The New User Problem in Collaborative Filtering

Matt Crane

a thesis submitted for the degree of

Master of Science

at the University of Otago, Dunedin,

New Zealand.

October 20, 2011

Abstract

The new user problem is a problem that all collaborative filtering systems will face. How do we make recommendations when we do not know what the user likes?

This thesis provides an overview of collaborative filtering and the new user problem. We develop new metrics to measure the performance of a system when dealing with a new user. We also show a progression of methods for dealing with the new user problem the last of which is a substantial improvement over the best current method.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Collaborative Filtering

Collaborative filtering is a methodology by which users co-operate in order to determine what is interesting or useful from a larger set of items, for instance, which movies a user should view and which they should not. The basic premise is that people prefer only to look at relevant, interesting content and one way to filter out the irrelevant data is to collaborate with other users and consider what they believe to be relevant.

The first application of collaborative filtering dates back to 1992 to a system called *Tapestry* developed by Xerox PARC (Goldberg, Nichols, Oki, and Terry, 1992). It was an attempt to select interesting messages from emails received from mailing lists. Each user had to manually define a group of trusted users, and from this group the system could predict the rating the user would give for messages the user had not yet seen.

The same idea was applied to Usenet articles in the *GroupLens* system (Resnick, Iacovou, Suchak, Bergstrom, and Riedl, 1994). GroupLens made advances over Tapestry in that it was able to automatically select the group of trusted users for the current user, and relied solely on collaborative filtering to predict the ratings for each article. It did not, however, attempt to recommend any articles to the user although it did display the predicted rating to them. It was made available to the public in 1996 and judged to be a success by the authors of the system (Konstan, Miller, Maltz, Herlocker, Gordon, and Riedl, 1997).

MovieLens is the latest system from the GroupLens group, and attempts to both predict ratings for, and recommend, movies using collaborative filtering. The Movie-

Lens project is an on-going research effort and is available to the public at `http://www.movielens.org`.

Collaborative filtering systems have also been applied to other areas, for example music (*Ringo* (Shardanand and Maes, 1995)), and jokes (*Jester* (Gupta, Digiovanni, Narita, and Goldberg, 1999)). Both these recommender systems made small changes to the GroupLens algorithm. Collaborative filtering and recommender systems are becoming more prevalent on the world-wide web, particularly with sites that have a commercial interest in getting users to buy or rent their products such as Netflix, Amazon (Linden, Smith, and York, 2003) and Last.fm. More sites are also using these systems to tailor their content to the user, for example Google News (Das, Datar, Garg, and Rajaram, 2007) and YouTube (Davidson, Liebald, Liu, Nandy, and Van Vleet, 2010).

In October 2006 this commercial interest led Netflix, a DVD rental and online streaming company, to challenge the data mining, machine learning and computer science communities to develop systems that could beat the accuracy of its Cinematch recommendation system by 10%. To support this challenge Netflix released a dataset containing 100 million anonymous ratings of nearly 18,000 movies by over 450,000 users collected from 11 November 1999 through 31 December 2005.

The Netflix recommendation system recommends those items that have the highest predicted rating among those that the user has not yet rated. This is overwhelmingly the most common method of generating recommendations, which is why the terms collaborative filtering and recommender systems are often used interchangeably. We can easily imagine that a recommendation system could use other methods to generate the recommendations, although we know of no such system.

Bennett and Lanning (2007) summarise the Cinematch system which uses a variant of Pearson's correlation with item neighbourhoods (discussed in Section 3.2.2) and then, as ratings are added from the user, a multivariate regression based on these correlations to calculate the personalised predictions. Bennett and Lanning (2007) additionally illustrate some of the early progress towards the prize, and techniques used over the first 18 months of competition. The grand prize was awarded on 21 September 2009 to an amalgamation of three of the teams that had been on the leaderboard since the beginning of the contest. The Netflix Prize spurred a renewed interest in the field of collaborative filtering and competing teams introduced several new ideas and

techniques.

Netflix were going to run a second challenge that was to be accompanied with a second dataset, which would provide demographic information about each user. However, due to a lawsuit regarding this second contest and the de-anonymization of the first dataset (Narayanan and Shmatikov, 2008) and related privacy issues, this second challenge was not started.[1]

There are several open problems and issues that exist in collaborative filtering systems. The privacy of the ratings that have been provided by the user (Zhang, Ford, and Makedon, 2006; Hofmann and Hartmann, 2005; Polat and Du, 2005). Trust in the ratings that other users have given (Massa and Avesani, 2004; Massa and Bhattacharjee, 2004; Victor, Cornelis, Teredesai, and De Cock, 2008). A related problem is the practice of shilling the system, that is, producing large numbers of fake ratings to make a certain item be recommended to a larger number of users (Lam and Riedl, 2004; Mehta, 2007; Chirita, Nejdl, and Zamfir, 2005). The sparsity problem, where a large proportion of potential ratings for items by users are missing, is a problem that occurs with all recommendation systems (Huang, Chen, and Zeng, 2004; Papagelis, Plexousakis, and Kutsuras, 2005). Another problem that faces collaborative filtering systems is the inconsistencies of the ratings provided by the user. Amatriain, Pujol, Tintarev, and Oliver (2009) found that by asking users to re-rate items they were able to gain better prediction performance.

Another clear issue lies in the way that recommendations are produced. Currently most, if not all, systems rely on being able to predict ratings accurately based on previous ratings and then recommending those items that have the highest predicted rating. This method for generating recommendations is perhaps the most natural, encouraging responses such as "you liked x, then you will love y". Gunawardana and Shani (2009) identify two types of recommendation tasks, those that predict ratings of items, and those that predict usage patterns. They found that changing only the method by which similarity between users was decided (discussed in Section 3.2) significantly changed the relative performance of methods between the different prediction tasks.

This method of generating recommendations, and the reliance upon past ratings of the user or item, can sometimes lead to recommendations that are predictable, monotonous, non-serendipitous and un-interesting (Abbassi, Amer-Yahia, Lakshmanan,

[1] `http://blog.netflix.com/2010/03/this-is-neil-hunt-chief-product-officer.html`

Vassilvitskii, and Yu, 2009; Celma and Herrera, 2008; Weng, Xu, Li, and Nayak, 2008b; Ziegler, McNee, Konstan, and Lausen, 2005). Another potential issue that has arisen is that of recommendation stability. As ratings are entered into the system by the user, the predicted ratings, and therefore the recommendations, can vary wildly between visits to the system (Adomavicius and Zhang, 2010).

But, the first question that all collaborative filtering systems have to address is the cold-start problem.

1.2 Cold-Start Problem

What is generally called the cold-start problem actually consists of a family of three related problems that can occur with any collaborative filtering system. These are the new item, new user and new system problems. The terminology has been muddled in the past and the cold-start problem has been used to refer to any combination of these sub-problems. Additionally the term "cold-start" has been applied in situations where a small set of ratings already exist.

New Item Problem The new item problem refers to when a new item is introduced to the community. With no rating given to the item, the system cannot recommend the item to any users.

New User Problem The new user problem refers to when a new user enters the system and there are no ratings by that user. Most, if not all, prediction techniques rely on the past ratings of a user to calculate the predictions for that user. Without any past ratings the prediction methods fail to generate any predictions.

New System Problem When the system is new there are no ratings from any user for any item. Only a few attempts have been made at addressing the new system problem (Park, Pennock, Madani, Good, and DeCoste, 2006; Park and Chu, 2009).

This thesis shall focus on the new user problem. The methods discussed could be used in any recommendation system, but for clarity and because the experiments will deal with movies, the discussion will generally assume that movies are to be recommended.

1.3 Previous Approaches

How does a new user to the system, let us call her Anna, begin to start using a movie recommendation system? She could probably browse through a listing of available movies and their reviews and ratings, but then she is the one doing the work – the system is not providing any input. But, to provide input, the system needs to have some basis for making recommendations to Anna. We might expect the system to intervene and attempt to gather some information from Anna before she is allowed to use the system proper.

Previous approaches to the new user problem have mostly focussed on the metadata and prompting approaches outlined below. The new item problem has received a larger proportion of research, possibly due to the wealth of publicly available metadata about the movies.

1.3.1 Metadata

The metadata about items can be used to create content-based recommender systems. For instance, Balabanović and Shoham (1997) present a recommender that uses features extracted from the text of documents to recommend them to users. These content-based systems can then be hybridised with ratings-based systems as the ratings come into the system. In Jung, Park, and Lee (2004) a content-based recommender system was used to provide predictions of missing ratings, then these predictions were used by a ratings-based system to provide recommendations.

Aspect models attempt to create a model using the metadata to create a network from which a hidden, latent variable can be derived. In Schein, Popescul, Ungar, and Pennock (2001) a variety of aspect models were created using the actor and director information about a movie. It was claimed that this method could be used for the new user problem when applied to the demographic information about the user (occupation, age and gender) which was attempted by Lam, Vu, Le, and Duong (2008) with moderate success although they comment that more demographics to further split the users into groups would increase the success rate.

In Park *et al.* (2006) a set of pseudo-users and -items (filterbots) that rated, or were rated, algorithmically were created in an attempt to provide base ratings to the system, such that no user or item would be without ratings. This is also a method by which

the sparsity problem, mentioned above, can be alleviated. These filterbots were made to rate on a variety of the movies' metadata, for instance the number of awards the movie had been nominated for and won. In Park and Chu (2009) the filterbots method was expanded to include a model of the relationships between a user's demographic information and an item's metadata. The set of filterbots that was used by the system was also fine tuned.

In Pilászy and Tikk (2009) an attempt was made to just use the metadata of the movies. The generated results were compared with a system where the users had provided a small selection of ratings. They found that the accuracy of the predictions was higher when those ratings were available.

For this method, the user needs to provide some demographic data to the system. For systems that are employed on commercial sites, it is not unreasonable to expect this data to be required on registering to the system. For other systems users may not be willing to part with this information due to privacy reasons.

1.3.2 Prompting

Another method for alleviating the new user problem is to ask the user to provide a set of ratings for selected movies. When the new user enters the system, they are presented with items to rate, these are not recommendations, but are rather selected to garner as much information about the user as possible. When this process is over, the system enters its normal phase of operation whereby the user is recommended items and the ratings from the user form a feedback loop. The key to this method being successful is the order in which the items are presented to the user.

The first system to use the prompting approach was called Active WebMuseum (Kohrs and Merialdo, 2001), a system that recommended paintings and artwork to museum visitors. They experimented with ordering items by the variance and entropy of the ratings that were given to each item. These ideas were further extended by the MovieLens research group in Rashid, Albert, Cosley, and Lam (2002) in which they suggested numerous methods for ordering items and began investigation into personalised orderings of items.

The MovieLens research group further extended their own work in Rashid, Karypis, and Riedl (2008) presenting more methods for improving the order of items, and an-

other attempt at personalised orderings. Recently a new method for a non-personalised ordering of items was presented in Golbandi *et al.* (2010), and a personalised ordering in Golbandi, Koren, and Lempel (2011).

This method was adapted as a solution to the new item problem in Brun, Castagnos, and Boyer (2011), where sets of users to present to were selected for each item, such that the information gained for each item allowed it to be recommended to a larger number of users.

When demographic data about the user is not available, as in the Netflix dataset, then this method is the only viable method to help alleviate the new user problem. As such, this is the method that we shall be using in this thesis.

1.3.3 Other Approaches

In Ahn (2008), a new similarity measure was constructed for use between pairs of users that have small numbers of ratings. This new similarity measure takes into consideration the relationship between pairs of ratings, the absolute value of the ratings given, and difference from the item's mean rating. This method can only be applied when the users already have some ratings, their experimentation considering users with fewer than 20 ratings.

In Zhang, Meng, Chen, Xiong, and Duan (2009); Hang, Guiran, and Xingwei (2009) implicit measures of a user's interest in an item were taken as being indicative of a preference for the item. This method does rely on systems where the user is able to express interest in items, or provide relative ordering of items without providing explicit ratings for those items.

1.4 Thesis Outline

The development of a solution to the new user cold-start problem requires several components:

- A metric to measure and compare the performance of systems.
- A method for making predictions of users' ratings.
- An algorithm for making a choice of items to present, and their order.

In Chapter 2 we consider metrics that allow us to measure different aspects of how well a recommendation system is performing, and discuss metrics that have been applied to the cold-start problem. We derive a new metric, which addresses some of the problems found in previous metrics, and measures the relevant aspects for a cold-start system. In Chapter 3 we discuss several methods that are used to make predictions of how a user will rate a movie. In Chapter 4 we show a progression of methods by which the system can determine which item should be presented to the user to rate next. In Chapter 5 we discuss the results of the final method we derive and the previous best performing method as well as presenting future work.

Chapter 2

Metrics & Performance

An important part of the evaluation of, and comparison between, recommendation systems is the ability to measure how well the system is performing. There are a variety of aspects of performance that can be measured with recommender systems, and more in-depth discussions can be found in Herlocker, Konstan, Terveen, and Riedl (2004) and Gunawardana and Shani (2009).

2.1 Prediction Accuracy

To evaluate any method of generating predictions, we need to determine how to measure the accuracy of the predictions made. The most common methods of doing this are the Mean Absolute Error (MAE) shown in Equation 2.1 and the Root Mean Squared Error (RMSE) shown in Equation 2.2.

$$\text{MAE} = \frac{\sum_{r \in N} |\hat{r} - r|}{|N|} \tag{2.1}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{r \in N} (\hat{r} - r)^2}{|N|}} \tag{2.2}$$

Here $\hat{r}$ is the predicted rating, r is the real rating, and N is the set of predictions to be made.

Netflix chose to use RMSE because it includes a larger implicit punishment of "missed opportunities" (predicting a low rating, when the actual rating would be high) and "deal breakers" (predicting a high rating, when the actual rating would be low)

than when compared to MAE. RMSE as a measure of prediction performance was made popular during the Netflix Prize as a direct result, whereas previously MAE had been the more prevalent measure.

The normalised versions of the above metrics allow for direct comparison between datasets that have different rating scales. There are two methods of normalisation that are employed, scaling by the range of ratings, or by the expected error for a random predictor (McNee, Riedl, and Konstan, 2006). In the case of MAE these are:

$$\text{nmae} = \frac{\text{MAE}}{\max - \min} \tag{2.3}$$

$$\text{nmae} = \frac{\text{MAE}}{E\left[\text{MAE}\left(\text{random}\right)\right]} \tag{2.4}$$

where max is the maximum rating that can be given, min the minimum and $E\left[\text{MAE}\left(\text{random}\right)\right]$ is the expected MAE from generating random predictions. The equations for normalised RMSE are equivalent.

Furthermore the prediction errors can be calculated across all predictions for all users, or averaged across each individual user's prediction error. Measuring across all predictions biases the measurement towards those users that have more predictions to be made.

2.2 Top-n Lists

The metrics discussed above are useful when determining the overall accuracy of a recommender system. Sometimes, though, it is more useful to determine how well the system performs when presenting the user with a list of recommended items, as this is the typical use-case of a system.

In Breese, Heckerman, Kadie, and Others (1998) utility gain was employed, and defined, as a measure of how likely a user is to look at any given item in the list. Utility gain is defined as

$$R_u = \sum_i \frac{\max\left(r_{ui} - d, 0\right)}{2^{(i-1)/(\alpha-1)}} \tag{2.5}$$

where r is the rating given for the item by the user, d is the neutral vote (usually taken as the median of possible ratings, or global average), i is the position of the item in the list, and α is the position of the first item in the list such that there is an even

chance the user will view the item, and is typically set to 5. For items that are cannot be rated by the user, a rating of d is assumed, which effectively removes that item from the calculation.

Utility gain attempts to provide a measure of how useful the list of recommendations is for an individual user. For a group of users, U, the utility gain is defined as:

$$R = 100 \frac{\sum_{u \in U} R_u}{\sum_{u \in U} R_u^{\max}} \tag{2.6}$$

where $R_u^{\max}$ is the maximum utility for user u, which is achieved if all the items that the user is able to rate had been at the top of the presented list, and had been in descending order of rating given by the user.

In Schein, Popescul, Ungar, and Pennock (2005) an alternative measure called the Customer Receiver Operator Characteristic (CROC) is used which considers the hit-rate and false alarm rate of a list of recommendations. Each item on the list can be considered to fall into a binary classification, for which they used a like/did not like scheme, where a like was counted as a rating of 4 or 5 on a 5 star scale, and everything else was counted as did not like. Based on the relationship between the actual status and the predicted status each item on the list can then be considered either a true or false, positive or negative.

