features containing profanity or sexual references will not be appropriate for children). For features which appear to have been entered with actual malicious intent, an administrator can just delete the feature and any and all ratings in which it appears.

It is also possible that users will independently try to enter the same feature with spelling which is different enough from an existing feature that our search mechanism doesn't view them as the same. For this reason another page exists through which an administrator can easily merge two or more features into one new feature.

Within the past year, several features have been made invisible, but none have been so bad as to require being deleted.

Usage information The system keeps track of and displays upon demand, recent usage, including a list of the newest users and the amount of feedback each has provided. Various other pieces of information are provided to save the administrative users from having to actually log in to and query the database.

The system also provides some usage information graphically, including a graph of the number of items rated for each user, and a histogram of the number of

overall ratings made with each of the possible rating values (*i.e.* in the range $[1, 10]$).

Other information There are also several pages which display results in which an administrator might be interested, including the correlation values between features as calculated in Chapter 3, Section 4.2 using Eq. 3.11.

Options Administrative users have more control over the parameters used in the recommendation engine, such as neighbourhood size and the values of the weighting values ω_r, ω_o, ω_q. The average, casual user will likely not take the time to read about the details of the recommendation algorithm and so to present the users with these options would most likely be overwhelming. For such users these parameters are set and controlled by the administrators.

CHAPTER 5

Results

1. Evaluation

Before presenting our results, we will introduce the methods used to evaluate the system.

1.1. Error measure. A commonly used error measure in recommender system research is the *normalized mean absolute error* (NMAE) (e.g. in [**HKB+99, GRGP01, Can02**]). The *mean absolute error* (MAE) for user u is defined as

$$\mathrm{MAE}(u) = \frac{1}{|I_u|} \sum_{i \in I_u} |\tilde{r}_{ui} - r_{ui}| \tag{5.1}$$

The NMAE then is the Normalized MAE. The normalization is with respect to the particular rating system being used. If $r_{\max}$ and $r_{\min}$ are the maximum and minimum rating values being used (in our case 10 and 1, respectively), then

$$\mathrm{NMAE}(u) = \frac{\mathrm{MAE}}{r_{\max} - r_{\min}} \in [0, 1] \tag{5.2}$$

Carenini *et al.* suggest that MAE is a rough and "somewhat unrealistic" measure of error because it focuses on the accuracy of predictions and not the usefulness of a recommendation to the user. They propose that an error measure based in utility theory might be more appropriate [**CSP03**]. Very recently, Carenini and Sharma proposed and evaluated several measures of the effectiveness of recommender systems which are grounded in decision theory [**CS04**]; however, since (N)MAE is a very commonly used error measure, it is the measure we use to analyze the results of our system.

1.2. Recall. It is good to be able to make accurate recommendations, but if the system can only recommend a very small number of items, that accuracy will not be very useful. For that reason it is important to also consider the number of items the system can recommend to a given user. This measure is referred to as *recall*, and is the percentage of items for which the system attempts to predict a rating and is able to do so. Mathematically,

$$\text{Recall} = \frac{1}{|U|} \sum_{u \in U} \frac{|I_u \cap I_u^{\tilde{r}}|}{|I_u|} \in [0, 1] \tag{5.3}$$

where I_u is the set of items rated by user u, $I_u^{\tilde{r}}$ is the set of items for which the system could compute predicted ratings for user u, and U is the set of all users. Note that recall could also be defined per user, but we are interested in the system's overall recall.

1.3. Procedure. In evaluating the performance of Recommendz we used a *leave-one-out cross-validation* scheme. In this method, for each user, his or her rating on one item is held out, and then the remaining ratings are used to predict a rating on the hold-out item, using the prediction methods described below. It is possible that the system will not be able to make a prediction for this user and item (e.g. if this is the only user who has rated it, or if no one in the user's neighbourhood of most similar users has rated the item), but if the system *is* able to calculate a predicted rating, then the absolute error of that prediction is calculated and contributes to the user's NMAE as described above. The mean NMAE of all users is then recorded for the trial. A trial is performed for each setting of the number of nearest neighbours to be used and the weights in the

This variation of the cross-validation technique is commonly used in data sets where there is not enough data to partition the data into larger training and testing sets, and this is the major reason we selected it: In all of our tests we have used all users who have rated 10 or more items, so many of the users examined have a relatively small number of ratings to work with. This test set contains 225 users.

In our algorithm we have several adjustable parameters, including the number of neighbours used to make the prediction, and the weights ω_r, ω_o, ω_q used to compute the hybrid similarity in Eq. 3.5. We examined a large number of combinations of weights so in order to simplify the discussion of the results we have named several of the more useful weighting

schemes. These names are given in Table 5.1. Note that *Pure CF*, *Pure Opinion*, and *Pure Quantity* are not hybrid methods since they use only one of the three similarity measures.

Neighbourhood size ranged from 1 to 100.

TABLE 5.1. Explanation of the weighting combinations used in testing our approach.

	ω_r	ω_o	ω_q
Pure CF	1	0	0
Pure Opinion	0	1	0
Pure Quantity	0	0	1
Features Only	0	1	1
All	1	1	1
CF+	4	1	1
Opinion+	1	4	1
Quantity+	1	1	4

For comparison with a model-based approach to recommendation, we also evaluated the performance of *Eigentaste* [**GRGP01**] on our data set. Results of these tests are presented in Section 3.6.

1.4. Sparsity and similarity methods. In Chapter 3, Section 3, we presented two methods of calculating a hybrid similarity between users using feature information.

In the Feature Bias method we compute similarity by comparing bias towards features (Chapter 3, Section 3.1) and in the Item-by-Item method we only examine feedback on features which were used in common and on the same items (Chapter 3, Section 3.2). The former approach was actually found to relieve rating sparsity: There are nearly four hundred user pairs for which a similarity could not be computed based on the *overall* component of item ratings (*i.e.* traditional CF could not compute a similarity s_o), but for which feature-based similarity could be computed. There were roughly five hundred pairs of users who rated items but who had used no features in common. On the other hand, the Item-by-Item approach resulted in just over 18000 pairs of users who could not have a feature-based similarity computed, since the users may have rated features in common but not on the same items. Both methods only apply to feature information so both use the same overall similarity values.

As a result of this finding regarding sparsity, we decided to not only evaluate the two hybrid similarity approaches separately, but to also evaluate a Combination approach

(Section 3.4), in which we combine the two by using feature similarities from the Item-by-Item method where they could be computed, and to use the Feature Bias everywhere else, in the hopes that this would result in increased accuracy while avoiding the large reduction in recommendable items that could result from such sparsity.

1.5. Baseline performance. Note that in an appendix to [**GRGP01**], Goldberg *et al.* showed that NMAE for prediction against a random data set by guessing random values was either 0.333 or 0.282, depending on whether the distribution of ratings and predictions were uniform or normal, respectively. Clearly a recommender system must be able to beat these baselines by a significant margin in order to be of any use.

A more useful baseline for the performance of a recommender system is the NMAE achieved by non-personalized predictions which are calculated as the global mean ratings (*i.e.* the so-called "POP" method [**BHK98**]). A recommender system which makes personalized recommendations but can not consistently outperform POP is doing extra computation (compared to POP) for no real reason.

1.6. Evaluating feature suggestion. Evaluating the feature suggestion method proposed has been difficult, as the evaluation measure used must take into account not only any gains in accuracy provided by the use of suggested features but also the user satisfaction with the features which have been suggested (*i.e.* have any of the features the user had in mind been suggested). In addition, for any given rating only a small number of features are available and so comparisons of the accuracy of different methods is difficult. As a result of these issues we have left the nature of this evaluation an open problem and intend to examine it in future work.

