NTNU
Innovation and Creativity

Collaborative Filtering for Recommending Movies

Cecilie Bøe

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Helge Langseth, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

As ever more information gets available in our everyday life, the "information overload" problem exists in that it gets harder to extract the valuable knowledge from the total quantity of information. In automated information retrieval methods, this is typically solved by using queries regarding some topic of interest, and for instance returning web pages containing a user-specified phrase.

On the other hand, collaborative filtering algorithms recommend new items to a user, or predict the utility for a particular item for a user, based on the information available about preferences of so-called like-minded users. Collaborative filtering is a hot topic within the machine learning area, and a lot of ongoing research focuses on the movie rating scenario.

The object of this project is to work with collaborative filtering algorithms, and includes the following tasks:
- give a description of various methods for collaborative filtering presented in the literature
- design a model for doing collaborative filtering, which is to be tailored for recommending movies (by using the MovieLens dataset)
- examine how demographic information (like the user's age and place of residence) can be utilized to improve the quality of recommendations

Assignment given: 15. January 2007
Supervisor: Helge Langseth, IDI

Abstract

There is a significant amount of ongoing research in the collaborative filtering field, with much of the research focusing on how to most accurately give item predictions to a user, based on ratings given by other users with similar rating patterns.

The objective of this project is to build movie rating prediction models with a simple and intuitive representation, based on previous work within the area. Important factors are the investigation of the predictive power of these models, and the research on how the use of content information can improve accuracy when the available data is sparse.

We show that latent class models provide an expressive, but yet simple way to represent the movie rating scenario, and that the models have great potential when it comes to predictive accuracy. We conclude that the inclusion of additional content features into the models can help improve the accuracy when there is little data available.

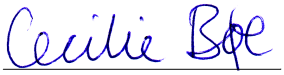
PREFACE

This master thesis was written during the spring semester 2007 for the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

The subject of the project was worked out in cooperation with my supervisor at IDI, associate professor Helge Langseth in the Division of Intelligent Systems.

I would like to thank Helge Langseth for valuable input and continuous feedback throughout the project period. I also give my thanks to my fellow students at "Ugle" for great coffee, numerous discussions regarding all kinds of droll topics, and instant technical assistance whenever required.

Trondheim, June 2007



Cecilie Bøe

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the project, by giving the motivation for starting it. A description of the context in which the project is to be carried out is given, as well as a listing of the main goals for the project. The chapter concludes with an outline of the rest of the report.

1.1 Motivation

The motivation behind the use of collaborative filtering for recommending movies is that there are generally too many movies available, and too little time to go through the list of all movies, to see the ratings or the descriptions of the movies. The idea is that you would want to ask friends or other acquaintances to recommend you some movies if you know they usually have a similar taste in movies as you. In recommender systems this process is automated, so that recommendations are based on ratings from all other users in the system, acquaintances or not.

Collaborative filtering is a hot topic within the machine learning area, and there is much research going on. At the time there is an ongoing contest about movie rating, namely the Netflix Prize contest (www.netflixprize.com). The goal is to achieve as good accuracy as possible using their training data set containing about 100 million ratings from over 480 thousand users on nearly 18 thousand movies. The contest has a first prize of one million dollars, and several of the most prominent researchers within the field of collaborative filtering take part in the competition.

1.2 Project context

There were some fundamental conditions on which the project was based when it was started. These regard the choice of data set, and the premises for the implementation tool to be used.

1.2.1 Data set

The data set on which the model should be tested, is a data set extracted from the movie recommending site MovieLens (movielens.umn.edu/login), and made available for research purposes by GroupLens Research on their web page [35]. The data set

contains 100 000 ratings (rating scale from 1 to 5) from 943 different users on 1682 different movies, as well as additional information about the movies and users, such as genre of the movie, and age of the user. The 943 users has at least 20 registered ratings each. However, 18 of these 943 users were registered with a zip code containing letters, and as this would create problems with some of the tests in the experiments, these 18 users were simply dropped from the data set, leaving 925 users with 97914 ratings on 1682 movies.

1.2.2 Implementation tool

The implementation tool/framework/programming language or combinations of these to be used for building the chosen model, should either be mastered already, or be fairly easy to get sufficiently acquainted with.