The hit rate is defined as the ratio of true positives to total positives (true positives and false negatives) $\frac{tp}{tp+fn}$ and the alarm rate as the ratio of false positives to total negatives (false positives and true negatives) $\frac{fp}{fp+tn}$. By varying the length of the list presented to each user, a new hit rate and alarm rate are gained and a curve of these points can be drawn.

The key difference between the CROC curve and the plain Receiver Operating Characteristic (ROC) curve is that each user is presented with lists of the same size, whereas in the ROC curve users may be presented with lists of differing lengths.

Additionally others have used the information retrieval measures of precision, recall and the $F1$ score to evaluate recommendation lists (Weng, Xu, Li, and Nayak, 2008a), where an item is judged as relevant if the user is able to provide a rating for it. The formula for these quantities are:

$$\text{precision} = P\left(\text{relevant} \mid \text{retrieved}\right) \tag{2.7}$$

$$\text{recall} = P\left(\text{retrieved} \mid \text{relevant}\right) \tag{2.8}$$

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.9}$$

The disadvantage of using precision/recall based metrics is that they consider each rating provided by the user to have equal importance, which we shall see later is not the case. Additionally they do not consider the information that might be gained from knowing a user is not able to rate an item (although to our knowledge no current system takes advantage of this information).

2.3 Additional Metrics

Other aspects of recommender systems are now being considered as areas that require measurement and improvement. One of these aspects is the diversity of the items within the recommendation list (Ziegler *et al.*, 2005), that is, making sure that the items that are recommended are sufficiently different from each other. For example, we could imagine that if a user is recommended a *Star Trek* movie, then having the rest of the recommendation list filled with *Star Trek* movies would not be satisfying.

Another aspect that is being considered is the stability of the recommendations over time (Adomavicius and Zhang, 2010). When a user provides a new rating for, or updates a rating of, an item the predicted ratings of un-rated items can vary substantially, which in turn changes the items that are recommended to the user.

The explainability of the recommendations that have been made is another area of recommender systems that is coming under more scrutiny also. In Herlocker, Konstan, and Riedl (2000) they found that users who encountered an explanation of the recommendations made were able to judge the suitability of the recommendations to a better degree. Tintarev and Masthoff (2007) provide a survey of explanation techniques and the features they provide to the user.

In McLaughlin and Herlocker (2004) a critique of using MAE as an evaluation metric for recommendations is made. Several points are raised including that an error of e will

have the same impact regardless of where the item is placed in a top-n recommender, and that highly accurate predictions of mediocre items drown out inaccurate predictions of highly rated items. To avoid these problems they consider a modified version of precision which includes unrated items as non-relevant.

Some are even going so far as to say that concentrating on measuring prediction accuracy only has hurt the development of recommender systems (McNee *et al.*, 2006). They raise several issues, such as items being recommended based on similarity alone which reduces the diversity of the recommendations and narrows the user to certain types of items. They also report that the user satisfaction in the recommendations is not correlated with high prediction accuracy.

2.4 The Cold-Start Problem

Given several ways of evaluating the usefulness of both predictions and top-n lists, how do we evaluate potential solutions to the cold-start problem? Previous methods have focused on either the accuracy after having gained a certain number of ratings from the user (Rashid *et al.*, 2002, 2008), or how many ratings the user was able to provide after presenting different numbers of items.

There is also the question of how many movies are presented before the user is able to rate a given number of them. This was used in both Rashid *et al.* (2002) and Rashid *et al.* (2008) as an auxiliary measure of performance. Other users have used precision and recall from the information retrieval field (Weng *et al.*, 2008a; Herlocker *et al.*, 2004; Ziegler *et al.*, 2005).

When a new user enters the system, they enter a priming phase in which they are prompted with a series of items that they should rate. We want the number of items that are presented to the user be minimised, while also allowing the system to gain as much information as possible from these selected items. After this priming phase is finished the system goes into its normal phase in which it is able to recommend items to the user. Including this small barrier to entry has been found to improve later contributions to the community (Drenner, Sen, and Terveen, 2008; Freyne, Jacovi, Guy, and Geyer, 2009).

The problem with using measures that perform evaluation at a single point is that they fail to take into account the future behaviour of the system. For instance, it is not

hard to imagine a system that performs better than others after receiving n ratings, but at $n + 1$ onwards it performs worse. It is also not unreasonable to imagine that comparing one system which presents highly informative movies that can be rated by the user less frequently, with another system presents uninformative movies that the user can rate frequently, the first of these systems will have a lower hit rate, but could have better error reduction.

To counter this we are going to measure the performance of the cold-start solution by measuring the accuracy of predictions on a hold out test set for the user after each movie we present. As a result when comparing two systems we will be able to see the difference between absolute performance after presenting a certain number of items, and also to determine how many additional items the more poorly performing system must present to get an equivalent prediction error.

Algorithm 2.1 outlines the testing procedure that we are going to follow. The test set for each user is constructed by taking the most recent 10% of ratings given by that user, this means that users with fewer than 10 ratings are excluded from the procedure. Each user's ratings are removed such that they will not influence the prediction calculation or selection of the next item to present. The accuracy of the predictions will be measured by using the MAE (Equation 2.1) of predictions on the test set. We take the average performance across users of the cold-start as we believe this will more fairly represent the experience of a new user to the system. We also restrict ourselves to a random sample of 500 users due to processing time.

Algorithm 2.1 Testing Procedure

```
for all users in sample do
    remove the user's ratings from the dataset
    while the user is able to rate more movies do
        present the next movie to the user
        if the user can rate the movie then
            add the rating to the dataset
            calculate the new prediction error on the test set for the user
        end if
    end while
end for
```

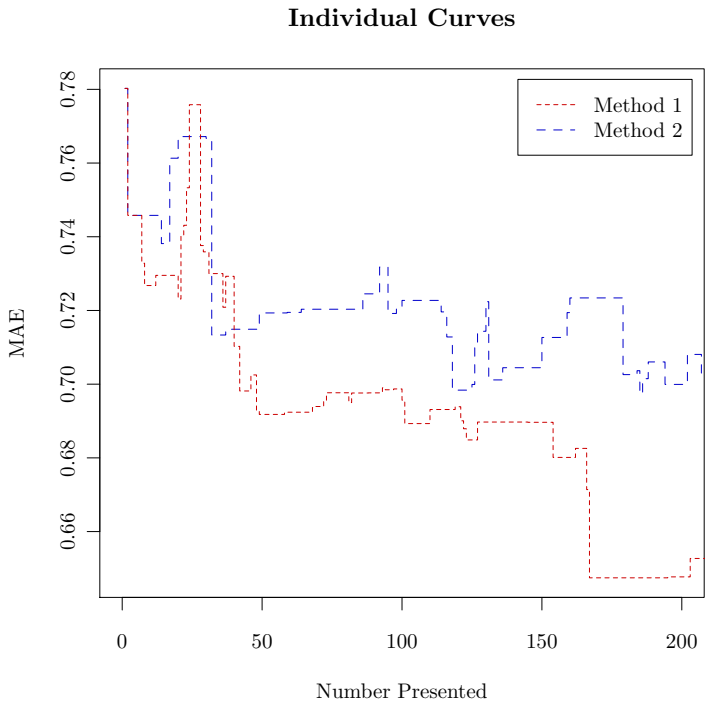

Figure 2.1: Individual Curves

When the data is presented as a graph of the prediction error as a function of how many items have been presented to the user then we get performance lines for each user, an example of which is shown in Figure 2.1. It should be noted that while we expect most of the users to only be presented with up to 50 items, we show the results for up to 200 for reasons outlined above. The plateaus are a result of the user not being able to rate some of the items that have been presented.

When these lines are averaged across the users that are being tested we get smoother curves that have the noise from the individual graphs removed, as shown in Figure 2.2. The lower each of these curves is, the better the method is performing. Clearly then, we could conceive of this as a problem of minimising the area under each of these generated curves (AUC). In fact, it is rare that one method outperforms another initially and then falls back, so the curves are more useful simply to allow us to make the comparisons mentioned above.

We believe that this method of representing the performance of a cold-start system yields a significant amount of information. It shows the absolute difference between the prediction error after presenting a certain number of items, and it also shows how

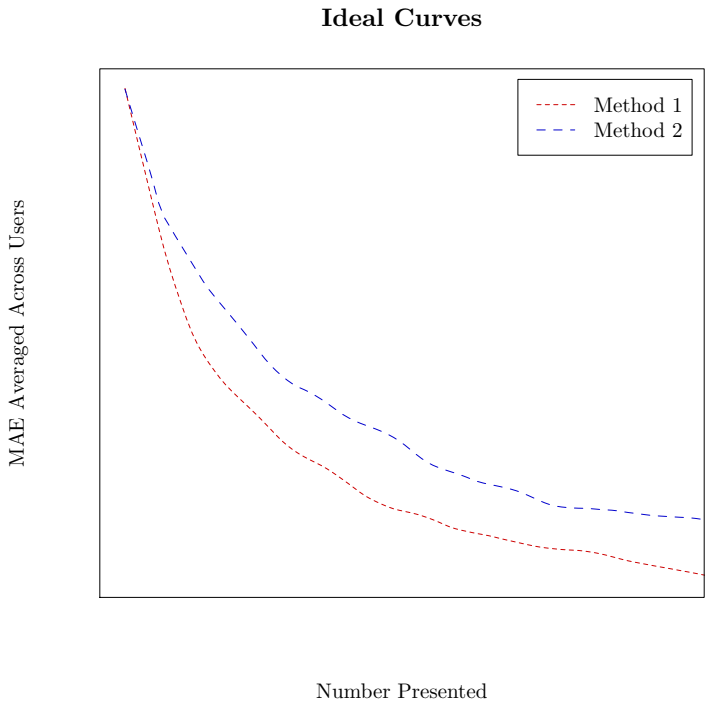

Figure 2.2: Ideal Curves

many more would be needed to be presented to get an equivalent prediction error by another method.

A similar method of measuring the performance of a cold-start system was also derived in Golbandi *et al.* (2010), although they are comparing the prediction error against a test set that contains ratings from all users, rather than the average of each user's prediction error.

The one downside is that this method of presenting the cold-start system performance fails to distinguish the difference between a system that allows a user to rate more items that are less informative as opposed to a system that would restrict the number the user can rate, but each item was more informative.

Figure 2.3 shows an example of how we might compare this information. The graph shows the rate at which the user is able to rate items. By scaling the axes to $(0, 1)$, we are able to calculate the area under curves (AUC), which in turn allows us to compare two sets of curves directly.

In this case, Method 1 has a larger AUC than Method 2. If these two methods produced identical error/presented graphs, then we could infer that Method 2 was able

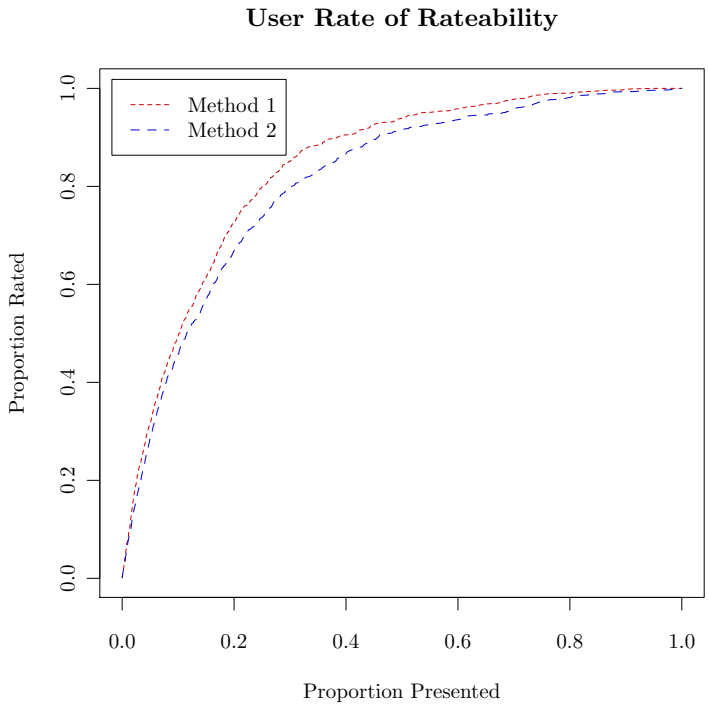

Figure 2.3: User Rate of Rateability

to find items that were of higher information content, whereas Method 1 was faster at finding items the user was able to rate.

Chapter 3

Predictions

The task of making predictions is central to any recommender system. By generating predictions the system can recommend those items that have the highest predicted rating. The task is to predict what rating a user would give to a movie. Additionally there might be the requirement to predict that rating for a particular time, as was the case for the Netflix Prize.

3.1 Simple Predictions

Simple predictions have the benefit that they are very quick and easy to calculate. They suffer, however, from the fact that their simplistic nature often does not reflect the way that people rate items. Simple predictors can be made using a constant value for every prediction, global average of all ratings, or average rating for either the user or movie being predicted.

3.2 Nearest Neighbour

The nearest neighbourhood method is based upon the idea that similarities in past rating patterns can be indicative of the future ratings for a person. There are two main variants of this method, user-based and item-based, both of which are discussed below.

3.2.1 User-Based

The user-based method uses the similarity between the user being predicted for, u, and all the other users of the system, and uses a weighted average of the neighbourhood, N, of users similar to u that have rated the movie being predicted for, m.

$$p_{um} = \frac{\sum_{v \in N} s_{uv} r_{vm}}{\sum_{v \in N} |s_{uv}|} \tag{3.1}$$

where s_{uv} is the similarity between users u and v, N is the neighbourhood which is typically defined to be the k most similar users to u that have rated m.

The similarity between two users is typically defined to be the Pearson correlation of the ratings of the users for items they have both rated:

$$s_{uv} = \frac{1}{|S|} \sum_{m \in S} \left(\frac{r_{um} - \bar{u}}{\sigma_u} \right) \left(\frac{r_{vm} - \bar{v}}{\sigma_v} \right) \tag{3.2}$$

where S is the set of movies that have been rated by both u and v, $\bar{u}$ is u's average rating of, and σ_u is the standard deviation of, the ratings u gave to movies in S.

3.2.2 Item-Based

The item-based method can be thought of as analogous to the user-based method. Rather than the relationship between users being employed, the relationships between the items the user has rated and the one being predicted for are used. The method uses the k most similar items to m in the dataset that the user u has rated and uses a weighted average across the ratings the user gave to those k items.

$$p_{um} = \frac{\sum_{n \in N} s_{mn} r_{un}}{\sum_{n \in N} |s_{mn}|} \tag{3.3}$$

where s_{mn} is the similarity between items m and n, and N is the neighbourhood which is typically defined to be the k most similar items to m that u has rated.

Often the cosine correlation between the items is used as the similarity metric between the items rather than the Pearson correlation:

$$s_{mn} = \frac{M \cdot N}{\|M\| \|N\|} \tag{3.4}$$

where M is a vector of ratings of m by people who have also rated n, and N the vector of ratings of n by people who have also rated m.

3.2.3 Common Modifications & Extensions

There are a number of modifications that have been applied to the traditional neighbourhood models in an attempt to improve the accuracy of such systems. A selection are outlined below, for a more in-depth analysis see Herlocker, Konstan, Borchers, and Riedl (1999).

Neighbourhood Selection

Which users to select as part of the neighbourhood is an important question. Traditionally selecting the top k most similar has been used, and k has had to be tuned on a per-dataset basis.

In Herlocker *et al.* (1999); Shardanand and Maes (1995) a selection of options were presented that required items in the neighbourhood to meet certain conditions before being considered, such as meeting a minimum similarity level.

Similarity Weighting

Another important part of the traditional neighbourhood model is the similarity score between two users or items. The fewer shared ratings that exist between two users or items, the less the similarity can be "trusted". To adjust for this each similarity score can be scaled by $\frac{\min(n,\alpha)}{\alpha}$ where n is the number of shared ratings, and α is a tuning parameter typically set to 50, the value found by empirical observation in Herlocker *et al.* (1999).

Rating Normalisation

Normalising the ratings that have been given allows a direct comparison between ratings given by different users, or for different items. Some basic normalisation methods include z-scores and deviation from mean. The z-score converts all the ratings for a user to have a mean of 0 and a standard deviation of 1. Deviation from mean attempts to take into account the ratings scales of users, each rating is treated as relative to the user's mean rather than the absolute value of the rating, it can be considered a relaxed version fo z-scores where the standard deviation can vary.

3.3 KorBell Model

Bell and Koren (2008) revisited some of the issues raised in Section 3.2.3. The main areas they targeted were the amount, and type, of normalisation that is carried out, and the use of the similarity weights in the prediction equation.