2. User response

Since our rating system is new, the question arose as to how users feel about providing feedback. The initial concern was that the detail required might be too complicated, or too time-consuming and arduous. We felt this was a very important issue: The nature of our problem domain is such that accurate predictions will be useless if users are so annoyed by the interface that they never use the system.

In order to determine whether our rating system was in fact a problem, we examined feature usage. We found that among users of the system who are not affiliated with the research project, who had entered 7598 item ratings, the average number of features specified per item rated was slightly more than 2.5. In addition, among these same users 58% of all ratings entered were made using 3 *or more* ratings (with a maximum of 15 features entered by a user for a single item), 49% of all ratings were made using *exactly* 3 features, and only 23% of all item ratings were made using the bare minimum of only one feature. It is worth noting that by default the item rating screen allows the user to enter 3 features (the user enters more by clicking a button which reads "submit and add more features"), meaning that over half of all ratings made involved the user filling out all available feature slots, or specifying a new feature (see Fig. 4.4 for an illustration of the rating page). It is interesting to speculate whether providing the user with more spaces for specifying features would have led to users entering even more feature information on average.

Of the 956 users who had created an account as of writing, 568 had entered at least one rating into the system. As reports on other recommender systems do not typically publish information of this sort, we have nothing to compare this figure with and only include it for completeness, not drawing any conclusions, and point the user to our other usage results which are very promising.

Due to these results, we are confident that *on average* users do not find our system overly complicated or too arduous to use. There are exceptions of course. It is possible that many users find the process of providing detailed feedback fun in and of itself, but such a claim would require collecting user feedback about the system itself, something we have only done informally thus far; however, this hypothesis fits with results reported by Swearingen and Sinha indicating that users of recommender systems are willing to provide more feedback if they feel they are getting something in return [**SS01**].

As of writing, the system contains approximately 670 features and slightly over 2100 items, 1140 of which have been rated at least once. Since we compute inter-user similarities in order to make recommendations, only features and items which have been used by more than one user will be useful to the algorithm.

Fig. 5.1 illustrates the number of distinct users who have used each feature at least once in a rating. Of these features, 70% have been used by 2 or more users. Most features

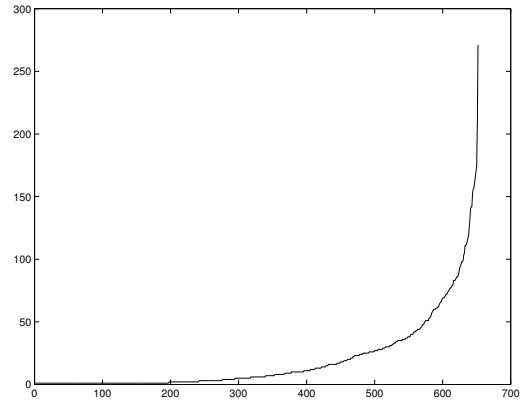


FIGURE 5.1. Features versus the number of distinct users who have used the feature. 70% of all features were used by 2 or more users.

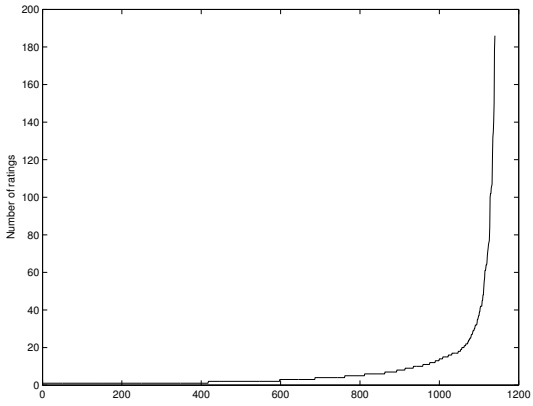


FIGURE 5.2. The number of ratings entered for each item (each item can only have one rating by each user at any moment). 63% of all items were rated by 2 or more users.

have been used by fewer than 50 users. Table 5.2 lists the ten most frequently-used features in the system, and the number of distinct users who have entered ratings using each.

In the movie database, there is a larger proportion of items which have been rated by only one user (63%). Fig. 5.2 illustrates the number of users who have rated each item. Just over 70% of all items have been rated by 5 or fewer users.

2.1. User feedback. We included a simple message board page on the website, so that users of the system could easily provide explicit feedback on the system. Many of the posts were questions about the system and its capabilities, but we received feedback as well.

For instance:

TABLE 5.2. The 10 most frequently-used features and the number of distinct users of each.

Feature	Number of users
action	271
comedy	210
clever	174
good acting	170
good storytelling	166
violence	161
adventure	157
sci-fi	156
funny	153
captivating plot	142

> I previously gave you an 8 of 10 in ratings, then I rented a few movies the system recommended ... now I give you a 9.5 out of 10. [1]

Another user wrote:

> I had fun with this system. It took 23 ratings before my tastes were really zeroed in on. I found that if there was a clunker in either likes or dislikes I could rate that and it helped a lot to focus on my tastes. Thanks. [2]

Some users did feel that the system is too complicated. For instance, one user wrote:

> What! Are! You! Doing! [3]

This user also complained that too much work was required to receive ratings.

Another user said, simply:

> It's too complicated [4]

however, it is worth noting that this user misunderstood the instructions slightly, believing that at least three feature ratings were required per item, when in fact only one is actually *required*.

In all, the negative feedback was less common than the positive, and the amount of use our system has enjoyed appears to be an indicator of a generally positive and enthusiastic response.

[1] Comment by user "D1e7el"
[2] Comment by user "grayster"
[3] Anonymous user via email
[4] Anonymous user

2.2. Item suggestion. Some users took a more active interest in the system and began suggesting more movies to be added to the database using our online suggestion form (see Chapter 4, section 2.6).

After the system had been up and running for 10 months, of the slightly more than 2000 movies in the system's movie database, 242, or over 10%, had been suggested by users who have no connection with the researchers.

It is also reassuring to note that to date there have been no "prank" or malicious item suggestions received, although some questionable features have been proposed, such as "Go go gadget Carpet!", and some reasonable features have been used inappropriately.

3. Experimental Results

In this section we present the results of tests designed to determine the effectiveness of our approach. Terms such as *Pure CF* and *Opinion+* refer to the weighting schemes described in Table 5.1, on page 50.

3.1. POP prediction. As described earlier, the POP prediction for an item is simply the global mean rating over all users. This is the sort of recommendation to be found in many popular sources (*e.g.* the metacritic [5] web site).

On our test set, the POP algorithm produced an NMAE of 0.2343.

3.2. Pure CF.

3.2.1. *Accuracy.* Pure collaborative filtering (*i.e.* no influence is given to the feature rating data) outperformed POP predictions on all but the smallest neighbourhood size but was in turn outperformed by all of the hybrid weighting schemes except for those which gave the most weight to the quantity values, *i.e. Pure Quantity* and *Quantity+* (see Fig. 5.3. The results for both *Pure CF* and POP are included in each of the graphs of results for comparison.

3.2.2. *Recall.* In terms of recall, the performance of *Pure CF* was slightly better than the hybrid weighting schemes under the Feature Bias similarity method. This is illustrated in the left-hand columns of Fig. 5.7 and Fig. 5.8, where the recall values for *Pure CF* are either equivalent to, or slightly better than the recall values for the hybrid methods.

[5] http://www.metacritic.com

3.3. Hybrid CF with Feature Bias.

3.3.1. *Accuracy.* In the results which follow we omit the NMAE values for the case where the neighbourhood used for predicting consists of only one user. The reason for this is illustrated in Fig. 5.3: All of the weighting schemes tested perform very badly when the neighbourhood is restricted to the single nearest user. The results are better illustrated by omitting these values.