1.3 Project goals

The goal for the project is to produce the following results:

- a thorough survey of the most important algorithms and techniques which exist in the Collaborative Filtering field today
- a simple, intuitive, and easily representable model for recommending movies based on a database of previous ratings, whose prediction performance is comparable to related works
- research results about whether inclusion of additional features into the model can affect the quality of predictions

1.4 Report outline

The rest of the report is organized as follows:

Chapter 2 gives an overview of various algorithms and techniques used in Collaborative Filtering, related to the state of the art within the field.

Chapter 3 reviews the collaborative filtering algorithms outlined in the previous chapter with respect to the data set, and to the problem definition and goals of the project.

Chapter 4 describes the model implementation.

Chapter 5 lists and comments the results of the executed experiments.

Chapter 6 states the most important conclusions drawn in the project, as well as potential improvements and additions in prospective further work.

Chapter 2

Related Work

This chapter gives a survey of related work within the field of Collaborative Filtering. An overview is given of the most important techniques in use and of the metrics most frequently applied to evaluate them. In addition, some important research issues regarding the area are discussed.

2.1 Introduction and background

As ever more information gets available in our everyday life, the ”information overload” problem exists in that it gets harder to extract the valuable knowledge from the total quantity of information. In automated information retrieval methods, this is typically solved by using queries regarding some topic of interest, and for instance returning web pages containing a user-specified phrase. This type of information filtering uses the *content* of the items to determine which items are relevant. A well-known example is the Google search engine (www.google.com).
However, there are cases in which it is inconvenient to specify a query based on the content [5], including:

- when the content of the items is not easily analyzed by automated processes
- when we want to filter items based on properties like quality and taste
- when we want to include the possibility of making serendipitous recommendations, that is, recommending items to a user that the user would not himself consider in the first place

The idea behind *Collaborative Filtering* is that you tend to listen to recommendations from people whom you know has the same taste as you, independently of the content of the item they recommend you. The goal in collaborative filtering is to recommend new items to a user, or predict the utility for a particular item for a user, based on the information available about preferences of so-called like-minded users [38].

The term Collaborative Filtering was coined by Goldberg et al. [15]. They built the Tapestry system, which relied on many evaluators who could evaluate the various items within a site, and then the users of the site could choose which evaluators' recommendations to pay attention to. Tapestry used a complex query language to formulate queries such as ”show me all documents voted for by Joe”.
However, in large information handling systems, you cannot rely on every user knowing the other users, and thus it is difficult to know which evaluator to trust, and another

difficulty arises in that you cannot rely on every user being able to formulate the necessary queries. In addition, there are privacy issues that makes some users want to give anonymous recommendations.
The GroupLens project [36] was the first project to introduce an automated collaborative filtering system. GroupLens provided personalized recommendations for users of Usenet news articles. A network system called Ringo [41] used collaborative filtering algorithms for doing music recommendation. Commercial systems successfully using collaborative filtering methods include the Internet book shop Amazon (www.amazon.com) and music shop CDNOW (www.cdnow.com). The collaborative filtering systems help the users find products that they will probably like, and the businesses benefit from getting more sales [38].

2.1.1 The collaborative filtering problem

In the collaborative filtering problem we typically have a database consisting of a set of *users* U, a set of *items* I, some possible *rating values* V, and a number of triples $\{u, i, v\}$ meaning "user u assigned rating value v to item i". The task of the collaborative filtering algorithms is then to use the provided data to make recommendations for an *active user* - the user served at the moment - on the items he has not yet rated.

2.1.2 Implicit and explicit ratings

There are different ways to extract rating values or preferences from users on items. When a user has available a discrete numerical rating scale, that is, the user can for instance assign from 1 star to 5 stars to an item, then the user's ratings are called *explicit* ratings. In this case the user consciously expresses his or her preference for the given item [5].
On the other hand, user preferences may also be extracted in an indirect way, by for instance collecting data for how many times a user visits a given web page, how long time a user spends reading a news article and so on. This kind of preference data is called *implicit* ratings, and the triple $\{u, i, v\}$ is now in many cases reduced to a tuple $\{u, i\}$ meaning, for instance, "user u bought item i".