The KorBell method is currently one of the top performing single method models, performing about 3% better than Cinematch on RMSE of predictions made, and is the model that shall be used for the remainder of the thesis. The model consists of two phases: rating normalisation followed by a neighbourhood model.

3.3.1 Statistical Model/Normalisation

The statistical model is a form of rating normalisation that can also be used as a prediction method. The model is also known as global effects, and attempts to model aspects of rating behaviour such as a user's tendency to rate differently as time progresses. Predictions are made by adding each effect to the global average rating.

Effect
Movie
User
User$\times$Time(user)$^{\frac{1}{2}}$
User$\times$Time(movie)$^{\frac{1}{2}}$
Movie$\times$Time(movie)$^{\frac{1}{2}}$
Movie$\times$Time(user)$^{\frac{1}{2}}$
User$\times$Average(movie)
User$\times$Support(movie)$^{\frac{1}{2}}$
Movie$\times$Average(user)
Movie$\times$Support(user)$^{\frac{1}{2}}$

Table 3.1: Global Effects for the KorBell Statistical Model

The effects are listed in Table 3.1. The User and Movie effects combined are also known as double-centering, and they take into account the systematic tendencies for some users to rate, and some movies to be rated, higher or lower. For instance, if the user tends to rate highly we would want to predict higher ratings than for a user that rates lowly. The time variable is the number of days since the user's or item's first

rating, the support is the count of ratings that have been given by a user or to an item, and the average is the average rating for the user or item.

The User×Time(user)$^{\frac{1}{2}}$ effect takes into account whether a user gets stricter or more relaxed with their ratings as time goes on. The User×Time(movie)$^{\frac{1}{2}}$ effect takes into account changes in rating behaviour that depend on how soon the movie is rated by this user, for instance a user rating a movie on its release day might rate it higher than if they had waited before rating it. The Movie×Time(movie)$^{\frac{1}{2}}$ and Movie×Time(user)$^{\frac{1}{2}}$ effects are analogous.

The User×Average(movie) and User×Support(movie)$^{\frac{1}{2}}$ effects attempt to capture whether the user follows the crowd, or is contrary. For instance, we can imagine a user that dislikes the *Lord of the Rings* trilogy but likes cult movies. The Movie×Average (user) and Movie×Support(user)$^{\frac{1}{2}}$ attempts to capture the same with roles reversed, for instance we could imagine a movie that is liked by people that rate a lot of movies, but is not liked by people who have only rated a handful.

Each effect is estimated in turn on the residuals from removing the previous effects. The Movie effect is calculated by Equation 3.5:

$$\text{movie effect}_i = \sum_u (r_{ui} - \bar{g}) \tag{3.5}$$

where u is the set of users that has rated item i, r_{ui} is the rating given to the movie by user u, and $\bar{g}$ is the average rating in the dataset.

The User effect is calculated by Equation 3.6:

$$\text{user effect}_u = \sum_i r_{ui} \tag{3.6}$$

where i is the set of items that user u has rated, and r_{ui} is now the residual from removing the movie effect from the rating user u gave to item i.

Equation 3.7 shows how each of the interaction effects is estimated, where r_{ui} is the residual from removing the previous effects from the rating that user u gave item i, x_{ui} is the variable of interest for user u and item i.

$$\hat{\theta}_u = \frac{\sum_i r_{ui} (x_{ui} - \bar{x}_{ui})}{\sum_i (x_{ui} - \bar{x}_{ui})^2} \tag{3.7}$$

For instance when calculating the User×Average(movie) effect, x_{ui} would be the average rating for the movie i, and $\bar{x}_{ui}$ is the average of the average ratings for the movies that the user has seen.

Furthermore, each of the effects is shrunk as in Equation 3.8 so as to minimize the impact of effects calculated on small numbers of observations.

$$\frac{n_u \hat{\theta}_u}{n_u + \alpha} \tag{3.8}$$

The effect is scaled by α, a tuning parameter and n_u, the number of ratings that the user u has given, or that item u has received, depending on the effect being calculated. The α values were made available on the Netflix Prize forum,[1] and have been replicated in Table 3.2.

Effect	α
Movie	25
User	7
User×Time(user)$^{\frac{1}{2}}$	550
User×Time(movie)$^{\frac{1}{2}}$	150
Movie×Time(movie)$^{\frac{1}{2}}$	4000
Movie×Time(user)$^{\frac{1}{2}}$	500
User×Average(movie)	90
User×Support(movie)$^{\frac{1}{2}}$	90
Movie×Average(user)	50
Movie×Support(user)$^{\frac{1}{2}}$	50

Table 3.2: Alpha Values for Global Effects Estimation

An alternative model for the calculating the global effects that resulted in predictions with higher accuracy was presented in Potter (2008), and personal experimentation has shown that an interaction effect between the user and release year of the movie would improve accuracy. However, we shall only consider the original implementation. In practice, for the cold-start problem we do not use the effects that contain the user time as we are prompting the user for the information. We also do not consider the effects that consider the time of the item ratings as this makes the predictions worse.

[1] `http://www.netflixprize.com/community/viewtopic.php?pid=5563#5563`

3.3.2 Neighbourhood Model

The KorBell neighbourhood model is a variant of the traditional item-based neighbourhood model. Rather than directly using the similarity between the two items as the weight, as discussed in Section 3.2.2, it uses the relationships between the neighbourhood items and the item being predicted for to learn an optimum set of weights. This addresses the issue of using the similarities between items directly in the prediction as described in Section 3.2.3.

For example, imagine that all three of the *Lord of the Rings* trilogy are in the neighbourhood of, and have a high similarity with, the item being predicted for. These ratings are then essentially triple-weighted as the three movies in the trilogy have a high correlation among themselves.

The KorBell model wants to find a set of weights w that allow predictions of the form:

$$p_{ui} \leftarrow \sum_{j} w_{ij} r_{uj} \tag{3.9}$$

where j is in the set of elements that make up the neighbourhood of i.

To start, the Bell and Koren (2008) first considered a hypothetical dense case where everyone but the user being predicted for has rated the item being predicted for and all its neighbours. They determined that the optimum weights could be calculated by solving the least squares problem:

$$\min_{w} \sum_{v \neq u} \left(r_{vi} - \sum_{j} w_{ij} r_{vj} \right)^2 \tag{3.10}$$

They identified that the optimal weights in this situation are given by:

$$Aw = b \tag{3.11}$$

where A is a $K \times K$ matrix:

$$A_{jk} = \sum_{v \neq u} r_{vj} r_{vk} \tag{3.12}$$

and b a vector of size K:

$$b_j = \sum_{v \neq u} r_{vj} r_{vi} \tag{3.13}$$

where r_{vj} is the rating, or residual after removing global effects, of user v's rating of item j.

In order to calculate the true values for these weights would require using only those users for which complete information exists, ignoring a large proportion of information about the relationships between items that the user had provided ratings for. Instead an estimate of A is given by $\bar{A}$, and b by $\bar{b}$:

$$\bar{A}_{jk} = \frac{\sum_{v \in U(j,k)} r_{vj} r_{vk}}{|U(j,k)|} \tag{3.14}$$

$$\bar{b}_j = \frac{\sum_{v \in U(i,j)} r_{vi} r_{vj}}{|U(i,j)|} \tag{3.15}$$

where $U(i,j)$ is the set of users that have rated both items i and j.

As discussed in Section 3.3.1 the effects of estimations based upon only a few observations (such as only 1 person having seen both i and j) decrease the effectiveness of the system. To alleviate this, $\bar{A}$ and $\bar{b}$ are in turn shrunk towards a common mean avg, taken to be the average of all the values in $\bar{A}$. This shrinking is controlled by a parameter β, typically set to 500.

$$\hat{A}_{jk} = \frac{|U(j,k)| \cdot \bar{A}_{jk} + \beta \cdot avg}{|U(j,k)| + \beta} \tag{3.16}$$

$$\hat{b}_j = \frac{|U(i,j)| \cdot \bar{b}_j + \beta \cdot avg}{|U(i,j)| + \beta} \tag{3.17}$$

The weights are then calculated by solving the slightly modified version of Equation 3.11 for w:

$$\hat{A}w = \hat{b} \tag{3.18}$$

which is the substituted into the original prediction equation (Equation 3.9).

Bell and Koren (2008) found that better weights, resulting in higher prediction accuracy, were calculated when restricting the weights to be non-negative. This method, called `NonNegativeQuadraticOpt`, for solving w was presented in Bell and Koren (2008), with added corrections and suggestions following discussions on the Netflix Prize forum.[2] The value of ϵ is set to 2.5E-8.

The items that are selected to form the neighbourhood are taken as being those that have the highest, or highest absolute, Pearson Correlation (Equation 3.2) on the residuals after removing global effects, with the item being predicted for that the user being predicted for has been able to rate.

[2] `http://netflixprize.com/community/viewtopic.php?id=837`

Algorithm 3.1 Fixed `NonNegativeQuadraticOpt`

// Minimize $w^T \hat{A} w - 2\hat{b}^T x$ *such that* $w \geq 0$

$w \leftarrow$ Small non-negative value

$r \leftarrow \hat{b} - \hat{A}w$ *// the residual*

// Find those variables that are pinned due to non-negativity

// constraints and set respective residuals to 0

while $\|r\| > \epsilon$ **do**

  for i $= 1$ **to** k **do**

    if $w_i = 0$ **and** $r_i < 0$ **then**

      $r_i \leftarrow 0$

    end if

  end for

  $\alpha \leftarrow \frac{r^T r}{r^T \hat{A} r}$ *// Max step size*

  // Adjust step sizes to prevent negative values

  for $i = 1$ **to** k **do**

    if $r_i < 0$ **then**

      $\alpha \leftarrow \min\left(\text{abs}\left(\alpha\right), \text{abs}\left(w_i / r_i\right) \cdot \left(\alpha / \text{abs}\left(\alpha\right)\right)\right)$

    end if

  end for

  $w \leftarrow w + \alpha r$

  $r \leftarrow \hat{b} - \hat{A}w$ *// the residual*

end while

return w

The KorBell model has been further extended and improved upon by the original authors (Koren, 2010a,b).

3.4 Other Prediction Methods

Numerous methods have been applied to collaborative filtering scenarios. What follows is a brief description of some of them.

3.4.1 Matrix Factorization

Matrix factorization is a relatively new method for the purposes of recommendation systems. The ratings given by users can be thought of as a matrix with users as rows and items as columns, or vice versa. This ratings matrix can then be factorized, or decomposed, into smaller vectors and matrices that can then be multiplied back together to generate recommendations.

A variant of Singular Value Decomposition (SVD) was used to good effect in the early stages of the Netflix Prize (`http://sifter.org/~simon/journal/20061211.html`), and has been developed further by Paterek (2007).

3.4.2 Content-Based

The content that is associated with items has been leveraged to provided recommendations rather than relying on explicit ratings from users (Balabanović and Shoham, 1997). Additionally attempts to hybridise these content-based systems with ratings-based systems have been attempted (Jung *et al.*, 2004; Basilico and Hofmann, 2004).

In Pilászy and Tikk (2009) the authors showed that more accurate predictions can be made when the user has provided minimal ratings than when the system uses the metadata of the items to generate predictions. The prediction method that they used was a variant of matrix factorization outlined above.

3.4.3 Belief Distribution

In McLaughlin and Herlocker (2004) an alternative prediction model called the Belief Distribution Algorithm was developed which attempted to correct flaws with traditional

nearest neighbour models. In this model each rating provided by the user is used as noisy evidence of the user's true rating which creates a set of probabilities attached to each possible rating.

These belief distributions are then combined across the set of nearest neighbours for the user to provide a belief distribution for them. From this a prediction of the rating is selected from that rating which has the highest probability in the belief distribution. This method of generating predictions performs poorly when measured with the prediction metrics described in Section 2.1, but highly in metrics over top-n lists described in Section 2.2.

The designers of this algorithm point to several aspects that make it desirable for recommendation tasks, such as its ability to display the confidence in the predictions that it makes.

3.4.4 Other

Boltzmann machines, a form of stochastic recurrent neural network, have also been used for prediction in collaborative filtering systems (Gunawardana, 2008; Salakhutdinov, Mnih, and Hinton, 2007).

Association rules have also been used and attempt to find a set of rules that describe relations between items, and then these rules can be applied to make predictions (Huang *et al.*, 2004).

3.4.5 Ensemble Methods

Recent literature has expanded on the use of ensemble learning, that is, taking a number of weak predictors and combining the predictions that they make for an overall better prediction (Bao, Bergman, and Thompson, 2009)

Jahrer, Töscher, and Legenstein (2010) provides an overview of several ensemble, or blending, techniques that can be applied, including linear regression, neural networks, bagged gradient boosted decision trees, kernel ridge regression, and k-nearest neighbours. They even promote the use of blending the output from some of these methods to further increase the accuracy of predictions.

For example, the winning submission of the Netflix Prize was a model that combined over 200 different individual predictors (Toscher, Jahrer, and Bell, 2009; Piotte and

Chabbert, 2009; Koren, 2009). It should be noted that not all 200 predictors were required to win the Netflix Prize, a subset of 18 predictors was found suitable.

Chapter 4

Methods

When a new user enters the system, the system needs to present to that user a selection of items for the user to rate before the system is able to make good recommendations. Now that we have a way of measuring how well our system is performing, and a way of making predictions, we can discuss methods by which the system can determine the next item to present to the user for rating.

4.1 Non-Personalised

The following methods present movies in the same order to every new user that enters the system. They have the advantage that the order only needs to be calculated once, however, they may not elicit much information.

4.1.1 Popularity

The Popularity method presents movies ordered by the number of ratings that they have been given. It is equivalent to ordering by the probability that a user has seen the movie.

As can be seen in Figure 4.1 the Popularity method performs substantially better than presenting movies at random.

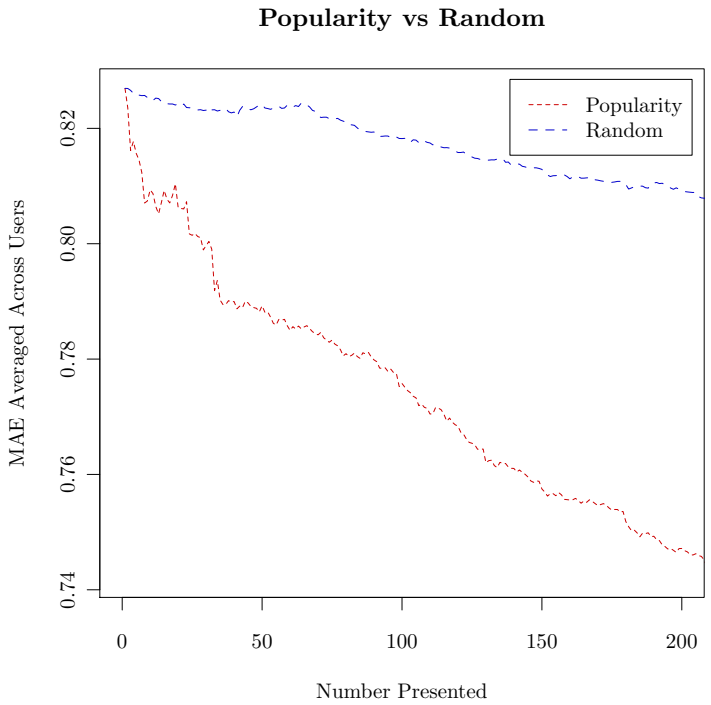

Figure 4.1: Popularity vs Random

4.1.2 Entropy

A drawback of the Popularity method is that it ignores any potential information that the ratings of the movie might contain. That is, we can imagine that certain movies could yield more information about a user's likes than others. Entropy was proposed as a possible method of calculating the amount of information embedded in a movie's ratings by Kohrs and Merialdo (2001) and was re-presented in Rashid *et al.* (2002).

The entropy for a movie, m, is defined as:

$$entropy_m = -\sum_r P(m_r) \log(P(m_r)) \tag{4.1}$$

where $P(m_r)$ is the probability that the movie was given a rating of r, which is in the range 1–5 for the Netflix dataset.

As we can see in Figure 4.2 the Entropy method performs poorly when compared to the Popularity method, and indeed performs worse than randomly presenting items. A quick investigation shows that this is due to movies that have small numbers, but flatter distribution, of ratings dominating the beginning of the entropy list.