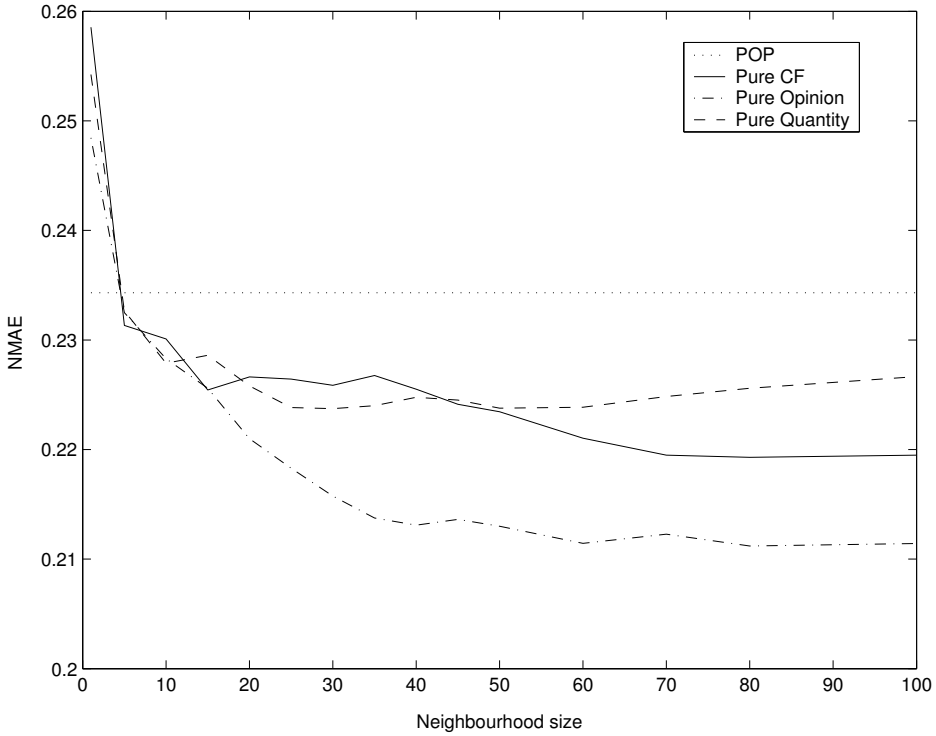


FIGURE 5.3. NMAE versus neighbourhood size for Feature Bias similarity schemes, for all neighbourhood sizes, including the neighbourhood of only the single nearest user

The graphs in Fig. 5.4 illustrate the NMAE of the various weighting schemes when using Feature Bias similarities, over neighbourhoods of sizes ranging from 5 to 100. All weighting schemes provide better performance than POP (the horizontal line at the top of each graph), and *Pure Opinion* and all of the hybrid weighting schemes provide better performance than traditional collaborative filtering (*Pure CF*). *Pure Quantity* does not outperform *Pure CF*.

The weighting schemes which consistently perform the best are those which place weight in the Feature Opinion values: *Pure Opinion*, *Opinion+*, *Features Only*, and *All*. On its own, the Feature Quantity bias information is not useful, as illustrated by the poor performance

of the *Pure Quantity* scheme; however, in combination with the Overall and Feature Opinion values, Feature Quantity bias does improve performance. In Fig. 5.5 we compare the best performing schemes from each of the figures in Fig. 5.4, along with an additional scheme that assigns equal weights to the Overall and Feature Opinion values ("*CF+Opinion*"). The fact that *All* provides better performance than *CF+Opinion* shows that including Feature Quantity information does improve accuracy. Despite this, the best performance is obtained with *All* at smaller neighbourhood sizes, and *Opinion+* at larger neighbourhood sizes. Clearly, Feature Opinion bias information is useful.

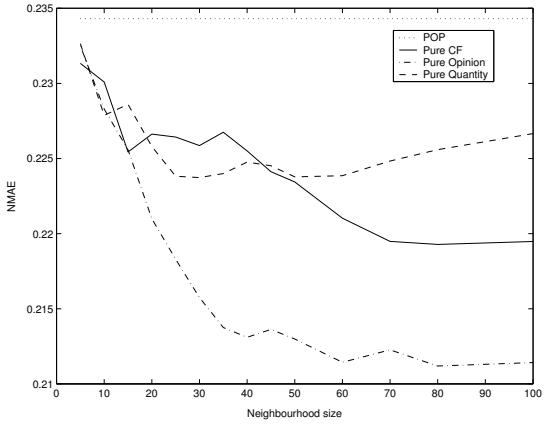
3.3.2. *Recall.* As observed for *Pure CF*, the recall values for the Feature Bias hybrid methods are either equivalent to, or are outperformed by, the recall of *Pure CF* (see the left-hand columns of Fig. 5.8 and Fig. 5.7). This is probably because using Feature Bias in computing the two feature-based similarity measures allows users with fewer items rated in common to be considered similar to each other. In the tests we are performing here this leads to decreased performance in terms of recall but in real-world use it could mean that a more diverse group of items can be recommended.

3.4. Hybrid CF with combined Item-by-Item comparison and Feature Bias.

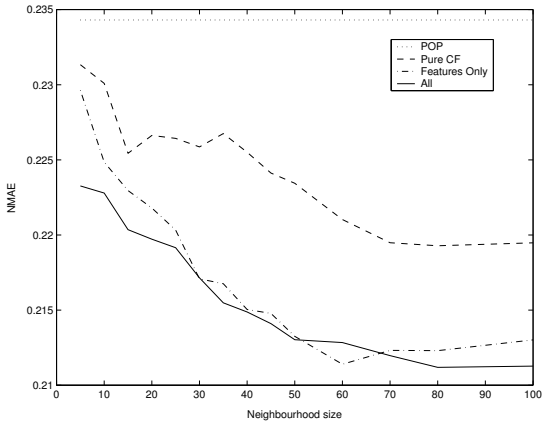
3.4.1. *Accuracy.* In this approach, which uses the Item-by-Item similarity values where it is possible to compute them combined with the Feature Bias similarity values to fill in the sparsity, we find that the accuracies of the various weighting schemes are very similar to those for the Feature Bias method presented in Section 3.3.1. In particular, note that the best performance is again given by the *Pure Opinion*, *All*, *Features Only*, and *Opinion+* weighting schemes.

This is perhaps unsurprising given that the Item-by-Item similarity is quite sparse so a large number of values need to be filled in with Feature Bias similarities; however, the NMAE values in this Combination similarity method reach lower values at lower neighbourhood sizes than in the Feature Bias method. This difference must be due to the influence of the Item-by-Item feature similarities. If this similarity were less sparse the differences might be more pronounced.

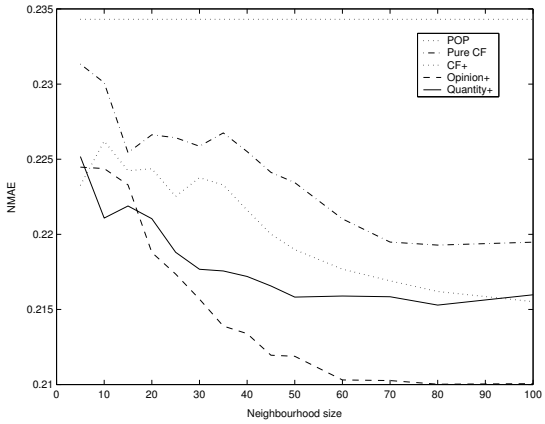
3.4.2. *Recall.* With the combination of Feature Bias and Item-by-Item feature similarity values, we see that the recall values of the hybrid methods are now consistently