2.1.3 Forced prediction and top-N recommendation

There are two main types of recommendations that can be provided by a recommender system [5]. In the first case, one item at a time is presented to the user, together with the rating value the system expects the user to give this item. The goal of the collaborative filtering algorithm is then to provide a rating value prediction as close as possible to the rating value the user would have given. This type of recommendations is often referred to as forced prediction [24]. This name is used because we have a situation where the system has to give a prediction for each specific item, that is, the system cannot "choose" for which items it wants to provide predictions.
Another recommendation type is top-N recommendation, in which the task of the recommender system is to give a list of N items that the user will like the most [38]. This list must contain only "new" items; items not already purchased or rated by the user. In this type of recommendation, the order of the top N items on the list is irrelevant, as long as all the best N items are on the list, and all other items are not.
Since forced prediction provides a potential relative ordering of all items for a user, this

type of prediction can also be used to produce a list of the top N items. However, in the top-N recommendation case we are only interested in the relative ordering of the items, and the prediction of how many stars the user would give each item is not necessary here. Hence, measuring the accuracy of each rating prediction makes no sense.

2.1.4 Memory-based and model-based collaborative filtering methods

Breese et al. [5] made a distinction between *Memory-based* and *Model-based* collaborative filtering algorithms, which has become the most typical classification of algorithms. Memory-based methods use the entire database for every prediction, computing some sort of similarity either between users or items, based on the history of the active user. Model-based methods, on the other hand, use an initial database to build a prediction model, which is subsequently used to make the predictions. This prediction model has to be updated from time to time as the database changes.

The distinction between memory-based and model-based methods can be compared to what is called lazy and eager commitment in the machine learning area [33]. The memory-based methods do not perform any learning until actually presented with a query, and when a query is given, they have to use the entire database every time to give a prediction. This is a typical example of lazy commitment, which leads to quick learning, but slower prediction making. In this lazy learning process, a local model containing the most similar neighbours is fitted around the active user.
On the other hand, the model-based methods start out by learning a global model based on all the given data, and when presented with a query, the prediction is made more quickly, based on the model already learned. This conforms to eager commitment, in which the decision rules are made as soon as possible.

In this chapter, an overview of the most important methods within the field of memory-based and model-based methods will be given.

2.1.5 The sparsity problem

The sparsity level of a database is defined as

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

When we have a collaborative filtering problem consisting of predicting product ratings for a user, there is typically a very sparse database available, as each user normally provides ratings only for a limited amount of products in the domain.

The lack of adequate ratings will in most cases affect the quality and coverage of the recommendations, and this is especially a problem for the memory-based algorithms [7]. An example of how the quality problem might emerge is given in Herlocker et al. [20]. When the recommender system is to make a prediction on an item for which only one user has previously provided a rating value, all predictions for this items will be the same rating value as given by the one user. In the case where a prediction is to be made on an item not previously rated, this cannot be done with straightforward memory-based algorithms.
Model-based methods partially escape these two problems by instead building a model in which the relationships between the users are computed based on the rating *patterns*, rather than ratings on common items. However, the quality of model-based methods is nevertheless challenged by data sparsity, as sparsity can lead to overfitting of the data

in the models built, and thus reducing prediction quality.
The sparsity problem, with some suggested modifications to overcome the problem, will be addressed throughout this chapter.

2.2 Memory-based methods

In memory-based collaborative filtering algorithms, the ratings of the other users in the database are used to make a prediction for the active user. The other users' ratings must be weighted in some way according to their similarity to the active user. The users contributing in the calculation of the prediction for the active user are said to be the active user's *neighbours*, and hence memory-based methods are also called *neighbourhood-based* methods [19].

Herlocker et al. [19] separate the neighbourhood-based methods into three steps;

1. weighting all users with respect to similarity with the active user
2. selecting a subset of users to use as a set of predictors
3. normalizing ratings and computing a prediction from a weighted combination of selected neighbours' ratings.

Each of these three steps will be described below.

2.2.1 Similarity weighting

The first step in the neighbourhood-based algorithms is to weight all users with respect to their similarity to the active user, so as to put more weight on the users with preferences similar to the active user's. There are several different ways of computing this similarity, and extensive descriptions are given by Herlocker et al. in [20].
Breese et al. [5] use the details of how the weight calculation is carried out to distinguish between various collaborative filtering algorithms. There are mainly two types of algorithms which adjust the weights in different ways; correlation and vector similarity.

Correlation

Generally, the correlation between two random variables indicates the strength and direction of a linear relationship between the two variables. The two most popular coefficients for measuring this correlation are the *Pearson correlation coefficient* and the *Spearman rank correlation coefficient*.