For example, in Figure 4.3 we see that *"In A Year With 13 Moons"* has 118 ratings

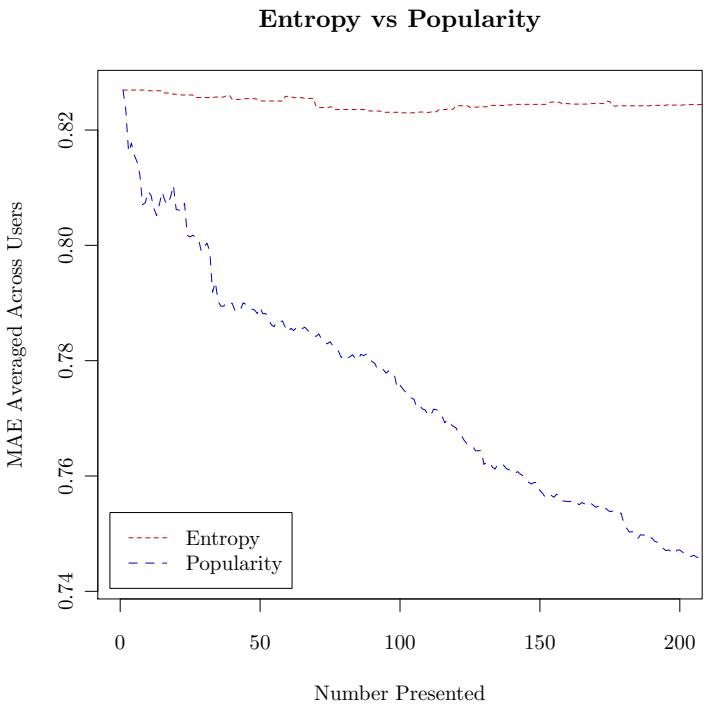

Figure 4.2: Entropy vs Popularity

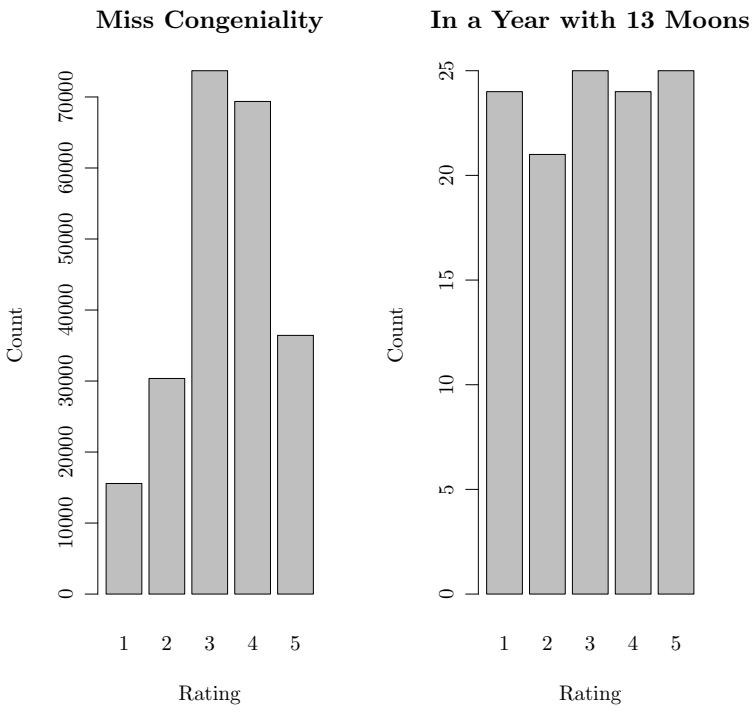

Figure 4.3: Entropy Examples

and an entropy of 1.607, while *"Miss Congeniality"* has 225,423 ratings and an entropy of 1.477. Clearly the probability that a user is able to rate the former is much lower than the latter.

4.1.3 Entropy0

Entropy0 was introduced in Rashid *et al.* (2002) as an attempt to fix the problem of movies with small numbers of ratings dominating. This was done by including the number of non-ratings (treated as rating of 0) and using a weighted entropy scheme.

The entropy0 of a movie, m, is now defined as being:

$$entropy0_m = -\frac{1}{\sum_r w_r} \sum_r P(m_r)\, w_r \log(P(m_r)) \tag{4.2}$$

where w are the weights to be used with $w_0 = 0.5$ and $w_{1\to5} = 1$. It is worth noting that these weights are arbitrary, although these weights provided the best results for the original experimentation.

Other methods of balancing the number of ratings with entropy, such as $\log(ratings) * entropy$ where made in Rashid *et al.* (2002), although the Entropy0 method was the best performing.

Now we see in Figure 4.4 that with the inclusion of the non-ratings *"Miss Congeniality"* has a flatter rating distribution, so it has a higher entropy0. This allows movies that have a higher chance of being seen to be near the top of the list. In our examples, *"Miss Congeniality"* now has an entropy0 of 0.234, and *"In A Year With 13 Moons"* 0.0005.

As shown in Figure 4.5 the Entropy0 method fixes the problems with the Entropy method, and outperforms the Popularity method, if only slightly. This suggests that indeed some movies contain more information pertinent to predicting ratings than others.

4.1.4 Greedy

Knowing that the Entropy0 method was able to extract some more information from the user, we may wish to see what the best possible scenario is. For the Greedy method we choose the next item from those that the user is able to rate such that the prediction error for their test set is minimised.

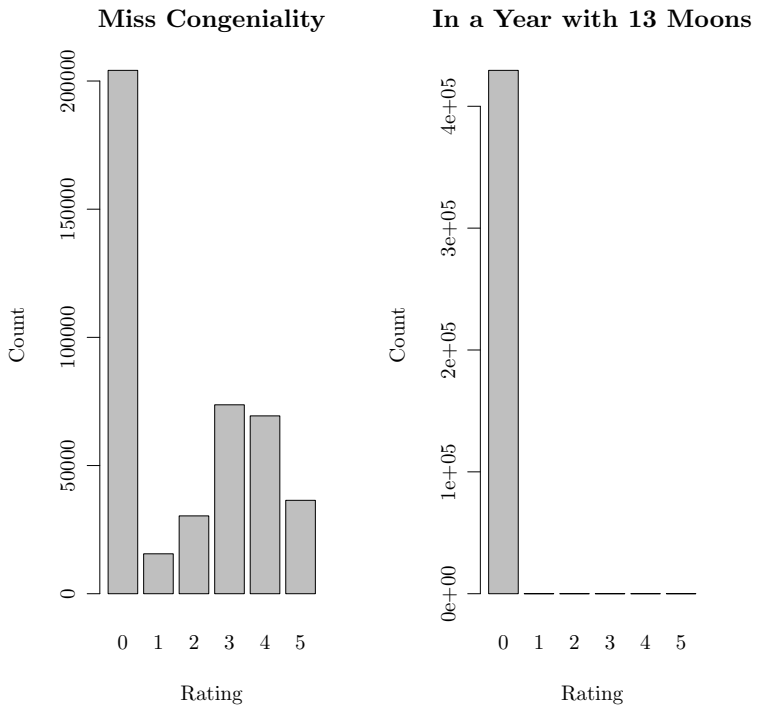

Figure 4.4: Entropy0 Examples

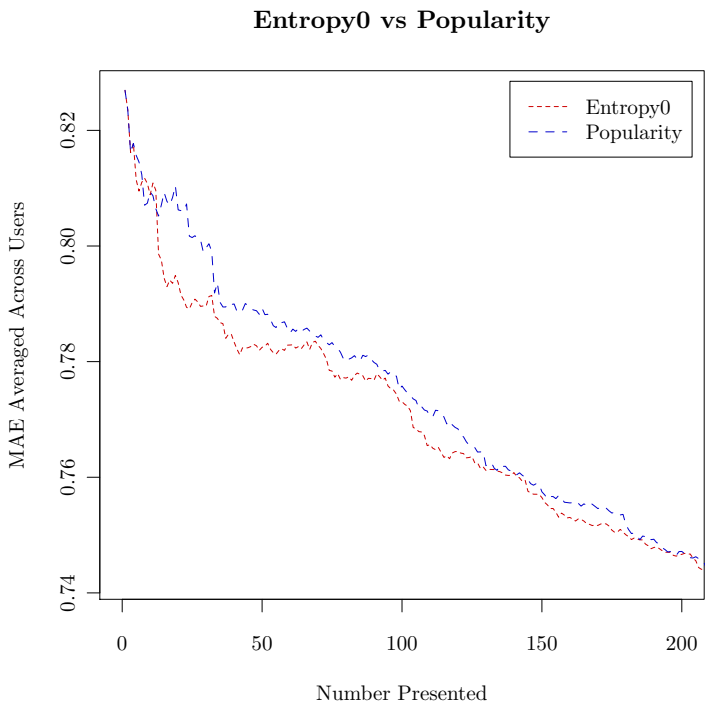

Figure 4.5: Entropy0 vs Popularity

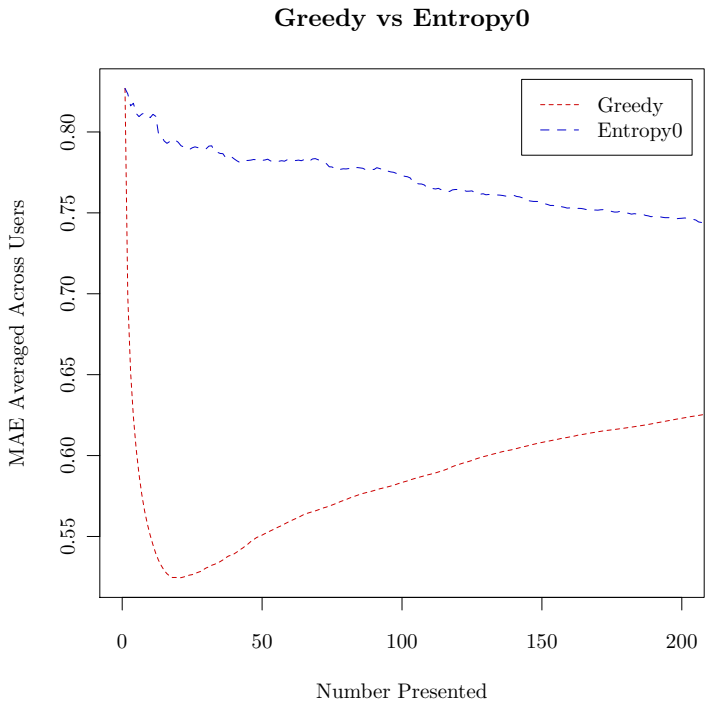

Figure 4.6: Greedy vs Entropy0

Clearly this method could not be used in practice as it requires knowing not only what each person is able to rate, but also what they rated it. However, using it as a baseline, Figure 4.6 shows that there is clear room for improvement. By examining the items that regularly occur at the top of the greedy lists we may gain some idea as to what makes a good movie to know the rating for.

Figure 4.7 shows the rating distribution of the movies that appear most commonly in the top 10 of greedy lists. The top row shows the distribution of the ratings given by the people in whose greedy list it appears in the top 10, while the right shows the ratings distribution of everyone that was able to rate that movie. From this we can see that the ratings given to items that end up on the top of greedy lists is different from the population. For instance, the ratings of *"Miss Congeniality"* show higher proportions of high (5) and low (1) ratings compared to the overall population. On the other hand, with items that were generally liked, such as *"The Lord of the Rings: The Return of the King"*, we see significantly fewer high ratings.

This leads us to suspect that while knowing the rating distribution of an item is important, it is more important to know what the user rated the item. That is, we

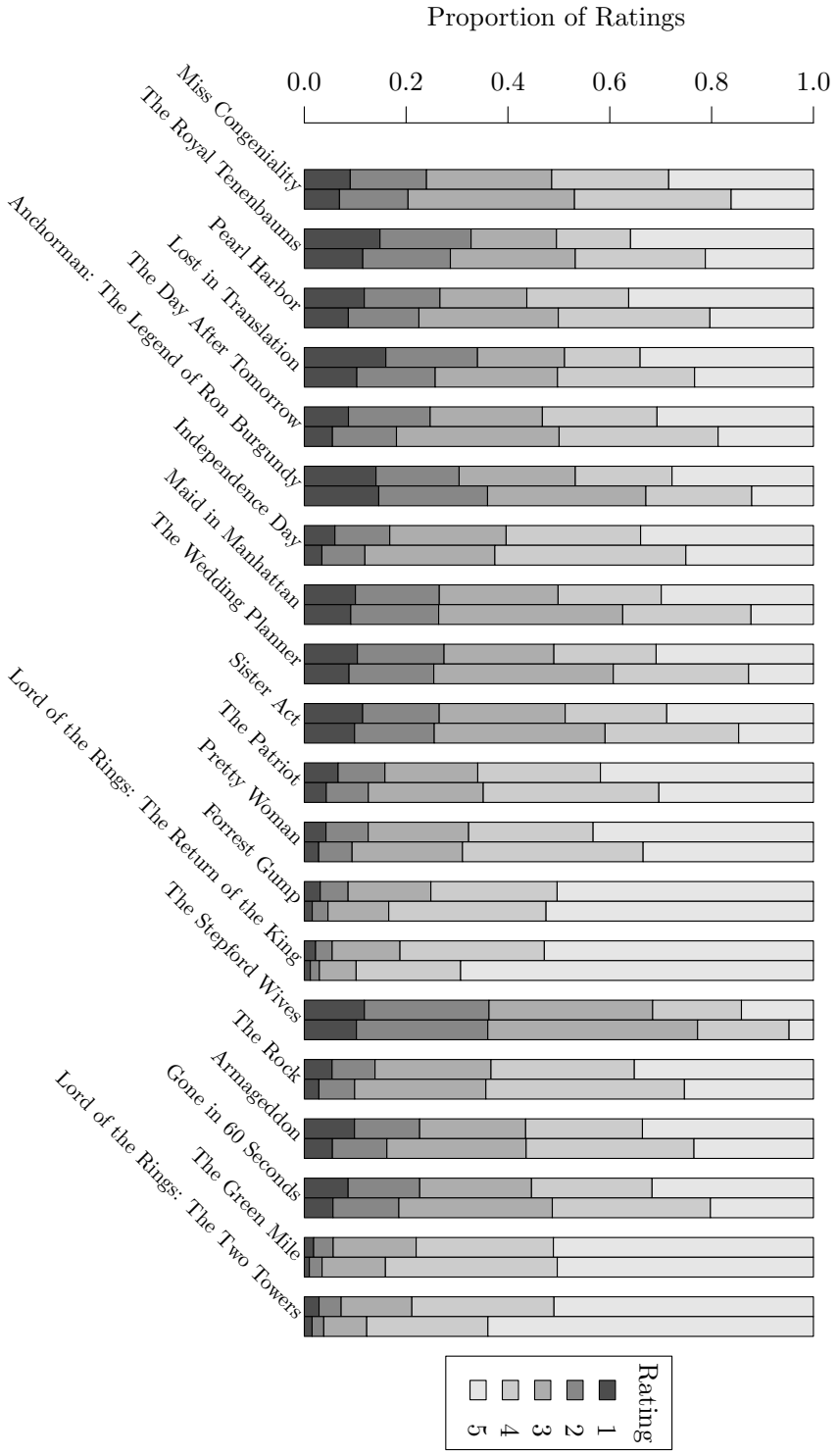

Figure 4.7: Rating Distribution of Most Frequent Greedy Movies, Comparing Population with Those Users For Whom it Appears in Top 10 of Greedy List

get more information from a low rating of a generally liked film than we do about a middle rating of an item with no general consensus.

Another interesting result is represented in Figure 4.6 by the decrease of prediction accuracy after having presented about 20 items to the user. This shows that it is of more use to have the ratings from the user for a small set of key items rather than a larger number of ratings over more general items.

4.1.5 Other People's Greedy

For each user we now have a list of movies ordered by their ability to minimise prediction error on a hold out test set, which was selected in a greedy fashion. We clearly cannot use these lists as it requires knowing what items the user rates, and what rating they gave those items. We can, however, use the information contained in other people's lists.

We rank the movies in descending order of the number of times they appear in the top n of other people's greedy lists. The items are then presented in this order to the user. The advantage of such a method is that for an item to be on top of the list it has to both be an item that reduces error a large amount (to appear in any person's greedy list) and an item that is popular (to appear in many people's greedy lists). The greedy lists are constructed on the data for everyone in the dataset not only those users in the sample.

One of the obvious questions is what should the value of n be set to. As can be shown in Figure 4.8 the effects of changing n are small. It should be noted that as n approaches the number of items in the dataset, the lists generated by this method will approach that of the Popularity method (every item a person can rate will be included in their greedy list).