(a) *Pure CF*, *Pure Opinion*, *Pure Quantity*



(b) *Pure CF*, *Features Only*, *All*



(c) *Pure CF*, *CF+*, *Opinion+*, *Quantity+*

FIGURE 5.4. NMAE versus neighbourhood size for Feature Bias similarity schemes

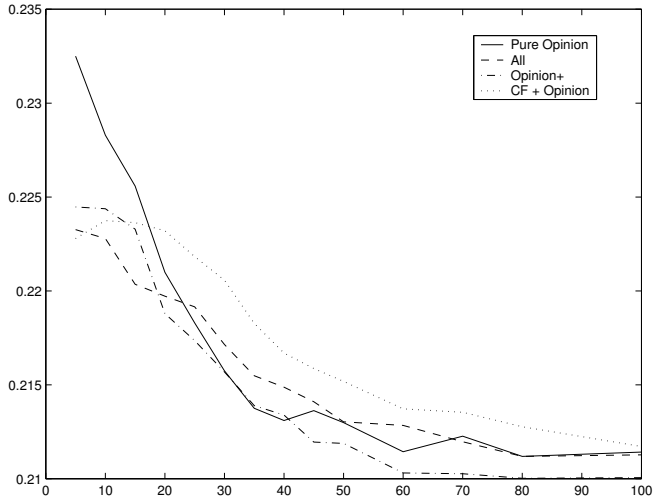
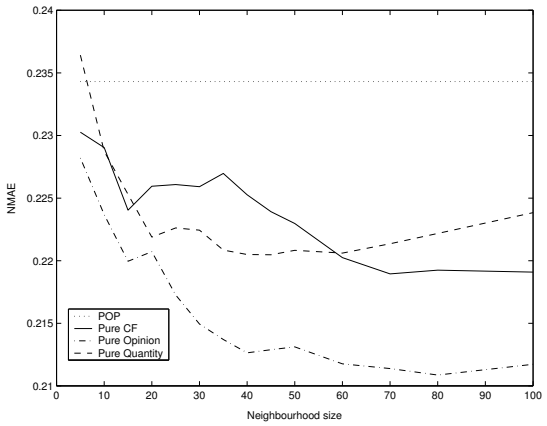


FIGURE 5.5. NMAE versus neighbourhood size for *Pure Opinion*, *All*, *Opinion+*, and *CF+Opinion*, a new scheme which gives equal weight to Overall and Feature Opinion values.

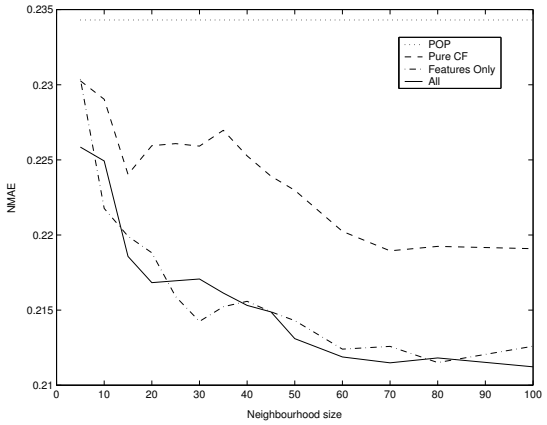
greater than those of *Pure CF* (see the right-hand columns of Fig. 5.7 and Fig. 5.8). The incorporation of Item-by-Item feature similarity wherever it can be computed means that many of the inter-user similarity values will be dependent on the users having not only used the same features, but having used the same features on the same items, which leads to a higher recall rate.

3.5. Hybrid CF with Item-by-Item comparison only.

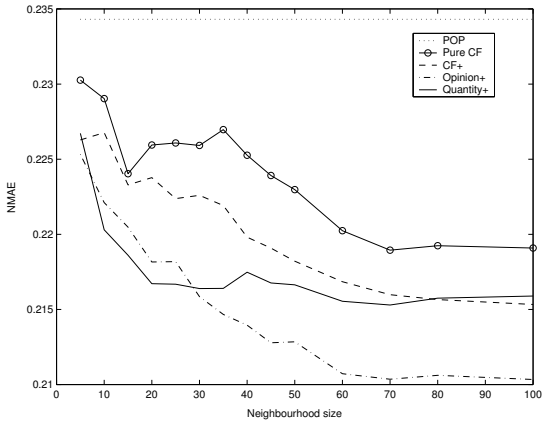
3.5.1. *Accuracy.* In the tests in which we used only Item-by-Item similarity values in calculating the inter-user similarities, we found that *Pure Opinion*, *Opinion+*, *Features Only*, and *All*, the weighting schemes which performed well in the two previous approaches, performed the best once again; however, the minimum error values achieved were, overall, not as impressive. The most interesting characteristic of these results is the fact that the NMAE drops off almost immediately at very low neighbourhood sizes and remains essentially constant from there on, whereas with Feature Bias the error values are higher until a larger neighbourhood is used (see Fig. 5.9 and Fig. 5.4, respectively). This also fits with the results of the Combination approach, where we noted that error values decreased more quickly than they had for the Feature Bias approach alone. That effect must have been the



(a) *Pure CF*, *Pure Opinion*, *Pure Quantity*

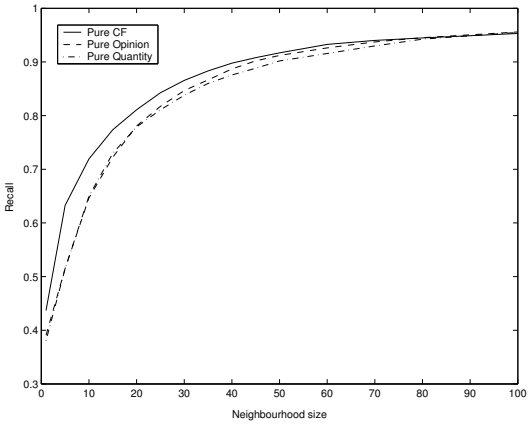


(b) *Pure CF*, *Features Only*, *All*

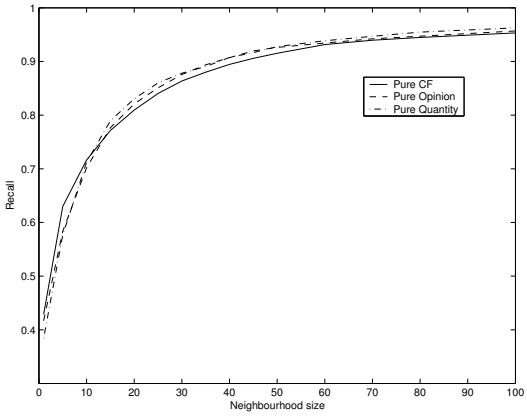


(c) *Pure CF*, *CF+*, *Opinion+*, *Quantity+*

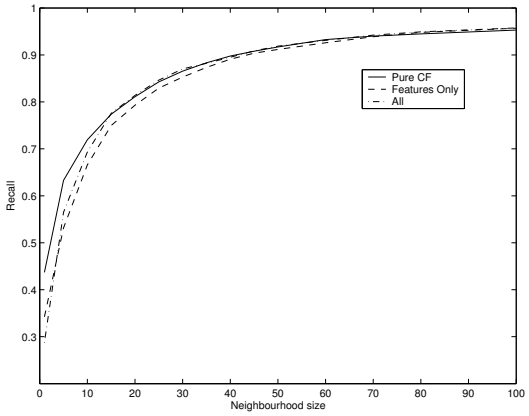
FIGURE 5.6. NMAE versus neighbourhood size for combined Item-by-Item and Feature Bias similarity



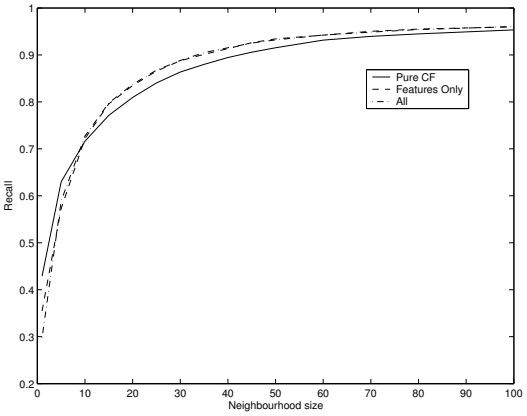
(a) Feature Bias similarity for "Pure" weighting schemes.