Pearson correlation: The similarity weight between the active user a and neighbour u depends on the degree to which a linear relationship exists between the two variables [19], and is defined by the Pearson correlation coefficient. This correlation coefficient divides the covariance of the two user variables by the product of their standard deviations. The Pearson correlation coefficient is stated in Equation 2.1 [5]

$$\omega_{a,u} = \frac{\sum_{i=1}^{m}(r_{a,i} - \bar{r_a}) * (r_{u,i} - \bar{r_u})}{\sqrt{\sum_{i=1}^{m}(r_{a,i} - \bar{r_a})^2 * \sum_{i=1}^{m}(r_{u,i} - \bar{r_u})^2}} \tag{2.1}$$

where m is the number of items for which the active user and user u have made common ratings, $\bar{r_a}$ is the mean user-specific rating for user a, and $\bar{r_u}$ is the mean user-specific

rating for user u. The mean ratings are subtracted to *normalize* the ratings, as different users may rate on different scales. The Pearson correlation relies on a number of assumptions regarding the data, including the rating scale and the assumption of existence of a linear relationship between the variables.

Spearman rank correlation: The Spearman rank correlation coefficient, stated in Equation 2.2 [19], is similar to the Pearson correlation coefficient above, but uses the *rank* of a rating instead of the actual value of the rating.

$$\omega_{a,u} = \frac{\sum_{i=1}^{m}(rank_{a,i} - \bar{rank}_a) * (rank_{u,i} - \bar{rank}_u)}{\sqrt{\sum_{i=1}^{m}(rank_{a,i} - \bar{rank}_a)^2 * \sum_{i=1}^{m}(rank_{u,i} - \bar{rank}_u)^2}} \tag{2.2}$$

For example, if the ratings $\{r_1 = 1,\ r_2 = 3,\ r_3 = 4\}$ are given, this corresponds to the ranks $\{rank_1 = 3,\ rank_2 = 2,\ rank_3 = 1\}$ in the Spearman correlation. This means that the computation of the Spearman rank correlation coefficient is independent of model assumptions like rating scale.

Herlocker et al. found in their experiments that Spearman correlation performed similarly to Pearson correlation [19]. They speculated that the great number of tied ranks (for instance, the ratings $\{r_1 = 2,\ r_2 = 2,\ r_3 = 3,\ r_4 = 3\}$ would result in the ranks $\{rank_1 = 3,\ rank_2 = 3,\ rank_3 = 1,\ rank_4 = 1\}$) might have led to a decrease in the performance of the Spearman correlation, and that increasing the size of the rating scale would give Spearman correlation an advantage to Pearson correlation. As the Spearman coefficient is not dependent of the model assumptions such as rating scale, or being hurt by outliers[1], it should therefore perform consistently better across diverse data sets.

Vector similarity

When the active user is to be compared to user u, the ratings of the active user and the ratings of user u can be transformed into two vectors. The *cosine vector similarity* is then computed as the cosine of the angle between the two vectors [5].

Breese et al. [5] compared cosine vector similarity and Pearson correlation for similarity weighting, and Pearson showed to give better accuracy.

In [38], Sarwar et al. suggested an *adjusted cosine similarity* measure. The idea behind this is that when computing the user vectors, the fact that users might be rating on different scales (rating "3" might mean different thing to different users), and this should be taken into consideration. This *normalization* of ratings is implicit in the Pearson and Spearman correlation coefficients, as the user-specific mean rating is subtracted from each rating. To provide such a normalization with vector similarity as well, the adjusted cosine similarity subtracts the corresponding user average from each co-rated pair. The experiments of Sarwar et al. gave a higher accuracy when using the adjusted measure over the basic one.

2.2.2 Selecting neighbourhoods

When GroupLens [36] introduced the first automated collaborative filtering system, they used a neighbourhood-based algorithm [19], weighting all users in the database

[1] An outlier with very high rating will only have one higher rank value than the next rating with the Spearman rank

according to their similarity to the active user. The Ringo music recommender used only the users in the database that were more similar to the active user than a given threshold [41]. The Bellcore Video Recommender used a random subsample of all the users to compute the most similar neighbours, to relieve the system from a great number of computations [21].