4.1.6 Switching Other People's Greedy

From Figure 4.8 we can see that a value of $n = 5$ performs better in the initial stages, while after presenting 20 items to the user a value of $n = 1$ performs better. The question arises as to whether changing these values at that point will provide a benefit.

The results of such a switch are shown in Figure 4.9 where the swap is initiated after presenting 15 items, despite the crossover point being after 20 items have been

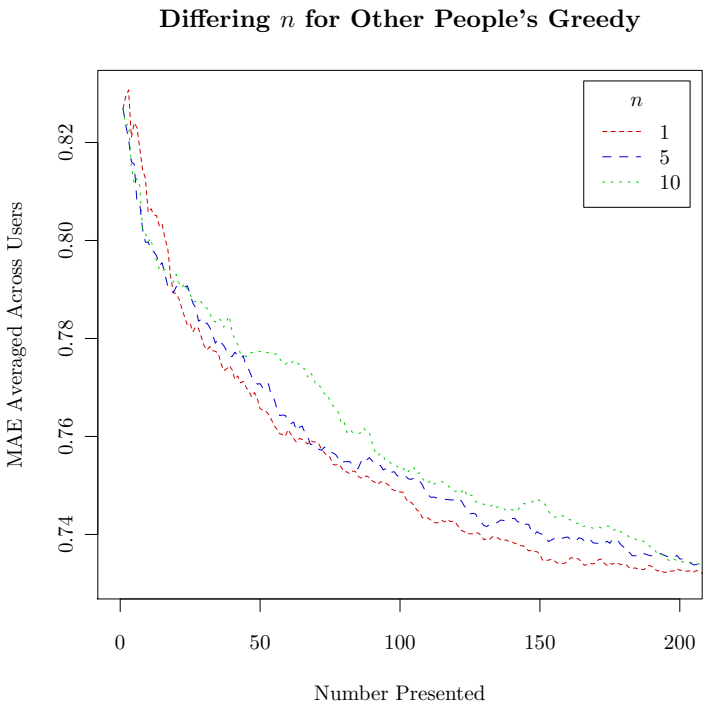

Figure 4.8: Differing n for Other People's Greedy

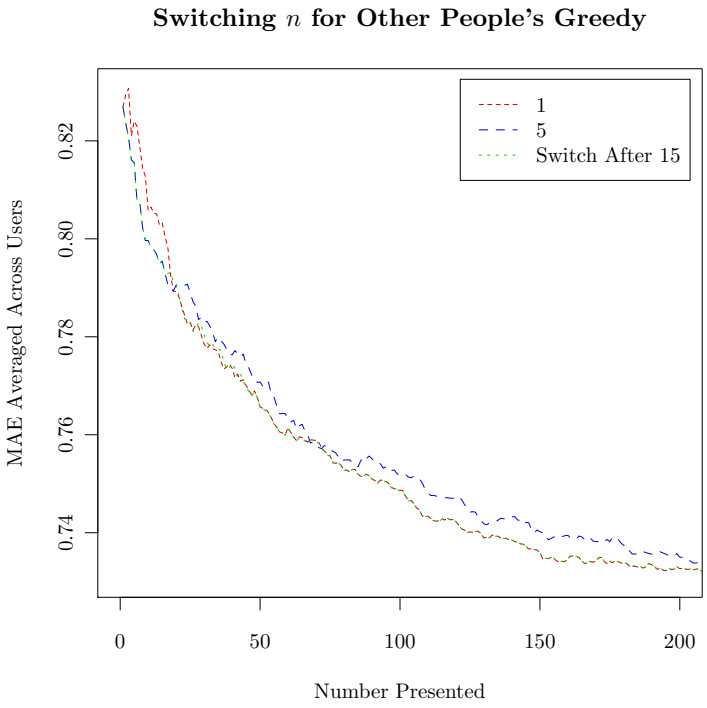

Figure 4.9: Changing n in Other People's Greedy

presented. The resulting method performs slightly better than either of the combined parts.

As we see in Figure 4.10 the Switching Other People's Greedy method improves significantly over the Entropy0 method. From this point references to Other People's Greedy refer to this variant where the value of n is changed from 5 to 1 after presenting 15 items.

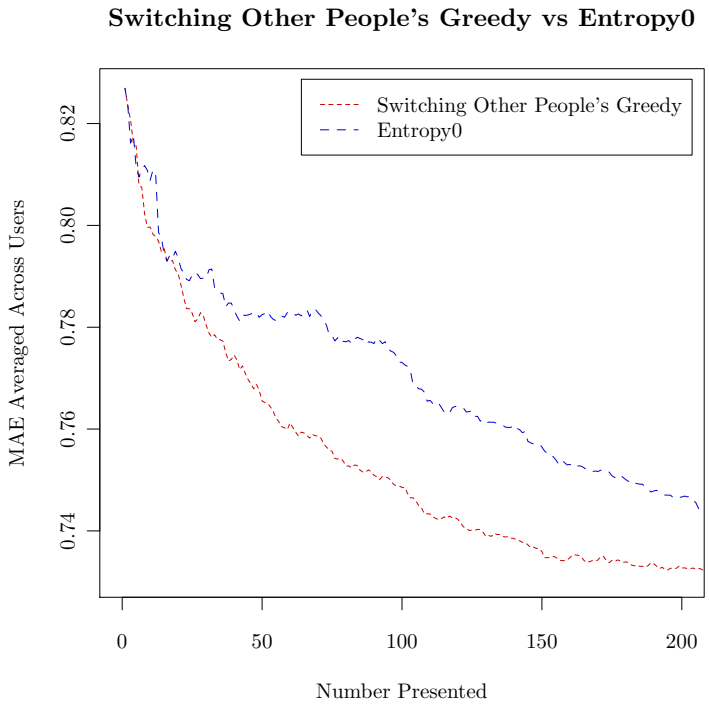

Figure 4.10: Switching Other People's Greedy vs Entropy0

A similar approach was also taken by Golbandi *et al.* (2010), although their method of selecting the next item to present differs significantly. They chose to select the next item such that the prediction error (calculated using RMSE), was minimised on the training set. They used the original training and testing sets that were constructed as part of the Netflix Prize (Bennett and Lanning, 2007).

Interestingly the list of the first 15 items that their method generates is not too dissimilar from the first 15 that the Switching Other People's Greedy method generates despite the different calculation methods, in particular the top 4 items are almost identical. Table 4.1 shows the items generated by their method, and gives the position of that item in the list generated by our method. The first item to appear in our

generated list that does not appear in the reported list occurs in position 6 on the list and is *"Anchorman: The Legend of Ron Burgundy"*.

Golbandi *et al.* (2010)		Position in Other
Position	Title	People's Greedy List
1	The Royal Tenenbaums	2
2	Miss Congeniality	1
3	Pearl Harbor	3
4	Lost in Translation	4
5	Sweet Home Alabama	26
6	Pulp Fiction	27
7	The Day After Tomorrow	5
8	Independence Day	7
9	Maid in Manhatten	8
10	Pretty Woman	12
11	Gone in 60 Seconds	18
12	Being John Malkovic	132
13	Mr. Deeds	35
14	Kill Bill: Vol. 1	32
15	How to Lose a Guy in 10 Days	25

Table 4.1: Position of Top 15 Items Generated by Golbandi *et al.* (2010) in List Generated by Other People's Greedy Method

We can speculate that this method is finding items that polarize reactions, certainly the first items follow this pattern. The presence of items such as *"Lord of the Rings: Return of the King"*, however, leads us to suspect that it is also choosing items for which the user may give a response that is particularly contrary to popular opinion.

4.2 Personalised

The methods that have been presented above are non-personalised, that is, they ignore the responses that the user has given to items already presented. We saw in Section 4.1.4 that the ratings distribution for items is different when comparing the

population as a whole and only those users for whom the item was in the top 10 of the greedy list.

Intuitively it seems that we should use the responses given by the user to determine which item should be presented to them next.

4.2.1 Naïve Bayes

By knowing whether a user is able to, or unable to, rate a movie we can work out the naïve bayes probability that the user is able to rate the other movies. This calculated probability can be used when choosing items to present. The first item presented is the item which has the highest number of ratings and therefore the highest probability of being able to be rated. Thereafter, this is in essence a personalised version of the Popularity method.

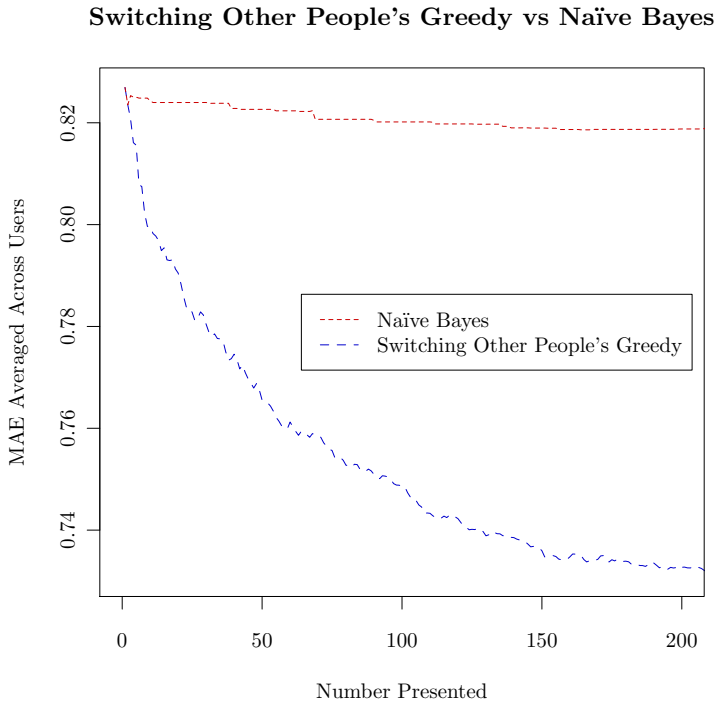

Figure 4.11: Other People's Greedy vs. Naïve Bayes

As can be seen in Figure 4.11 the Naïve Bayes method performs poorly when compared to the Other People's Greedy method, indeed it performs worse than random.

4.2.2 Perturbed Other People's Greedy

Rather than using the naïve bayes probability to construct the whole list, we could instead use it to perturb the list we have already generated by the Other People's Greedy method. To perturb the list we choose the item which has the highest probability of being able to be rated from the next n selected by the Other People's Greedy method.

This attempts to balance the benefits of the user being able to rate the item, and the amount by which the item reduced prediction error. By only selecting from the next n items on the list we still have items that will reduce prediction error, but we are also selecting those items in a better order for being able to be rated.

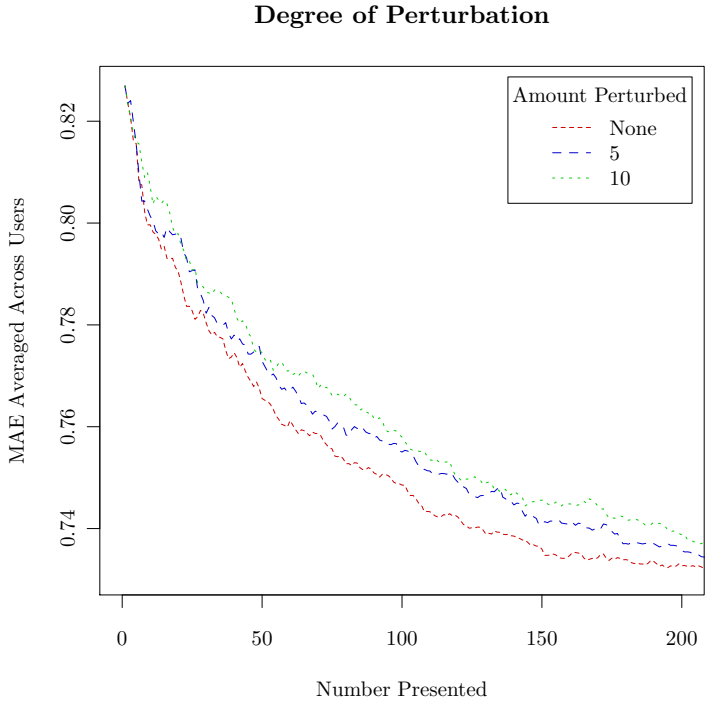

Figure 4.12: Amount of Perturbation

As we can see in Figure 4.12 all variations of amount of perturbation do not improve upon the Other People's Greedy method. As the amount of perturbation approaches the number of items in the dataset, the method approaches the Naïve Bayes method, albeit with a different starting item, and if the number to perturb is 0 or 1 then this is equivalent to the Other People's Greedy method.

4.2.3 Greedy Tree

Rather than perturbing the list of items that has been constructed with the Other People's Greedy method we can instead change which users we consider when we select which item to present next. One way of changing which users we are considering is to only consider those who rate similarly to the current user. Items are ordered by the number of times they appear in the top n of greedy lists of the people that we are still considering.

This method builds a tree where each node is the movie that should be presented and each edge shows which path to follow depending on the rating that was given to the last item presented. For example, Figure 4.13 shows the first 4 layers of the tree that is generated when using binary splits based upon whether or not the user has been able to rate the movie. The first movie that is presented is *"Miss Congeniality"*, and if the user was able to rate that, then they are next presented with *"Pearl Harbor"* to rate, otherwise, they are presented with *"The Royal Tenenbaums"* and so on. In the example generated tree we see that regardless of the response to *"Lost in Translation"* the user is always presented with the same movie, *"Punch-Drunk Love"*, next.

Branching Factor

In this method, the question of how we judge whether one user is similar to another arises. Above we used a simple decision based on whether the user could or could not rate the same items. We could determine the likeness by each individual rating, or any grouping of the possible ratings, in the case of the Netflix dataset, 1–5.

The notation we use for the splits that we are creating shows each of the possible splits grouped together in sets, with a 0 representing the user not being able to rate the item presented.

The splits that we consider are: whether the user is able to rate the item or not, represented as {0, 12345}; if the user was able to rate the items, whether they rated it high or low, {0, 123, 45}; if the user provided an indifferent rating for the item, {0, 12, 3, 45}; and on every possible rating, {0, 1, 2, 3, 4, 5}.

Figure 4.14 shows the performance of the different splits described above. We see that the best performance is gained through creating splits on whether the user was able to rate the previous movie or not.

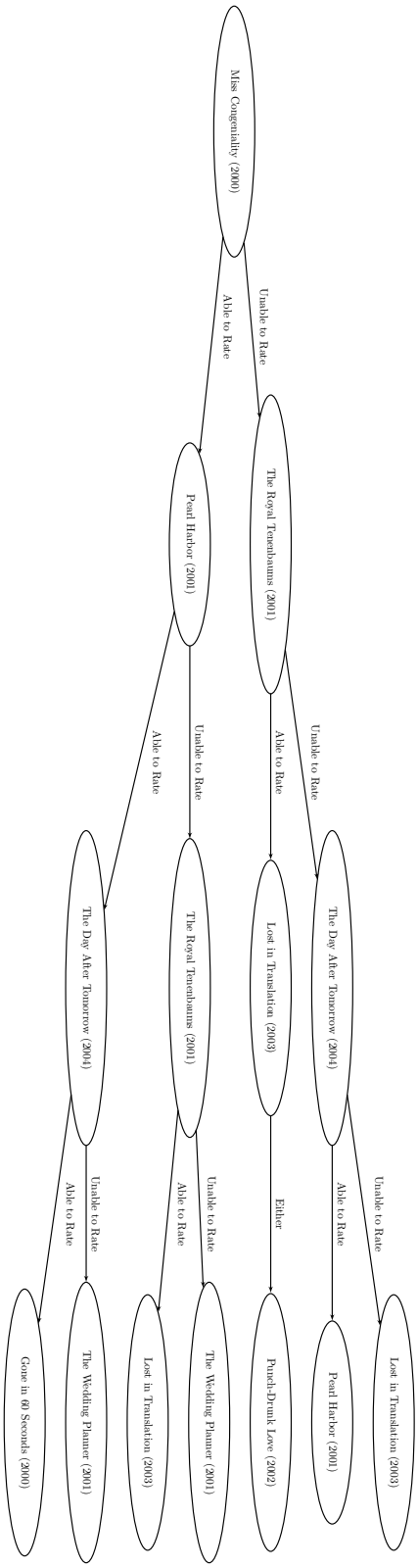

Figure 4.13: Greedy Tree Example

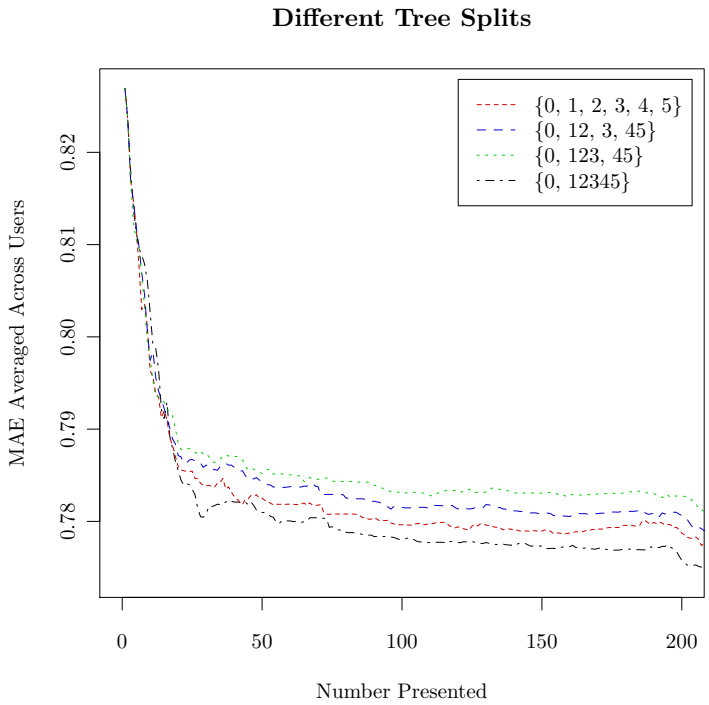

Figure 4.14: Different Greedy Tree Splits

Interestingly the performance gain that was noted by switching the value of n for the Switching Greedy Other People's method presented in Section 4.1.6 disappears when constructing the list using the Greedy Tree method.