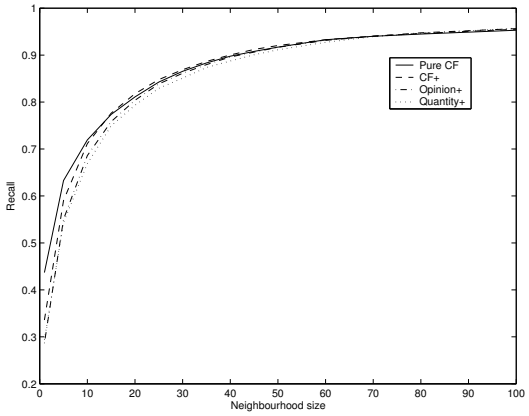
(b) Combination similarity for "Pure" weighting schemes.



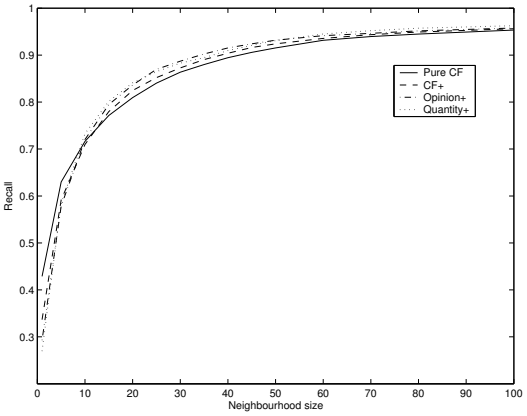
(c) Feature Bias similarity for *Pure CF*, *Features Only*, and *All*.



(d) Combination similarity for *Pure CF*, *Features Only*, and *All*.

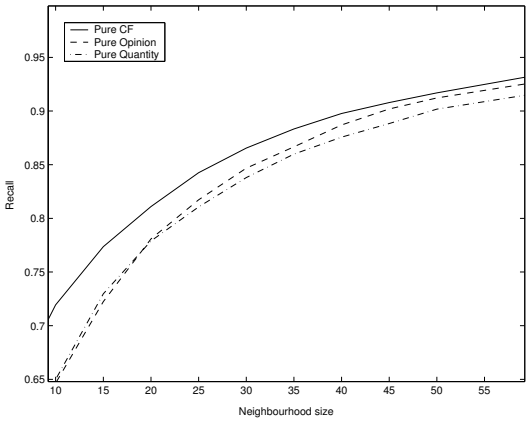


(e) Feature Bias similarity for *CF+*, *Opinion+*, and *Quantity+*.

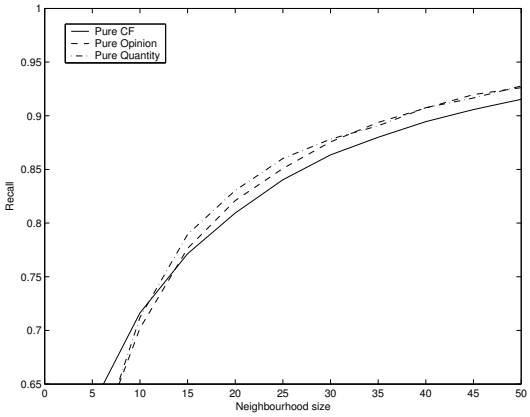


(f) Combination similarity for *CF+*, *Opinion+*, and *Quantity+*.

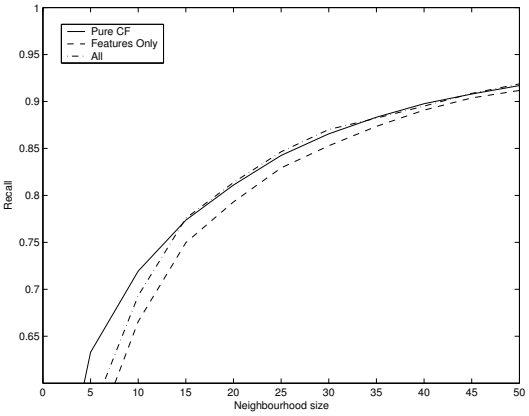
FIGURE 5.7. Recall versus neighbourhood size for all neighbourhood sizes.



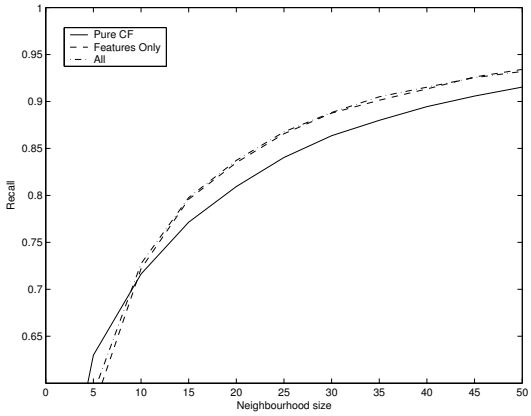
(a) Feature Bias similarity for "Pure" weighting schemes.



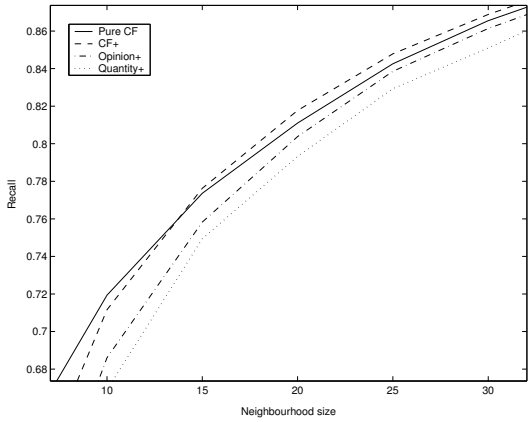
(b) Combination similarity for "Pure" weighting schemes.



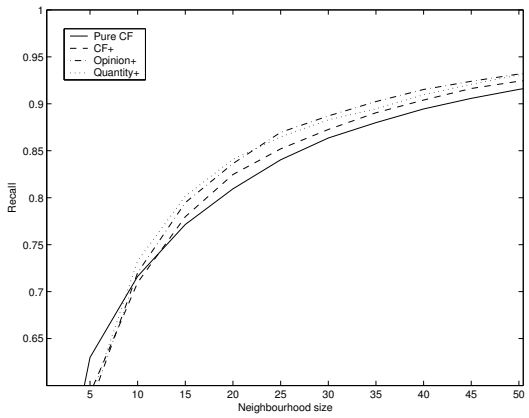
(c) Feature Bias similarity for *Pure CF*, *Features Only*, and *All*.



(d) Combination similarity for *Pure CF*, *Features Only*, and *All*.



(e) Feature Bias similarity for *CF+*, *Opinion+*, and *Quantity+*.



(f) Combination similarity for *CF+*, *Opinion+*, and *Quantity+*.

FIGURE 5.8. Detail of recall versus neighbourhood size.

result of the Item-by-Item similarities. This suggests that given the feature feedback information, even a small number of similar users can give a good indication of the preferences of another user.

3.5.2. *Recall.* As for the combination of Feature Bias and Item-by-Item similarity values, we find that using only Item-by-Item similarities consistently outperforms *Pure CF* (see Fig. 5.10).