Generally, two types of techniques are used to select how many neighbours to use; correlation-thresholding - choosing all the neighbours that are more similar than a given threshold, or best-N-neighbours - choosing the N most similar neighbours [19]. Herlocker et al. performed experiments with both types of techniques [19]. The results showed that using a subset of neighbours instead of all the users in the database tends to result in higher prediction accuracy. However, there is a trade-off between choosing enough neighbours to give acceptable coverage, and achieving higher accuracy by choosing few enough neighbours to keep the valuable high correlations from being lost in noise from lower correlations. Thus, the size of the neighbourhood is very important for the quality of the predictions.

2.2.3 Producing a prediction

The basic way to make a prediction is to compute a weighted average of the ratings of the selected neighbours [19]. This is what is done in the Ringo system [41]. In the GroupLens project [36], they computed the average deviation of a neighbour's rating from that neighbour's user-specific mean rating. This is done to take into account that users may be rating on different scales. GroupLens's *deviation-from-mean* approach is shown in Equation 2.3

$$p_{a,i} = \bar{r_a} + \frac{\sum_{u=1}^{n}(r_{u,i} - \bar{r_u}) * \omega_{a,u}}{\sum_{u=1}^{n}\omega_{a,u}} \tag{2.3}$$

where $\omega_{a,u}$ is the similarity weight between the active user a and the user u.
Herlocker et al. applied this approach in their experiments [19], and concluded that it led to significantly better prediction accuracy over a normal weighted average. They also hypothesized that considering the variances in each user's ratings (called Z-scores in their work) might improve the quality of the predictions, but the differences in spread between users' rating distributions showed not to have any effect on the predictions.
Rong Jin et al. [27] built two separate models for each user; one preference model capturing which items the user prefer, and a rating model which describes what rating a user would give an item given the preference information. The similarity between two users was then computed based on the preference models, not the explicit user ratings. They concluded that this model outperformed three other standard collaborative filtering methods in all experiments.

2.2.4 Extensions to memory-based methods

There are a number of direct problems with the use of memory-based methods, which either might effect the quality of the prediction, or the coverage potential of the method. Some of these problems, along with solutions or improvements suggested in the literature, are described here.

Default ratings

Memory-based methods obviously have a problem when they are to compare two users who have none or very few rated items in common. This fact was illustrated by examples in Section 2.1.5, and is easily understood when looking at the Pearson and Spearman coefficients in Equations 2.1 and 2.2. In those equations, m is the number of items for which the active user and user u have common ratings, and thus if m is zero or a very low number, this is bound to be a problem for the similarity computation. Breese et al. [5] suggested to assign a default rating to items rated by only one of the two users, such that the union of rated items for the two users could be used in the comparison. It is also possible to assign default ratings to items not rated by any of the users, and thus be able to utilize more items in the comparison, but of course the quality of the predictions will eventually suffer when there are a lot of "fake" ratings in the database.

Inverse user frequency

In a database of user ratings there typically exist some items which are universally liked by all users. These items might not be as useful to capture similarity between users as less popular items. To take this into consideration, Breese et al. [5] employed *inverse user frequency*, which reduces the weights on the universally liked items. This extension led to better performance in all of their experiments where the extension was applicable.
Herlocker et al. [19] realized the same problem, and applied variance weighting such that the distinguishing movies had more influence in determining a correlation. Though contrary to their hypothesis, this had no significant effect on the prediction accuracy of their algorithm. However, a user which actually dislikes an item that everyone else likes, could provide useful information which they did not utilize in their algorithm.

Significance weighting

If two users have rated just a few items in common, say two items, the similarity may be very high even though they do not really have the same taste. They just coincidentally rated these two items equally. It can be a problem for the similarity weighting that in some cases the similarity measures are based on little data, which leads to less trustworthy predictions. Herlocker et al. [19] experimented with adding a correlation significance weighting factor that devalued similarity weights which were based on a small number of co-rated items. They demonstrated that this in fact resulted in a significant gain in prediction accuracy in their experiments.

Case amplification

Breese et al. [5] also experimented with weighting higher the weights close to 1, and punish the lower weights, to increase the contribution of the closest neighbours, and decrease that of the more distant neighbours. They experienced a significant improvement in their top-N recommendation (see Section 2.1.3) experiments, but no visible effect in the forced prediction case. The combination of case amplification and inverse user frequency resulted in higher advantages than when using inverse user frequency alone.