Figure 4.15 shows the results of using the Greedy Tree method with the have/have not seen split compared to the Switching Other People's Greedy method. The method performs well initially but tails off after around 25 items have been presented.

Node Size

As mentioned earlier, the method has initial promise, but eventually falters. This is due to the fact that the number of people that are still being considered when deciding which item to present next drops off dramatically. For instance, it takes only 19 items to be presented before there are more possible paths generated than there are users in the dataset.

By definition there then have to be paths down which no users are still being considered, which is potentially the path that the current user is following. This problem becomes exacerbated when using the splits that have higher degree.

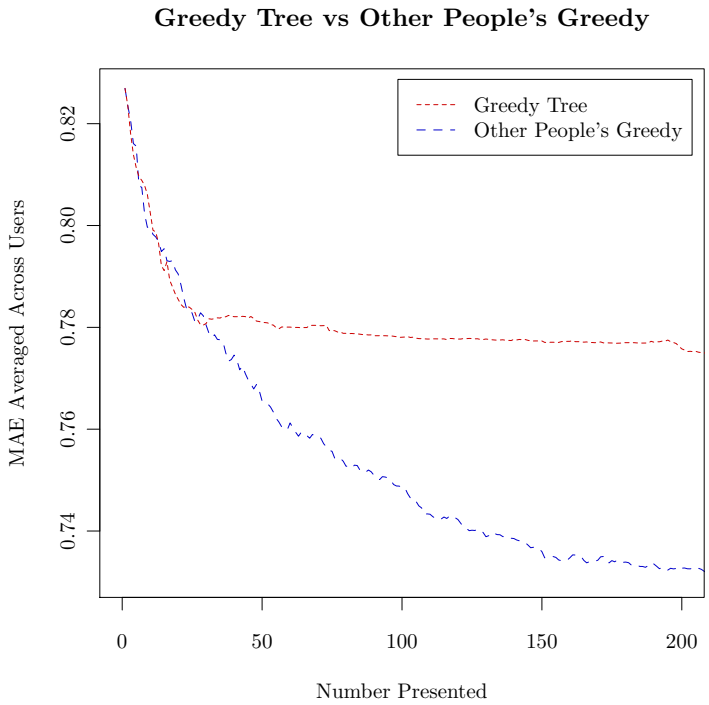

Figure 4.15: Greedy Tree vs Switching Other People's Greedy

There are a number of methods we can use to try and negate the impact that this problem causes:

Fixed restarting: Reconsider everyone again after every n items that have been presented.

Proportional restarting: Reconsider everyone again when the number of people that are currently being considered drops below a certain threshold.

History length: Only consider those users that are similar to the user for the past n decisions in the tree created.

We refer to the greedy tree that is created with none of these techniques applied as the Plain Greedy Tree.

Each path down the tree will be followed by different numbers of users. For this reason the **fixed restarting** method has the flaw that it may reconsider users too soon down the more popular paths, while reconsidering users too late down the less popular paths. When the value of n is set to 1 then this method is equivalent to Other People's

Greedy, and when set to the number of items in the dataset it is equivalent to Plain Greedy Tree.

The **proportional restarting** method fixes this flaw by only reconsidering users when the number still being considered drops below a certain proportion, $\frac{1}{n}$, of the users in the dataset. This means that if a path contains a large number of users, the others will not be reconsidered until later, whereas on a path that contains few users the others will be reconsidered sooner. When n is set to the number of users in the dataset, then this method is equivalent to Other People's Greedy, likewise, when n is set to 0 then it is equivalent to Plain Greedy Tree.

The **history length** method only considers the last n decisions that were made by the user when determining whether other users rated like them. This has the advantage over the proportional restarting method in that only the users that were filtered out by the $n^{th} - 1$ most recent decision will potentially be reconsidered. Each user that can now be reconsidered is required to still pass through the remaining filters before they are finally considered again. The length of the history is an important factor, with a history length equal to the number of items, then this method is equivalent to Plain Greedy Tree, likewise, if the history length is set to 0 then this method is equivalent to the Other People's Greedy method. Essentially the history length method maintains a tree that has a depth of at most n.

Interplay

There is some obvious interplay that exists between the branching factor and the node size alleviation method. The higher branching factor the sooner the alleviation methods need to be activated.

Figure 4.16 shows a combination of different values for the fixed restart method and branching factors. Figure 4.17 shows the same for proportional restarting and Figure 4.18 for history lengths. The respective values for n are shown in the title of each plot.

Figure 4.19 shows the comparison of the best of each alleviation method and branching factor. These are: fixed restart every 15 items with splits of {0, 12, 3, 45}; proportional restart when less than $\frac{1}{15}$ of the users are still considered and splits of {0, 12345}; and a history length of 10 and splits of {0, 1, 2, 3, 4, 5}.