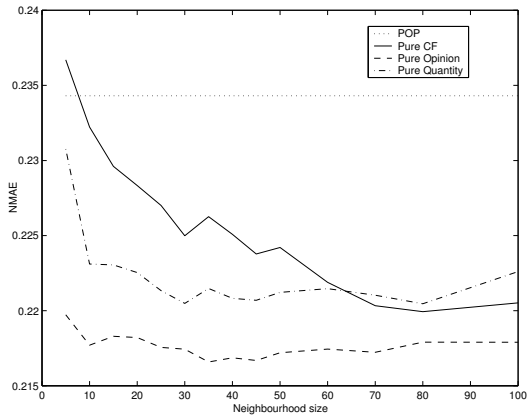
3.6. Comparison with Sparse Factor Analysis and Eigentaste. We obtained an implementation [6] of the *'Mender* [**Can02**] Sparse Factor Analysis recommendation engine and tested it on our ratings data, omitting users who have rated fewer than 10 items, as we did for our algorithms. In each test run, the data set was divided into a group of training and test users, with one tenth of the ratings selected randomly and held back as the test set each time. The training set was used to build a model using Sparse Factor Analysis (SFA) and predictions were made on the items in the test set to yield an NMAE.

First we doubled the number of linear regression steps used to initialize the Factor Analysis algorithm, and used off-the-shelf parameters for the other variables, including normalizing ratings by per-item mean ratings rather than the per-user mean ratings used in our method (the authors of the SFA package state that in their system normalization by per-item mean ratings outperformed normalization by per-user mean ratings). Over the 10 test runs we found an average NMAE of 0.2175 which is comparable to the best average NMAE of our algorithm.

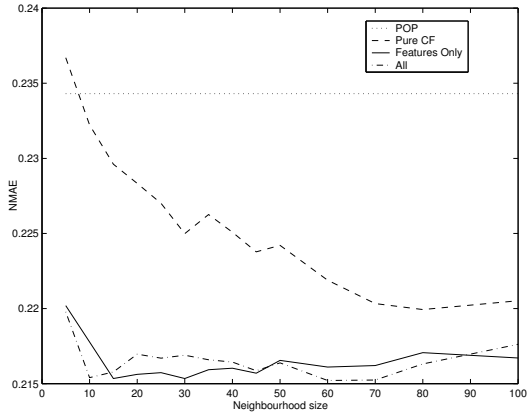
Next we increased the number of iterations used by the Factor Analysis portion of the model-building phase by 50%. With these settings we found that results improved to the point where test runs were consistently performing better than the average results of our method, with NMAE values around 0.18.

We also implemented and tested the Eigentaste [**GRGP01**] algorithm on our data set, using the same method to divide the data set into test and training users. PCA was used to reduce the data space to two dimensions and the training users were then clustered in this space using k-means clustering, with $k = 25$. Test users were then projected into the space, were assigned to a cluster, and recommended well-liked items from that cluster. Over 20

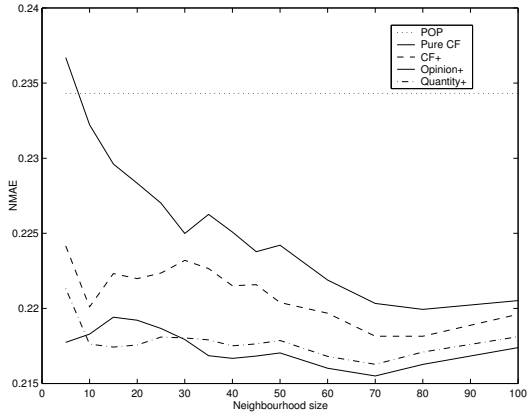
[6] http://www.cs.berkeley.edu/ jfc/'mender



(a) *Pure CF*, *Pure Opinion*, *Pure Quantity*

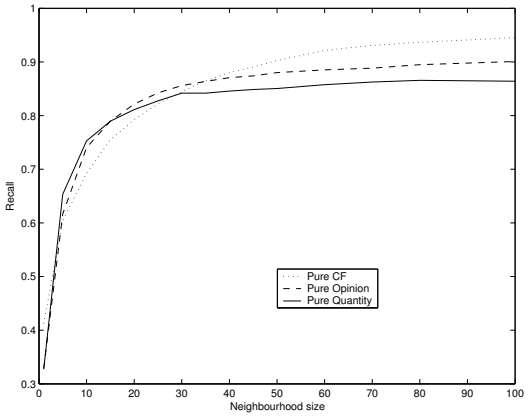


(b) *Pure CF*, *Features Only*, *All*

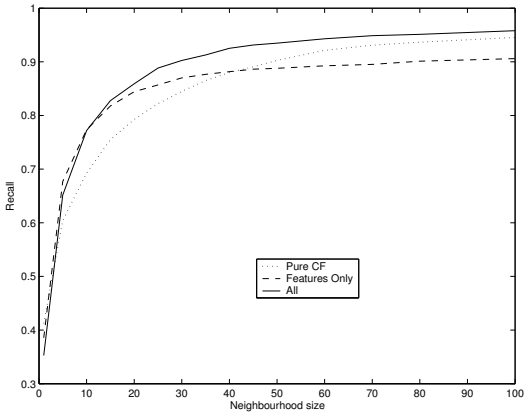


(c) *Pure CF*, *CF+*, *Opinion+*, *Quantity+*

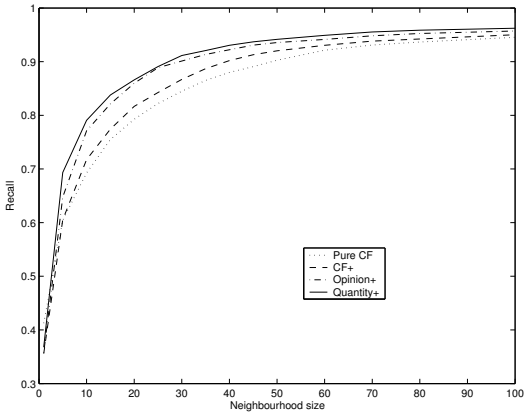
FIGURE 5.9. NMAE versus neighbourhood size for Item-by-Item similarity



(a) *Pure CF*, *Pure Opinion*, *Pure Quantity*



(b) *Pure CF*, *Features Only*, *All*



(c) *Pure CF*, *CF+*, *Opinion+*, *Quantity+*

FIGURE 5.10. Recall versus neighbourhood size for Item-by-Item similarity

test runs we observed an average NMAE of 0.2055 which slightly outperforms our method; however, our method is still competitive with error values around this level.

It is also worth noting that the more nuanced feedback collected through our system naturally lends itself to greater explanatory power than these other recommendation algorithms, including SFA. Although accuracy is of course very important, researchers are increasingly interested in the explanatory power of recommender systems [**HKR00**].

3.7. Feature correlation. It is difficult to validate the results of the feature correlation computation method except by manually looking at the results, and even then it is not obvious how to quantify a ground truth for these correlations. Due to this limitation, in this section we give an overview of some interesting qualitative results regarding feature correlation. In future work we hope to explore methods of quantitatively evaluating these correlations.

Table 5.3 (page 66) shows a list of the 10 most highly correlated features as determined by Eq. 3.11. As an example, for the Disney movie *Finding Nemo*, an animated film about talking fish, the user is suggested to use the feature "talking animals" (among 10 other features including "charming" and "social commentary") although no user has used this feature on this film.

Table 5.4 shows a list of the 10 pairs of features with the lowest correlations as determined by Eq. 3.11.