2.2.5 Item-based collaborative filtering

Nowadays, the memory-based approaches explained so far in this section are known as *user-based* collaborative filtering algorithms. This is because they employ users' similarities for the formation of the neighbourhood of most similar users [42]. In this kind of collaborative filtering systems, the amount of work increases with the number of users in the system. This can be a problem when the number of users increases, and Sarwar et al. suggested an *item-based* technique to overcome this problem [38]. Their technique used the user-item matrix to analyze relationships between items, to find similar items instead of similar users. Recommendations are then made by finding items which are similar to the items the user has previously liked.

The idea behind the item-based technique is that the system's containment of items is more or less static, that is, some users leave the system and new users enter, but the items for which they provide ratings are largely the same. This means that relationships between the items can be precomputed, and less time is necessary for online computation. The main motive for the item-based method is to be able to scale to large data sets and at the same time produce high-quality recommendations.

By using the similarity relationships between items instead of users, the data sparsity problem may also be reduced. This is because the majority of recommender systems typically deals with a lot more users than items, and users tend to enter and leave the database at a higher frequency than products arrive or disappear from the market. Hence, the relationships between items should be more dense, that is, there exist more common ratings concerning two random items in the database, than ratings concerning two random users.

It is worth noting that the item-based collaborative filtering method is not purely memory-based, as it is based on some precomputations which are stored in a model. It is thus leaning towards the model-based methods.

2.2.6 Machine learning approaches

In his master's thesis, Benjamin Marlin considered the use of classical machine learning approaches as classifiers within the collaborative filtering area [31]. The k-nearest neighbour algorithm classifier has a very close relationship to the memory-based algorithms in collaborative filtering algorithms, and will be described here. Other machine learning approaches considered by Marlin are largely model-based, and are described in the next section, in Section 2.3.3.

K-nearest neighbour

Marlin stated that the neighbour-based rating prediction algorithms are really just a specialization of the well-known k-nearest neighbour classifier [33]. The k-nearest neighbour (KNN) classifier is a memory-based or instance-based machine learning method, which stores all the training instances, and use the whole collection of training instances every time a classification is to be performed. To classify a new query vector, a distance function is used to compute the distance between the query vector and each vector stored in the database. The k nearest neighbours are chosen, and the output prediction is taken to be the class of the majority of these neighbours [33].

It might seem wrong to put equal weight to all the k nearest neighbours' classes, and a standard extension to KNN is to apply similarity weights to the k nearest neighbours [31]. The weights are applied so that the closer in distance a neighbour is, the more important is this neighbour's class for the final output prediction class.

As indicated before, the size of the neighbourhood has great importance for the coverage and the quality of the predictions of the method employed. Marlin implemented a weighted nearest neighbour algorithm with the Pearson correlation as similarity metric, and elected the active user's k nearest neighbours to compute the predictions. The algorithm was tested with neighbourhood size 1, 10, and 50, and the lowest mean error rates were obtained by just returning the class of one nearest neighbour, namely for k=1.

2.3 Model-based methods

From a probabilistic perspective, the collaborative filtering task can be viewed as calculating the expected value of a rating, given what we know about the user [5]. Model-based algorithms use the database of already given information to estimate or learn a model, which is subsequently used for predictions. The prediction process then consists of computing the expected value of a user prediction, given his ratings on other items [38]. The model building process is performed by different machine learning algorithms such as clustering, Bayesian network, and other modelling techniques. The most important of these techniques are described in this section. In addition, the data mining value received by applying the model-based methods is addressed.

Cheung et al. [7] divide the model-based collaborative filtering techniques into four categories.

- **Cluster models** which assume that the preference ratings form clusters; typically modelled using statistical cluster models.
- **Dependency models** which assume that the preference ratings are independent under certain dependency structures. Methods used to implement these dependency models include Bayesian networks, which are described in this section.
- **Classifier models** which use all the available ratings information (including the "no preference" options) as input to a classifier as well as the existing ratings of the active user as the targeted outputs, to train classifiers to support rating prediction. This corresponds to the approach by Billsus and Pazzani [4] described in the machine learning part of this section.
- **Subspace methods** which project preference ratings onto another subspace generated by singular value decomposition of the matrix of preference ratings. The technique is also called latent semantic indexing (LSI), and is described in the dimensionality reduction technique part of this section.

The section concludes with a description of latent class models, which are closely related both to clustering models and dimensionality reduction techniques.