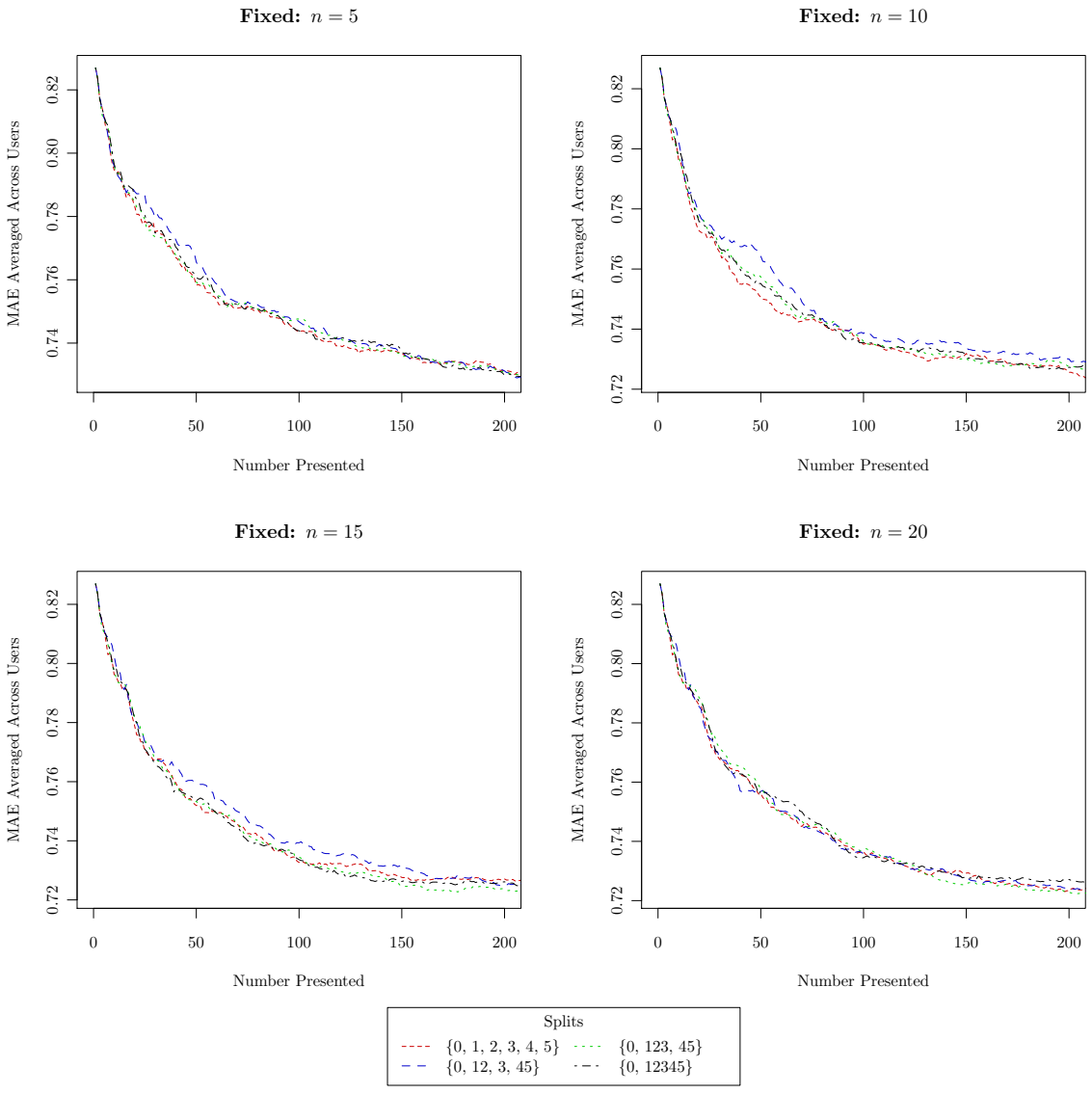

Figure 4.16: Different Splits and Fixed Restarting Positions

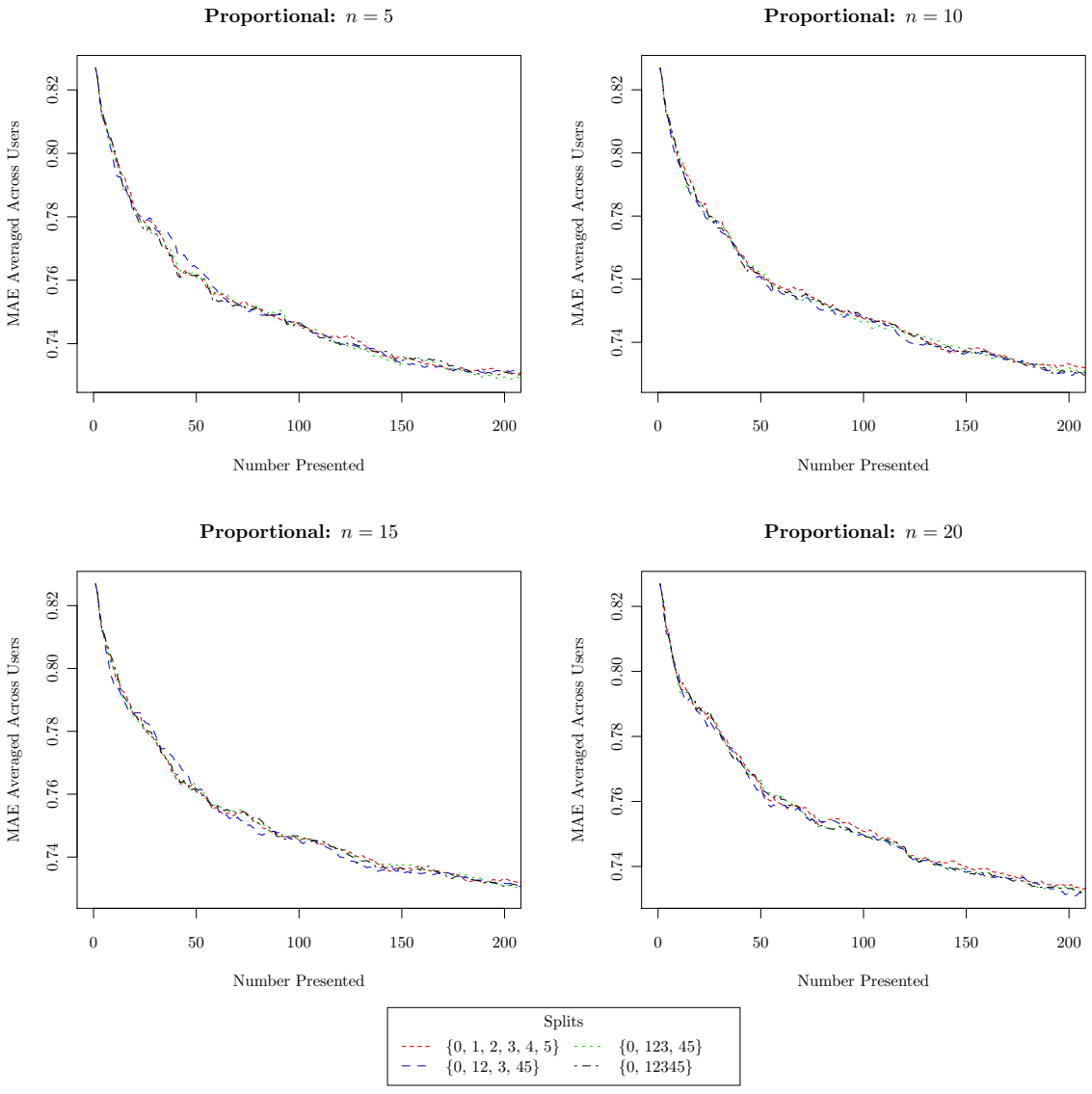

Figure 4.17: Different Splits and Proportional Restarts

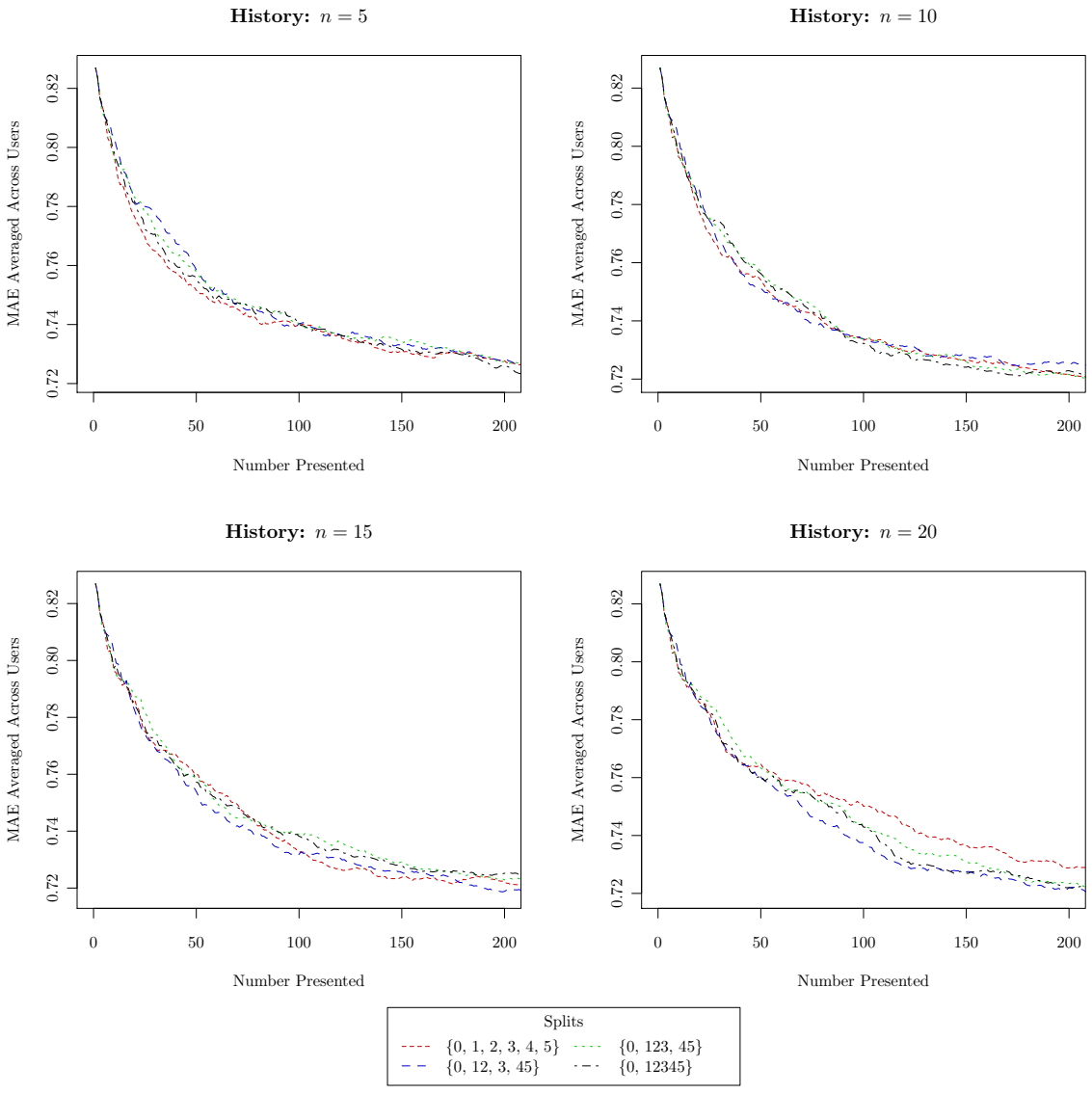

Figure 4.18: Different Splits and History Lengths

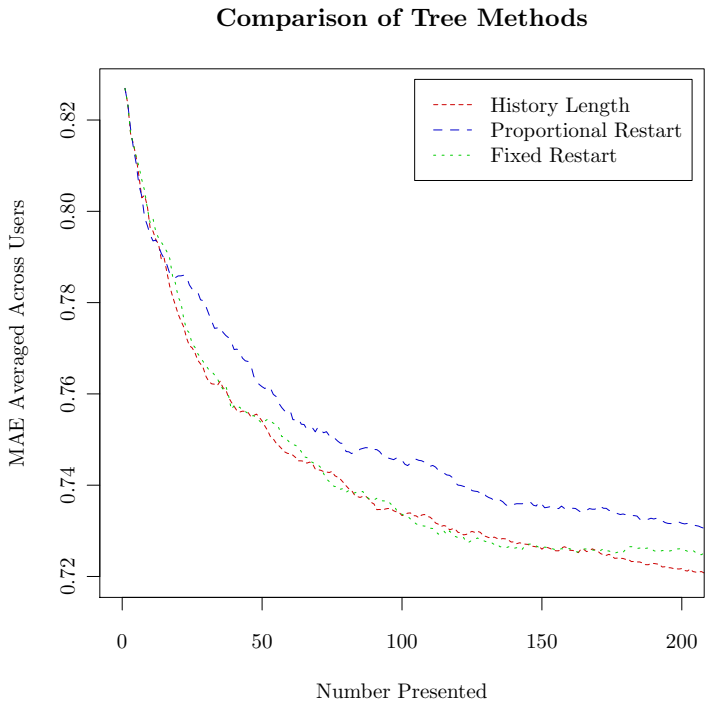

Figure 4.19: Comparison of the Best Tree Recovery Methods

Interestingly the proportional restarting method performs worse than the fixed restarting despite the benefits outlined above in Section 4.2.3. The fixed restarting and history length methods perform on par with each other, with history length performing marginally better in the area we suspect will be most used (up to 50 items presented).

By using the metrics discussed in Section 2.4 we find that the history length method is better at finding high information items to present than the fixed restarting method, which is better at finding items the user can rate. For this reason we prefer the history length method over the fixed restart method.

Henceforth, the Greedy Tree method refers to the tree that is generated when splitting users on individual ratings and using the history method with length 10 to alleviate the node size reduction problem. This method is the final method that will be presented in this thesis.

Figure 4.20 shows the comparison between the Greedy Tree method and the Switching Other People's Greedy method. As can be seen, the Greedy Tree method provides a substantial improvement.

Coincidentally, the use of decision trees to determine which item to present next was

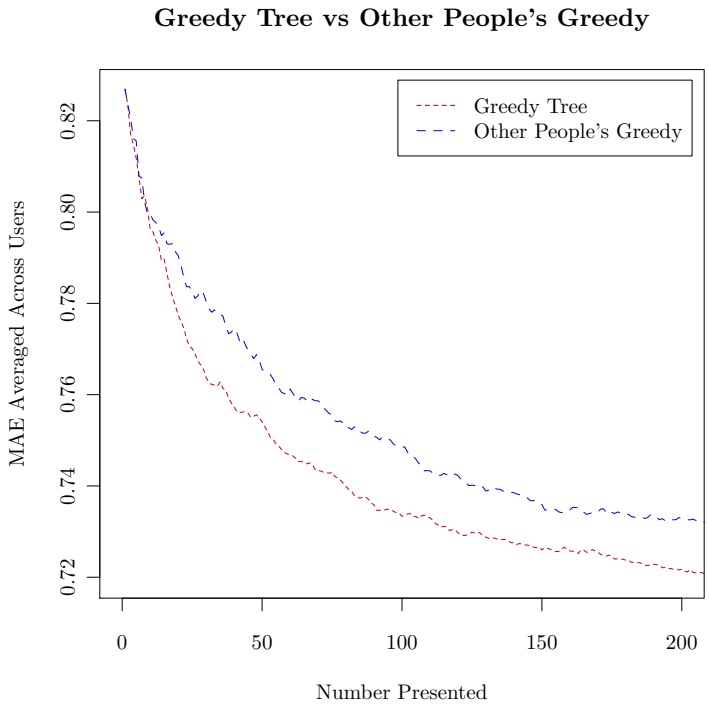

Figure 4.20: Comparison of Greedy Tree and Switching Other People's Greedy

recently described in Golbandi *et al.* (2011). Like the differences between the Other People's Greedy method, as described in Section 4.1.5 and the method described in Golbandi *et al.* (2010), their decision on how to choose which item to present next was different.

Their approach also considered each item presented as a filter for the other users in the dataset. To select the next item to present they chose that item, such that it would minimise prediction error on the test set for the users still being considered as well as splitting the remaining group of users into as equal sized groups as possible.

The tree they constructed used the high/low splits of $\{0, 123, 45\}$, and they did not investigate having different splits, stating that it is "well recognized that decision trees work better with fewer splits". They do, however, recognize that their choice may be sub-optimal. The tree that they generated was only generated to a depth of 6, where they note the problems inherent with the method discussed in Section 4.2.3.

One interesting direction that they take is the construction of multiple trees, to create a forest, which allows them to present multiple items to the user at the same time. Their experimentation shows that this allows them to generate predictions of higher accuracy much sooner in the process.

Chapter 5

Results & Conclusion

Taking the best method that we came up with, that is, the Greedy Tree with splits {0, 1, 2, 3, 4, 5} and history length 10, compared to Entropy0, we see that we have made a significant improvement.

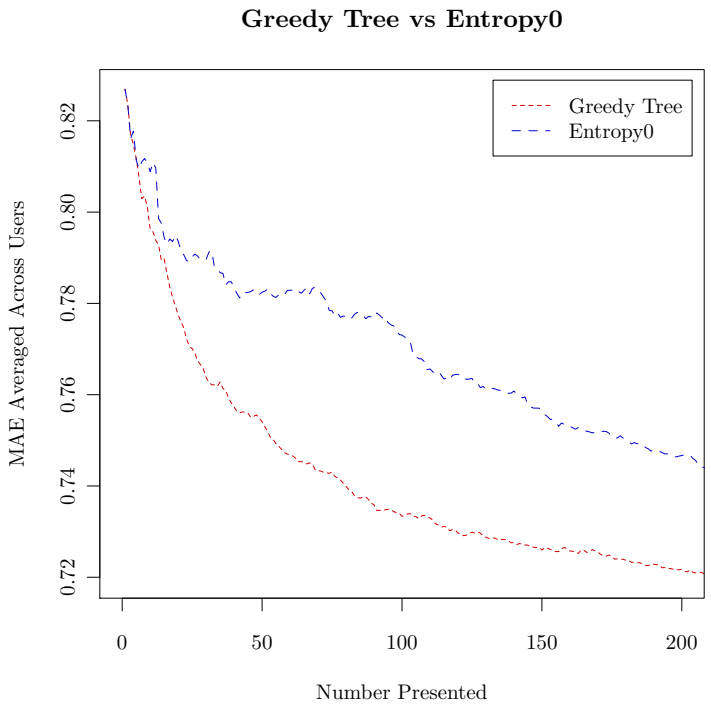

Figure 5.1: Comparison of Greedy Tree and Entropy0

Figure 5.1 shows the comparison between the Entropy0 and the Greedy Tree methods. After presenting 35 movies, the Greedy Tree method has an MAE of 0.762 when

averaged across the users in the sample, while the Entropy0 method has an MAE of 0.786. Although this difference is quite small, to get an equivalent accuracy the Entropy0 method would have to present an additional 91 items. The predictions are made by a method that generates higher accuracy predictions than a commercial system when performance is measured using RMSE on a hold out test set (Bell and Koren, 2008).

Thirty-five items may sound like a large number of items to present to users before they can use the system. In Rashid *et al.* (2002) an online experiment was performed in which they presented groups of 15 items to users upon entry to the system. They recorded the number of items that had to be presented until the user was able to rate 10 items.

Using their methodology we find that the Popularity and Entropy0 methods both require an average of 28 items to be presented, while the Greedy Tree method requires an average of 35 items. Table 5.1 shows the average number of items that are rateable by the user for different numbers of items presented. The numbers show that the Greedy Tree method finds fewer items that the user is able to rate than the Entropy0 method, which in turn finds slightly fewer than the Popularity method.

	Average Number Ratable by User		
Presented	Popularity	Entropy0	Greedy Tree
10	4.09	4.01	3.55
20	7.56	7.49	6.26
30	10.79	10.77	8.83
40	13.98	13.81	11.34
50	17.02	16.83	13.73
60	19.89	19.69	16.10
70	22.65	22.36	18.25
80	25.24	24.89	20.37
90	27.96	27.45	22.50
100	30.11	29.87	24.47

Table 5.1: Average Number of Items Able to be Rated by the User for Different Methods

In addition Rashid *et al.* (2002) noted that some users had to be presented with over

200 items before they were able to rate 10 of them. The Greedy Tree method performs best in this regard, with the worst performing user having to view 3963 items, compared to 4747 for the Popularity method, and 4788 for the Entropy0 method.

Of the 351 users that entered the system during their experimentation phase, 302 completed the sign-up process by rating those 10 items. We note that there need not necessarily be a set number of items that have to be presented for this phase of the system to finish, the user should be able to end this process at any time. This barrier to entry of the system has been shown to improve the later participation in the system (Drenner *et al.*, 2008; Freyne *et al.*, 2009).

We also note the difference between a method that has to generate pages of 15 items and can only update every page, and a method that is able to update after receiving feedback for every item.

A paired t-test against the null hypothesis that there was no difference between the accuracy of the predictions generated by the two methods was carried out. The resulting significance (p-values) after presenting different numbers of items are shown in Table 5.2. We note that we have $p < 0.05$ for $n \geq 20$.

Items Presented	p-value
5	7.250E-1
10	1.731E-2
15	2.505E-1
20	1.090E-2
25	2.240E-2
50	6.610E-5
75	4.356E-8
100	1.021E-9
125	1.379E-8
150	2.892E-7
175	1.547E-7
200	2.518E-8

Table 5.2: p-values for Entropy0 and Greedy Tree After Presenting Different Numbers of Items

All the results presented so far have been against a sample of 500 randomly selected

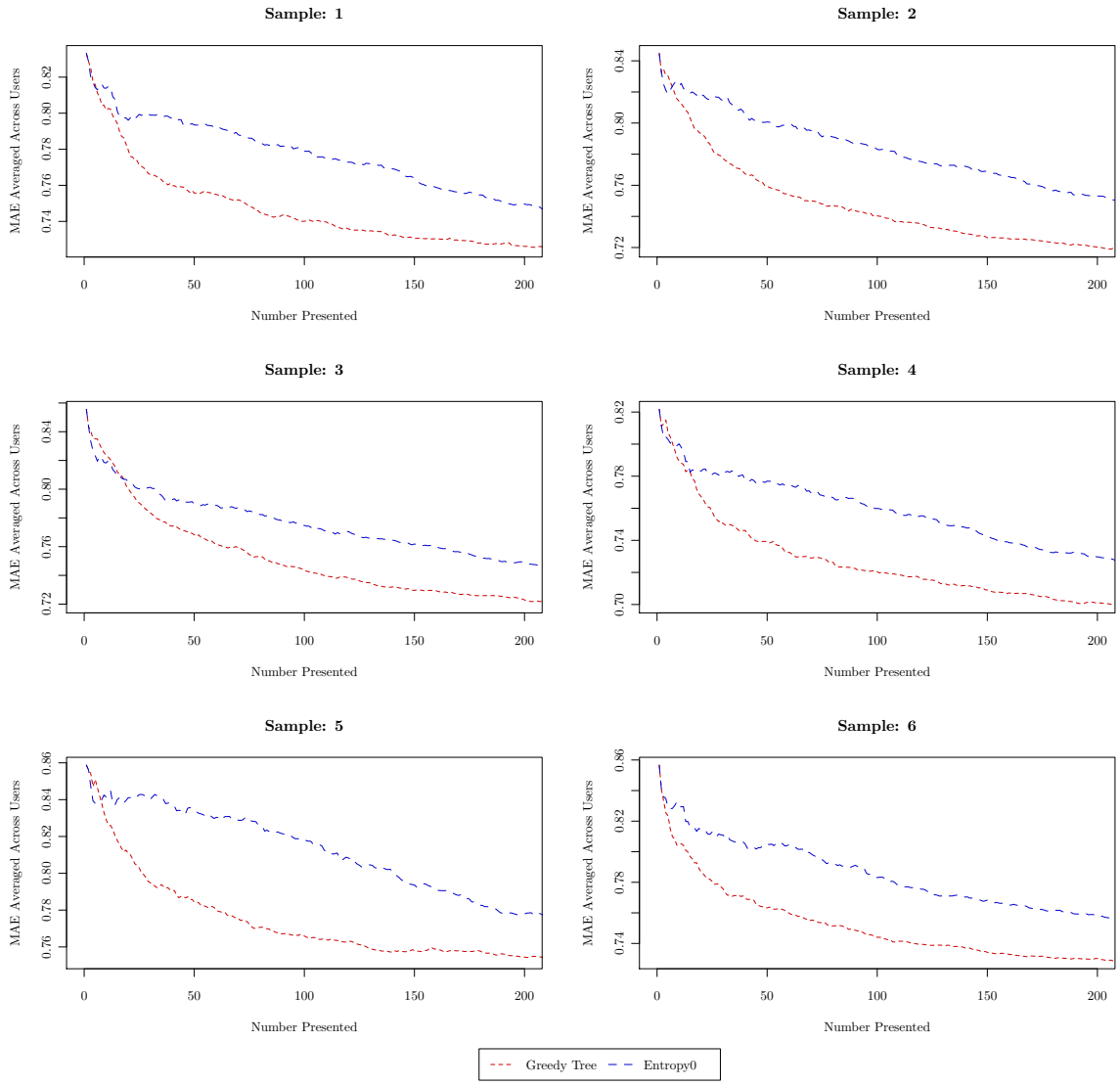

Figure 5.2: Comparison of Entropy0 and Greedy Tree Across Additional Random Samples

users from the dataset. Figure 5.2 shows the performance of the Entropy0 method against the Greedy Tree method on a further six, non-overlapping, random samples of 500 users. We see that the results across these samples is very similar.