TABLE 5.3. The 10 most highly correlated feature pairs as calculated by Eq. 3.11

great animation	family fun
cynical	directed beautifully
cartoon	Disney
fantasy	cartoon
social commentary	foreign
likeable characters	great acting
talking animals	computer graphics
foreign	directed beautifully
funny	cartoon
cinematography	foreign
illogical	ridiculous

TABLE 5.4. The 10 feature pairs with the lowest correlations as calculated by Eq. 3.11

erotic	children
complex plot	children
not for kids	children
children's fun	violence
charming	violence
interesting plot	realism
predictable	intelligent
predictable	not for kids
deep	violence
complex plot	beautiful imagery

CHAPTER 6

Discussion

In this thesis we have presented an approach to recommender systems which involves a novel system for providing feedback on items. This feedback goes beyond the traditional feedback mechanism of "like"/"dislike", allowing users to explain the factors which were important to their preferences, and provide content information for the items in the process. This extra information allows us to gain a better understanding of the similarities and differences between users, and also provides a generic method for collecting item content information and combining that information with the preference data of collaborative filtering. We have developed a method for using these ratings in order to provide recommendations on items for the users of our system. Our recommendation method is modular and allows for many different variations on the basic approach. Our recommendations can be supported by explanations consisting of item content information which was supplied by users during the rating process.

We have implemented this system in a website which has been used by nearly one thousand people, with several more joining each day. With the data we have collected from these users, we have explored and evaluated several variations on a basic approach to recommendation by finding a neighbourhood of users who are similar both in terms of overall preferences and the reasons they give for those preferences.

We have shown that incorporating this feature rating information into the inter-user similarity calculation improves performance over the traditional nearest neighbour-based approach to collaborative filtering, indicating that the additional information is useful to understanding user preferences.

Our results are based on the use of standard "classic" collaborative filtering algorithms and performance metrics and suggest that the quality of recommendations can be substantially improved by the use of semantic features. This is in addition to the explanatory power such features can provide, and their ability to generalize across items and users.

We have also examined two alternative recommendation algorithms:
Eigentaste [**GRGP01**] and Sparse Factor Analysis [**Can02**]. By tuning their parameters we can obtain recommendation performance, in terms of NMAE, that slightly outperforms the best possible with classical methods, even when using feature data.

This suggests that by combining the principles behind the Eigentaste or SFA algorithm with feature data, we may be able to outperform any existing algorithm while also being able to provide explanatory power and generalization across items. Such a combination would require a substantial extension of the existing methods, and while it seems quite feasible it remains an open problem for the present.

1. Future Directions

Several open problems have been identified through this research, and appear to provide promising directions for future work.

Our system requires more feedback on the part of the user than some other approaches. Although many users enjoy this aspect of our system, others have complained that they find it too time-consuming. It would be useful to attempt to accommodate both types of users, perhaps by introducing a hierarchy of user types which use more or less accurate recommendation methods depending on the amount of feedback the user is willing to provide. Some users might be willing to go as far as writing extended reviews on items while others would continue using the rating system we have described in this thesis. Other users might prefer to select a user stereotype from a list and receive predictions based on the system's understanding of that stereotype. Still other "parasitic" users might prefer less accurate, unpersonalized recommendations which require no feedback at all.

It is also possible that some of the users who have been less impressed with our rating system would feel more enthusiastic about it given a different interface. It would be useful to investigate other interfaces to the rating system. An ideal interface would be engaging and even fun, self-explanatory, and would present features in a such a way that the user

could easily find exactly what he or she is looking for. Based on user interaction with our system we are confident that we have a good start toward these goals, but by gathering user feedback on the interface we could improve it greatly.

Although we currently use a memory-based approach, recommendations are still made quickly despite a consistently growing user population; however, in order to allow the system to continue to scale, it would be useful to incorporate some of the methods designed to this task. In Chapter 2, Section 4 we have mentioned an approach to this task which involves recognizing irrelevant and redundant users, and another approach which uses data structures which are designed specifically to make memory-based recommender systems more efficient.

We currently use preference elicitation techniques to suggest features for use in rating items. These preference elicitation techniques could also be applied to suggesting items for the user to provide feedback on. In addition, we can experiment with different measures of inter-feature correlation and feature importance in order to present the user with more useful feature suggestions for rating.

Although we have observed general trends in the accuracy of our prediction method depending on the weighting scheme used, the best weighting scheme varies from user to user, and for many users there exist parameter settings which significantly outperform the overall results presented. Rather than using the best overall scheme for all users, it would probably be fruitful to apply a learning scheme to determine the best weighting scheme and neighbourhood size for each user based on known ratings, and to use those parameters to make recommendations in the future.

The fact that we found improved accuracy could be achieved from methods involving dimensionality reduction, and that our method improved results over classical collaborative filtering techniques suggests that better accuracy could be achieved by combining our feedback system with dimensionality reduction. This combination would build models on a greater amount of information, making use of that extra information while identifying important patterns and eliminating noise.

While continuing to strive for better performance in accuracy, we cannot forget that a recommendation which is not justified by a good explanation may not have much impact. The explanation of a recommendation should not only tell the user about the content of the item being recommended but also explain the similarities and differences between the

target user and his or her neighbours, and similarities and differences between the target user's preferences and the content of the item. To this end we should both explore different methods of composing and presenting personalized explanations, and make sure to get user feedback on those methods in order to determine which are the most useful.

Due to the nature of the features being suggested, it is possible that different users will interpret features in different ways. This could be a problem if the interpretations are wildly divergent, but in practice interpretations do not appear to vary enough to cause a serious problem in terms of the performance of the system, and no users have complained of this issue being a problem; however, it is possible that the system's performance could be further improved by reducing any impact this issue does have. One solution to this issue would be to require each user to specify a definition of the feature along with the short description, but this burden would likely be too onerous, both for the creators and users of features. This issue provides another possible direction for future research.

Finally, the promising results shown by Recommendz suggest that it would be interesting and useful to apply the system to other item domains, especially domains in which the features that are important to user preferences are difficult to extract automatically (*e.g.* images or music).

REFERENCES

[ABB+03] Michelle Anderson, Marcel Ball, Harold Boley, Stephen Greene, Nancy Howse, Daniel Lemire, and Sean McGrath. RACOFI: A rule-applying collaborative filtering system. In *Proceedings of COLA'03*. IEEE/WIC, October 2003.

[ama] Amazon.com.

[Aud] Audioscrobbler, http://www.audioscrobbler.com.

[Bal97] Marko Balabanović. An adaptive web page recommendation service. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 378–385, New York, 5–8, 1997. ACM Press.

[BDO95] M.W. Berry, S.T. Dumais, and G.W. O'Brian. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.

[BHC98] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the American Association for Artificial Intelligence*, pages 714–720, 1998.

[BHK98] John S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

[Bou02] Craig Boutilier. A POMDP formulation of preference elicitation problems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 239–246, 2002.

[BP98] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In J. Shavlik, editor, *Machine Learning: Proceedings ofthe Fifteenth International Conference*, pages 46–54. Morgan Kauffman, San Francisco, CA, 1998.

[BP99] Daniel Billsus and Michael Pazzani. A hybrid user model for news story classification. In *Proceedings of the Seventh International Conference on User Modeling (UM'99)*, June 1999.

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

[Bur00] Robin Burke. Semantic ratings and heuristic similarity for collaborative filtering. In *AAAI Workshop on Knowledge-based Electronic Markets*, pages 14–20. AAAI, 2000.

[Bur02] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, November 2002.

[BZM03] Craig Boutilier, Richard S. Zemel, and Benjamin Marlin. Active collaborative filtering. In *UAI 2003*. Conference on Uncertainty in Artificial Intelligence, August 2003.

[Can02] John Canny. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 238–245. ACM Press, 2002.

[CG99] Yung-Hsin Chen and Edward I. George. A bayesian model for collaborative filtering. In *Online Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, 1999.

[CGM+99] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *ACM SIGIR Workshop on Recommender Systems - Implementation and Evaluation*. ACM SIGIR, August 1999.