2.3.1 Cluster models

The general idea behind the cluster models in collaborative filtering is to cluster users with similar or correlated preferences together in one group, such that the active user's

neighbours will be the users belonging to the same group, and these users will be the ones used when computing the active user's predicted ratings. The model is represented as a naive Bayesian classifier where the probability of the users' ratings will be conditionally independent given membership in the group [5]. This model is shown with user 1 in Figure 2.1.

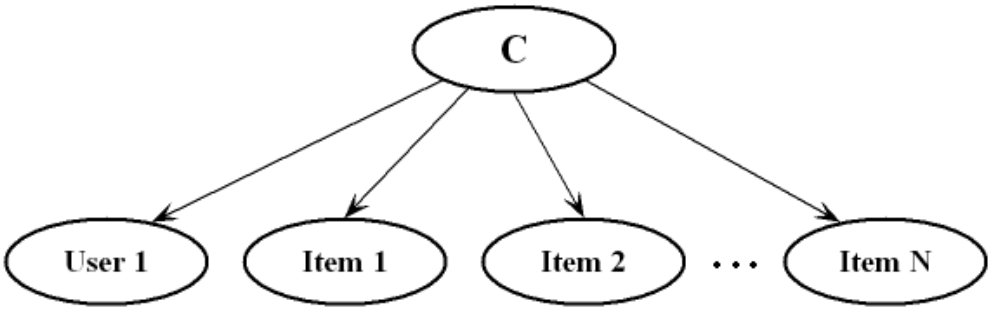


Figure 2.1: Cluster model

`C` is the class variable, with states corresponding to each of the user groups, while `Item 1`,...,`Item N` are the items, with states corresponding to the rating for the given item. We see from the figure that the ID of the user will be independent of his ratings if the class is given. The conditional probability table (CPT) for `Item K` in Table 2.1 illustrates this fact.

Table 2.1: CPT for the variable `Item K` in the cluster model

	`C`		
	group1	group2	...
rating1	p_1	q_1	...
rating2	p_2	q_2	...
...			

In the cluster model, each user is assigned only to one group, that is, each state in `C` represents a distinct set of users. This is the actual *clustering* part of the model. The clusters, or user groups, are unobserved, that is, there is no data telling us which user belongs to which group. The model thus has a *latent* part, an unobserved variable (the clustering of the users) which can be seen as the *cause* for the state at a given item node. It is an important issue how many such user groups there are. Breese et al. selected the number of groups which gave the largest marginal likelihood of the data [5]. When the clusters are established and all users are placed in their corresponding groups, predictions for a user is made by averaging the opinions of the other users in that cluster [38].

There are a number of clustering algorithms in machine learning, and Marlin categorized them into two broad classes; hierarchical clustering and standard clustering [31]. His work on clustering methods was largely based on the survey done by Berkhin in [3].

Standard clustering methods

In the standard clustering methods, there is an iterative optimization procedure that moves the users between the k clusters. In each cluster, there is a representative vec-

tor, called the *cluster prototype*, which is maintained at each step. The optimization procedure works as follows:

1. for each input vector, place it in the cluster which has the cluster prototype most similar to it
2. for each cluster, let the new cluster prototype be the vector in this cluster which is most representative for the cluster

In user clustering, the input vectors correspond to the rows of the user-item rating matrix. The similarity between two vectors is described by a distance function, and the two most popular functions are absolute distance and squared distance. When the absolute distance is used, the method is called *K-medians*, while the *K-means* method uses squared distance. The optimal prototype for a cluster in the case of K-medians is the median of the vectors assigned to that cluster, while for K-means the optimal prototype is given by the mean of the vectors.

Hierarchical clustering methods

The hierarchical clustering methods constructs a tree of clusters [31]. In these cluster trees the parent node contains all the vectors which are partitioned into its children nodes. The construction of the tree is done either bottom-up, starting with each vector partitioned into its own cluster, or top-down, starting with all the vectors in one cluster.

In the bottom-up approach, called *agglomerative clustering*, the process starts with one cluster per vector. For each step, the two most similar clusters are merged. There are several possible linkage metrics to apply for deciding which clusters are more similar, and these include single linkage, complete linkage and average linkage. The merging continues until only one node remains.