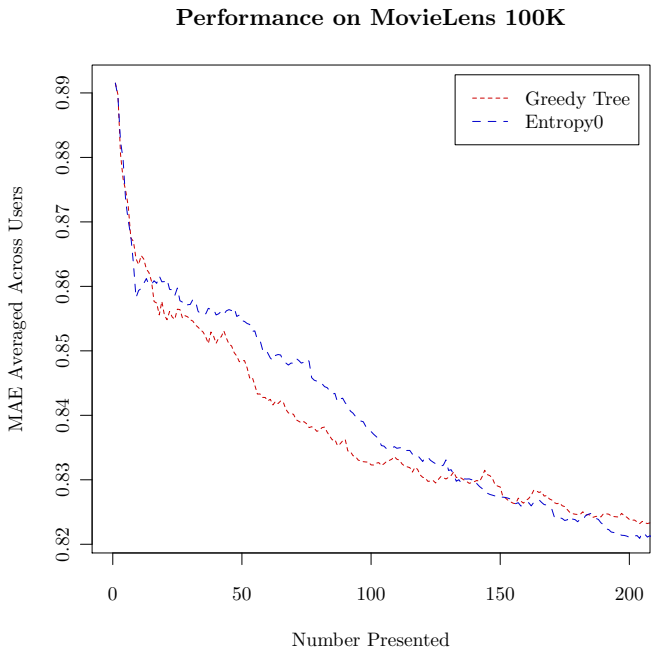

Figure 5.3: Performance on MovieLens 100K

The final method presented also performs better than the current best performing method when using the metric developed in Chapter 2 on the MovieLens 100K dataset, except in the region of 5–15 items and 130+, as shown in Figure 5.3. We posit that with parameter tuning of the parameters in the prediction method (in particular the α values for the statistical modelling described in Section 3.3.1), and the splits and history length of the tree generation method this would be overcome. This secondary dataset was chosen because of its different characteristics, in particular it has more items than users, the opposite of the Netflix dataset.

Figure 5.4 compares the number of items that have to be presented to a user until all the ratings they can give are given by using the Entropy0 and Greedy Tree methods. When points fall below the diagonal line it indicates that the Entropy0 method was able to finish sooner, as was the case for 323 of 500 users, and vice versa for the remaining 177 users. We show this as a point of interest only, noting that users are unlikely to be required to give all the ratings they can before being able to use the system.

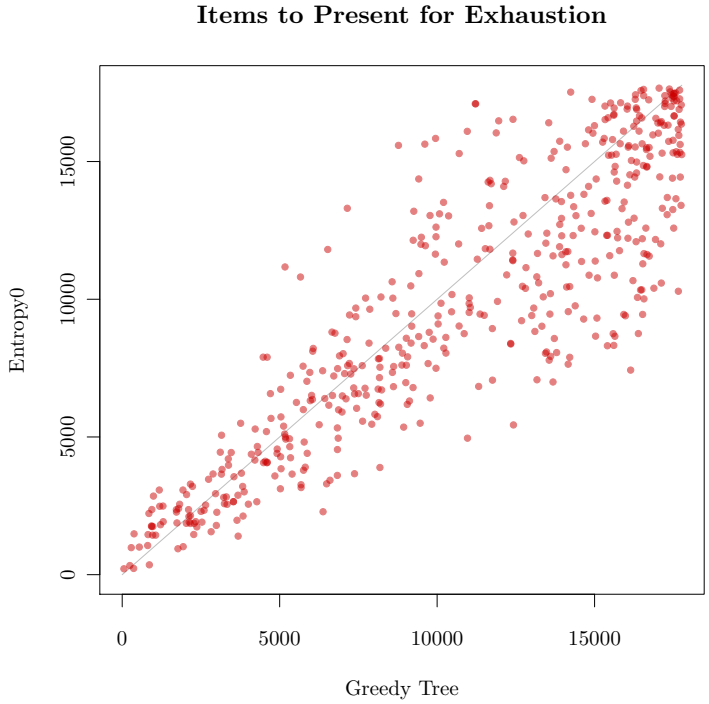

Figure 5.4: Items to Present Until All Ratings From User are Gathered

5.1 Future Work

There remains a reasonable amount of work to be done in the new user problem and in collaborative filtering systems as a whole. We can envision a formal cost function for an item that would take into consideration both the benefit of knowing the rating for that item and the chance that the user would be able to provide a rating for it.

The Greedy Tree method proposed in Section 4.2.3 has an element of this embedded within it. For an item to appear in this list it has to be able to reduce error as well as have been seen by a large number of people who have also rated like the current user has.

The usage of demographic information may also provide an alternate route to the new user problem than described in this thesis. By using such information we may never need to prompt the user for ratings of items before they can use the system. However, with the recent lawsuit against Netflix, we believe that the chances of large datasets being released with this information to be small.

Another approach to the construction of a non-personalised list was recently presented in Golbandi *et al.* (2010). This method also used a greedy approach, selecting items in the order that reduced prediction error on the training set the most. They utilised the original testing and training sets created as part of the Netflix Prize, construction of which is described in Bennett and Lanning (2007). This method was the best performing among several that outperformed the Entropy0 method presented in Rashid *et al.* (2002). The first 15 items generated from this method are compared to the Switching Other People's Greedy method in Table 4.1.

In Golbandi *et al.* (2011) the authors further extended their method by introducing elements from decision trees. At each node the item to present is selected such that the squared error of predictions is minimised across all the groups that the user can be placed into. The branching factor that they consider is that of either being unable to rate the item, liking it, or disliking it, which is represented in our notation as {0, 123, 45}.

The tree that is generated is only generated to a depth of 6, as "going beyond such depths yields very limited accuracy gains". We believe this is related to the method that they use to measure the performance at each node, as our results above indicate substantial improvement after presenting 6 items. In addition the authors did not

investigate other potential splits described in Section 4.2.3.

Another direction that Golbandi *et al.* (2011) take is the use of multiple decision trees to generate a forest, which allows them to present multiple items at once to the user. The generation of the trees for the forest is controlled by a randomization factor that instead of selecting the item that minimises the squared error, it instead assigns to that item the highest probability of being selected. The trees to participate in the forest are then selected such that the blending of predictions made would minimize error on the test set. The forest created with 5 trees provides a significant drop in RMSE of predictions.

The work in Golbandi *et al.* (2011) clearly shows there is room for improvement when constructing the decision tree to use in the initial elicitation procedure. The use of a forest is also another interesting direction that is worthy of exploration. Additionally, we point to some of the features of our methods that could be further expanded upon.

The branching factor, discussed in Section 4.2.3, in the current method is static, we could easily imagine that for some movies you get more information knowing the exact rating given, while others would be provided the same, or more, information just knowing it was rated.

The methods by which the depth of the tree is restricted, discussed in Section 4.2.3, is another area that warrants further investigation; currently the history length is static throughout the tree. This could be changed to become dynamic, remembering as many items as necessary to keep the number of users above a certain threshold.

The decision by which we determine which users to still consider when constructing the tree in the Greedy Tree method is another area that could be looked at. Rather than using just users that have rated the same as the current user, we could consider those users that have a high correlation with the current user. This would, however, require a large amount of processing time and the ability to update correlations as the ratings for the current user were entered into the system.

Along with these methods of choosing the next item to present to the user, in Golbandi *et al.* (2010) the authors present a similar metric to the one derived in Section 2.4. Rather than using the prediction error averaged across users, they instead choose to measure the prediction performance on all users concurrently as a function of how many items were presented to the user. We believe that our method more accurately captures the performance of the system for a new user entering it.

Taking into consideration this subtle difference in the metrics presented in this thesis and in Golbandi *et al.* (2010) it would be interesting to compare the methods presented in Golbandi *et al.* (2010, 2011) with the methods presented in this thesis using both sets of metrics.

5.2 Conclusion

In this thesis we have shown a new way of measuring the success of a method for alleviating the new user problem. This new metric provides a direct comparison on the attributes that we want a method to excel in, that is, we wish the system to increase the prediction accuracy while asking for as few ratings as possible.

We have also provided a way to distinguish between one system that can find larger numbers of items the user can rate which do not have high information content, versus another system that could find fewer items that the user can rate, but each of which has a higher information gain.

We have also shown a progression of methods that the system can use to determine which item to present to the user next. Each step of this progression gradually improves performance, bar two. The final method we describe is a dynamic, adaptive, personalised method. We have shown that even without parameter tuning, this final method also performs admirably on a dataset that was selected for its different properties, that is, having more items than users.

The final method presented improves prediction accuracy after presenting a set number of items when compared to the Entropy0 method described in Rashid *et al.* (2002), and requires significantly fewer items to be presented to get an equivalent prediction accuracy.

References

Abbassi, Z., Amer-Yahia, S., Lakshmanan, L., Vassilvitskii, S., and Yu, C. (2009). Getting recommender systems to think outside the box. In *Proceedings of the Third ACM Conference on Recommender Systems*, 285–288. ACM.

Adomavicius, G. and Zhang, J. (2010). On the stability of recommendation algorithms. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, 47–54. ACM.

Ahn, H. (2008). A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, *178*(1), 37–51.

Amatriain, X., Pujol, J., Tintarev, N., and Oliver, N. (2009). Rate it again: increasing recommendation accuracy by user re-rating. In *Proceedings of the Third ACM Conference on Recommender Systems*, 173–180.

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

Bao, X., Bergman, L., and Thompson, R. (2009). Stacking recommendation engines with additional meta-features. In *Proceedings of the Third ACM Conference on Recommender Systems*, 109–116.

Basilico, J. and Hofmann, T. (2004). Unifying collaborative and content-based filtering. *Proceedings of the Twenty-First International Conference on Machine Learning*, 9–16.

Bell, R. and Koren, Y. (2008). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE International Conference on Data Mining*, Volume 07, 43–52. IEEE.

Bennett, J. and Lanning, S. (2007). The Netflix Prize. In *Proceedings of KDD Cup and Workshop*, Volume 2007, 3–6.

Breese, J., Heckerman, D., Kadie, C., and Others (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, Volume 461, 43–52.

Brun, A., Castagnos, S., and Boyer, A. (2011). *Social recommendations: mentor and leader detection to alleviate the cold-start problem in collaborative filtering*, 1–21.

Celma, O. and Herrera, P. (2008). A new approach to evaluating novel recommendations. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, 179–186. ACM.

Chirita, P.-A., Nejdl, W., and Zamfir, C. (2005). Preventing shilling attacks in online recommender systems. In *Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management*, 67–74. ACM.

Das, A., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, 271–280. ACM.

Davidson, J., Liebald, B., Liu, J., Nandy, P., and Van Vleet, T. (2010). The YouTube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, 293–296. ACM.

Drenner, S., Sen, S., and Terveen, L. (2008). Crafting the initial user experience to achieve community goals. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, 187–194. ACM.

Freyne, J., Jacovi, M., Guy, I., and Geyer, W. (2009). Increasing engagement through early recommender intervention. In *Proceedings of the Third ACM Conference on Recommender Systems*, 85–92. ACM.

Golbandi, N., Koren, Y., and Lempel, R. (2010). On bootstrapping recommender systems. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, 1805–1808. ACM.

Golbandi, N., Koren, Y., and Lempel, R. (2011). Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, 595–604. ACM.

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

Gunawardana, A. (2008). Tied Boltzmann machines for cold start recommendations. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, 19–26.

Gunawardana, A. and Shani, G. (2009). A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research*, *10*, 2935–2962.

Gupta, D., Digiovanni, M., Narita, H., and Goldberg, K. (1999). Jester 2.0: Evaluation of a new linear time collaborative filtering algorithm. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 291–292.

Hang, Y., Guiran, C., and Xingwei, W. (2009). A Cold-Start Recommendation Algorithm Based on New User's Implicit Information and Multi-attribute Rating Matrix. *Ninth International Conference on Hybrid Intelligent Systems*, *2*, 353–358.

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

Herlocker, J., Konstan, J., and Riedl, J. (2000). Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, 241–250. ACM.

Herlocker, J., Konstan, J., Terveen, L., and Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, *22*(1), 5–53.

Hofmann, T. and Hartmann, D. (2005). Collaborative Filtering with Privacy via Factor Analysis. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, 791–795.

Huang, Z., Chen, H., and Zeng, D. (2004). Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems*, *22*(1), 116–142.

Jahrer, M., Töscher, A., and Legenstein, R. (2010). Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 693–702. ACM.

Jung, K.-Y., Park, D., and Lee, J. (2004). Hybrid collaborative filtering and content-based filtering for improved recommender system. *Computational Science-ICCS 2004*, 295–302.

Kohrs, A. and Merialdo, B. (2001). Improving Collaborative Filtering For New-Users By Smart Object Selection. In *Proceedings of International Conference on Media Futures*.

Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. (1997). GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, *40*(3), 77–87.

Koren, Y. (2009). The BellKor solution to the Netflix grand prize. Submitted to Netflix to meet grand prize conditions. Available from: `http://netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf`.

Koren, Y. (2010a). Collaborative filtering with temporal dynamics. *Communications of the ACM*, *53*(4), 89–97.

Koren, Y. (2010b). Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data*, *4*(1), 1–24.

Lam, S. and Riedl, J. (2004). Shilling recommender systems for fun and profit. In *Proceedings of the 13th international Conference on World Wide Web*, 393–402. ACM.

Lam, X., Vu, T., Le, T., and Duong, A. (2008). Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication*, 208–211. ACM.

Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, *7*(1), 76–80.

Massa, P. and Avesani, P. (2004). Trust-aware collaborative filtering for recommender systems. *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, 492–508.

Massa, P. and Bhattacharjee, B. (2004). Using trust in recommender systems: an experimental analysis. *Trust Management*, *2995*, 221–235.

McLaughlin, M. and Herlocker, J. (2004). A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 329–336. ACM.

McNee, S., Riedl, J., and Konstan, J. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*, 1097–1101. ACM.

Mehta, B. (2007). Unsupervised shilling detection for collaborative filtering. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, Volume 22, 1402–1407.

Narayanan, A. and Shmatikov, V. (2008). Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy*, 111–125. IEEE.

Papagelis, M., Plexousakis, D., and Kutsuras, T. (2005). Alleviating the sparsity problem of collaborative filtering using trust inferences. *Trust Management*, *3477*, 224–239.

Park, S. and Chu, W. (2009). Pairwise preference regression for cold-start recommendation. In *Proceedings of the Third ACM Conference on Recommender Systems*, 21–28. ACM.

Park, S., Pennock, D., Madani, O., Good, N., and DeCoste, D. (2006). Naive filterbots for robust cold-start recommendations. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 699–705. ACM.

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, Volume 2007, 2–5.

Pilászy, I. and Tikk, D. (2009). Recommending new movies: even a few ratings are more valuable than metadata. *Proceedings of the Third ACM Conference on Recommender Systems*, 93–100.

Piotte, M. and Chabbert, M. (2009). The Pragmatic Theory solution to the Netflix Grand Prize. Submitted to Netflix to meet grand prize conditions. Available from `http://netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf`.

Polat, H. and Du, W. (2005). SVD-based collaborative filtering with privacy. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, 791–795. ACM.

Potter, G. (2008). Putting the collaborator back into collaborative filtering. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, 1–4. ACM.

Rashid, A., Albert, I., Cosley, D., and Lam, S. (2002). Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th International Conference on Intelligent user Interfaces*, 127–134.

Rashid, A., Karypis, G., and Riedl, J. (2008). Learning preferences of new users in recommender systems: an information theoretic approach. *ACM SIGKDD Explorations Newsletter*, *10*(2), 90–100.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–186. ACM.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, 791–798. ACM.

Schein, A., Popescul, A., Ungar, L., and Pennock, D. (2001). Generative models for cold-start recommendations. In *Proceedings of the 2001 SIGIR Workshop on Recommender Systems*, Volume 6.

Schein, A., Popescul, A., Ungar, L., and Pennock, D. (2005). CROC: a new evaluation criterion for recommender systems. *Electronic Commerce Research*, *5*(1), 51–74.

Shardanand, U. and Maes, P. (1995). Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 210–217.

Tintarev, N. and Masthoff, J. (2007). A survey of explanations in recommender systems. In *IEEE 23rd International Conference on Data Engineering Workshop*, 801–810. IEEE.

Toscher, A., Jahrer, M., and Bell, R. M. (2009). The BigChaos solution to the Netflix grand prize. Submitted to Netflix to meet grand prize conditions. Available from: `http://netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf`.

Victor, P., Cornelis, C., Teredesai, A., and De Cock, M. (2008). Whom should I trust?: the impact of key figures on cold start recommendations. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, 2014–2018. ACM.

Weng, L., Xu, Y., Li, Y., and Nayak, R. (2008a). Exploiting item taxonomy for solving cold-start problem in recommendation making. In *20th IEEE International Conference on Tools with Artificial Intelligence*, Volume 2, 113–120. IEEE.

Weng, L., Xu, Y., Li, Y., and Nayak, R. (2008b). Improving recommendation novelty based on topic taxonomy. *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology Workshops*, 115–118.

Zhang, L., Meng, X., Chen, J., Xiong, S., and Duan, K. (2009). Alleviating Cold-Start Problem by Using Implicit Feedback. In *Advanced Data Mining and Applications*, 763–771. Springer.

Zhang, S., Ford, J., and Makedon, F. (2006). A privacy-preserving collaborative filtering scheme with two-way communication. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, Volume 9, 316–323. ACM.

Ziegler, C., McNee, S., Konstan, J., and Lausen, G. (2005). Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, 22–32. ACM.