[CHW01] Sonny Han Seng Chee, Jiawei Han, and Ke Wang. RecTree: An efficient collaborative filtering method. In *Data Warehousing and Knowledge Discovery (DaWaK)*, pages 141–151, 2001.

[CLWB01] Mark Claypool, Phong Le, Makoto Waseda, and David Brown. Implicit interest indicators. In *Proceedings of ACM Intelligent User Interfaces Conference (IUI)*, January 2001.

[CMZF00] Zhixiang Chen, Xiannong Meng, Binhai Zhu, and Richard H. Fowler. Websail: From on-line learning to web search. In *Web Information Systems Engineering*, pages 206–213, 2000.

[CS04] Giuseppe Carenini and Rita Sharma. Exploring more realistic evaluation measures for collaborative filtering. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI2004)*, pages 749–754, 2004.

[CSP03] Giuseppe Carenini, Jocelyin Smith, and David Poole. Towards more conversational and collaborative recommender systems. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*. IUI, 2003.

[DG03] Gregory Dudek and Matthew Garden. On user recommendations based on multiple cues. In J.T. Yao and P. Lingras, editors, *WI/IAT 2003 Workshop on Applications, Products, and Services of Web-based Support Systems*, pages 139–143, Halifax, Nova Scotia, Canada, October 2003. Web Intelligence Consortium and IEEE, Department of Mathematics and Computing Science, Saint Mary's University.

[DLL03] S. Dasgupta, W.S. Lee, and P.M. Long. A theoretical analysis of query selection for collaborative filtering. *Machine Learning*, 51:283–298, 2003.

[Dro99] Jon Dron. CoFIND - an experiment in n-dimensional collaborative filtering. In *Proceedings of WebNet 99*, 1999.

[Dud] Dude, check this out : http://wwww.dudecheckthisout.com.

[Eac] Eachmovie data set, http://research.compaq.com/src/eachmovie/.

[GNOT92] David Goldberg, David Nichols, Brian M. Oki, and Douglas B. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, CACM 35(12):61–70, 1992.

[GRGP01] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[GSHJ01] 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.

[GSK+99] N. Good, J.B. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 1999 Conference of the American Association of Artificial Intelligence (AAAI-99)*, pages 439–446, 1999.

[HC00] Conor Hayes and Padraig Cunningham. Smartradio: Building music radio on the fly. In *Expert Systems 2000*, December 2000.

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

[HKR00] John Herlocker, Joseph Konstan, and John Reidl. Explaining collaborative filtering recommendations. In *ACM 2000 Conference on Computer Supported Cooperative Work*, pages 241–250, December 2000.

[HMB+89] S.R. Hathaway, J.C. McKinley, J.N. Butcher, W.G. Dahlstrom, J.R. Graham, and A. Tellegen. *Minnesota Multiphasic Personality Inventory-2: Manual for Administration*. University of Minnesota Press, 1989.

[HO99] John Herlocker and Mark O'Connor. Clustering items for collaborative filtering. In *Proceedings of the ACM SIGIR Workshop on Recommender Systems*, August 1999.

[Hof01] Thomas Hofmann. Learning what people (don't) want. In *European Conference on Machine Learning (ECML) 2001*, 2001.

[Hof04] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22:89–115, January 2004.

[KK03] C. Kim and J. Kim. A recommendation algorithm using multi-level association rules. In *Proceedings of the 2003 IEEE/WIC Conference on Web Intelligence (WI'03)*, pages 524–527, October 2003.

[KOR01] Jinmook Kim, Douglas W. Oard, and Kathleen Romanik. User modeling for information access based on implicit feedback. In *ISKO France Workshop on Information Filtering*, 2001.

[Lem04] Daniel Lemire. Scale and translation-invariant collaborative filtering systems. *Information Retrieval*, 7:1–22, 2004.

[ME95] D. Maltz and K. Ehrlich. Pointing the way: Active collaborative filtering. In *Proceedings of CHI'95*, pages 202–209. ACM Press, 1995.

[met] Metacritic.com.

[MLR03] Miquel Montaner, Beatriz López, and Josep Lluís De La Rosa. A taxonomy of recommender agents on the internet. *Artificial Intelligence Review*, 19:285–330, June 2003.

[mys] Mysql, http://www.mysql.com.

[Nic97] David M. Nichols. Implicit rating and filtering. In *Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering*, pages 31–36. ERCIM, November 1997.

[PHLG00] David Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, UAI 2000*, pages 473–480, Stanford, CA, 2000.

[Pyt] Python, http://www.python.org.

[RIS+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[RnM+03] V. Robles, P. Larra naga, E. Menasalvas, M.S. Pérez, and V. Herves. Improvement of naive bayes collaborative filtering using interval estimation. In *Proceedings of the 2003 IEEE/WIC Conference on Web Intelligence (WI'03)*, pages 168–174, October 2003.

[rot] Rottentomatoes.com.

[RV97] Paul Resnick and Hal R. Varian. Recommender systems. *Communications of the ACM*, 40:56–58, March 1997.

[RZ02] David Ross and Richard S. Zemel. Multiple cause vector quantization. In *Advances in Neural Information Processing Systems 15 (NIPS-2002)*. NIPS, 2002.

[SKB+98] B.M. Sarwar, J.A. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of CSCW'98*, 1998.

[SKKR00a] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender systems–a case study. In *Proceedings of the WebKDD 2000 Workshop*. ACM SIGKDD, 2000.

[SKKR00b] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–167, 2000.

[SKKR01] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the tenth international conference on World Wide Web*, pages 285–295, 2001.

[SKR01] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.

[SM95] Upendra Shardanand and Pattie Maes. Social information filtering algorithms for automating 'word of mouth'. In *Proceedings on Human Factors in Computing Systems*, pages 210–217, 1995.

[SPUP02] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM SIGIR, 2002.

[SS01] Kirsten Swearingen and Rashmi Sinha. Beyond algorithms: An HCI perspective on recommender systems. In *ACM SIGIR 2001 Workshop on Recommender Systems*. ACM SIGIR, 2001.

[THA+97] Loren Terveen, Will Hill, Brian Amento, David McDonald, and Josh Creter. PHOAKS: A system for sharing recommendations. *Communications of the ACM*, 40(3):59–62, 1997.

[UF98] Lyle H. Ungar and Dean P. Foster. Clustering methods for collaborative filtering. In *Proceedings of the AAAI Workshop on Recommender Systems*, 1998.

[VCBB96] Christopher C. Vogt, Garrison W. Cottrell, Richard K. Belew, and B.T. Bartell. Using relevance to train a linear mixture of experts. In *Proceedings of the Fifth Text Retrieval Conference*, 1996.

[Yah] Yahoo news, http://news.yahoo.com.

[YST+03] K. Yu, A. Schwaighofer, V. Tresp, W.-Y. Ma, and H. Zhang. Collaborative ensemble learning: Combining collaborative and content-based information filtering via hierarchical Bayes. In C. Meek and U. Kjærulff, editors, *Proceedings of UAI 2003*, pages 616–623. Morgan Kaufmann, 2003.

[YXS+02] Kai Yu, Xiaowei Xu, Anton Schwaighofer, Volker Tresp, and Hans-Peter Kriegel. Removing redundancy and inconsistency in memory-based collaborative filtering. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 52–59. ACM, 2002.

[YXT+02] K. Yu, X. Xu, J. Tao, M. Ester, and H. Kriegel. Instance selection techniques for memory-based collaborative filtering. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SDM'02)*, 2002.

[Zop] Zope: The z object publishing environment, http://www.zope.org.

Document Log:

Manuscript Version 0

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-LaTeX — 16 December 2004

Matthew Garden

McGill University, 3480 University St., Montréal (Québec) H3A 2A7, Canada, *Tel.* : (514) 398-7071

E-mail address: mgarden@cim.mcgill.ca

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-LaTeX