The top-down approach, called *divisive clustering*, starts with one single cluster containing all the vectors. New clusters are obtained by iteratively splitting the existing ones. This can be done until each leaf node contains less than a maximum number of vectors, until a maximum number of clusters is reached, or until each cluster satisfies a predefined within-cluster similarity. A standard splitting technique is to randomly pick one vector within a cluster, and find the vector least similar to this. These two vectors are then placed in two clusters, and the remaining vectors in the original cluster are placed in the cluster together with the vector they are most similar to of the two.

Clustering as a preprocessing step

Another often used application of clustering is as a preprocessing step to reduce the dimension of the user-item matrix. It can either be clustering of similar items into groups, to reduce the dimension of the item space to deal with sparse matrices (matrices with mostly empty entries) [31], or it can be clustering of similar users into groups to shrink the candidate set in a nearest neighbour algorithm [38].

O'Connor and Herlocker applied item clustering as a preprocessing step in their experiments [34]. However, their results showed that this actually decreased the quality of the predictions compared to the unclustered base case. However, a reduction in computational complexity was achieved.

2.3.2 Bayesian network model

In the full Bayesian network model, the independence assumption from naive Bayes is removed. The idea is now to learn a network where each node corresponds to an item, so that in the learned network each node will have a set of parent nodes which are the best predictors for its ratings [5]. This is shown in Figure 2.2.

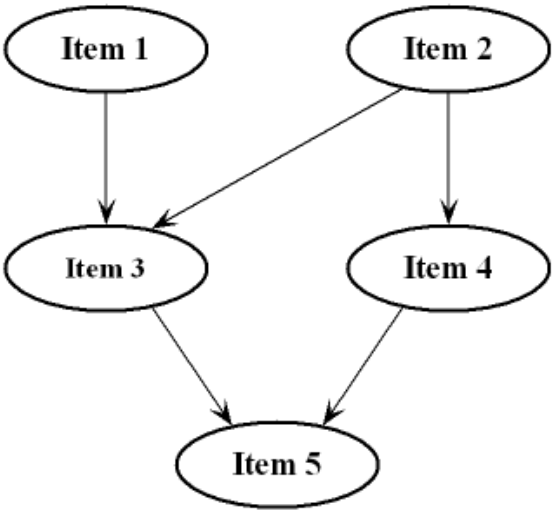


Figure 2.2: Bayesian network model

Again, `Item 1`,...,`Item 5` are items in the domain, with states corresponding to the rating for the given item. Breese et al. also included a "no rating" state for each item [5]. The CPT for `Item 2` in this model is shown in Table 2.2.

Table 2.2: CPT for the variable `Item 2` in the Bayesian network model

	`Item1`			
	no rating	rating1	rating2	...
no rating	p_1	q_1	r_1	...
rating1	p_2	q_2	r_2	...
rating2	p_3	q_3	r_3	...
...				

In the experiments of Breese et al. they represented each CPT as a decision tree. The network structure which yielded the largest marginal likelihood of the data was chosen. Since this likelihood increases the fewer nodes the network contains, Breese et al. used a structure prior which punished each addition of a free parameter.

2.3.3 Machine learning approaches

Marlin considered some classical machine learning approaches suitable for collaborative filtering, which are based on building a model [31]. These include decision trees and artificial neural networks, which will be described here.

Decision trees and artificial neural network

Marlin concluded that application of machine learning techniques such as decision tree classifiers and artificial neural networks (ANN) is possible in collaborative filtering, although there exist some problems with missing values in the input.
For decision trees, the method used in the popular decision tree algorithm C4.5 [33] can be used to deal with missing values. Here, fractional instances are propagated during learning.
However, in the case of neural networks, missing values have to be explicitly represented. There are two different ways to do this [31]:
i) "missing" can be considered as one of the rating values. There are situations when this is sensible, but in the rating data case, it is not justified.
ii) use some sort of feature extraction technique to reduce the data into a compact matrix, which does not contain missing data.

The last approach was the one employed by Billsus and Pazzani [4]. In the training process, they converted the training data, which was a matrix containing a lot of missing entries, into boolean feature vectors, resulting in a matrix filled with zeroes and ones. Thereafter they applied a dimensionality reduction technique called *Singular Value Decomposition*, which is to be explained together with other dimensionality reduction techniques below. In this way, latent structure is utilized, so that two users do not need to rate any common items to become predictors for each other's ratings. Billsus and Pazzani trained an artificial neural network with the reduced data matrix and used this trained network to do the predictions. However, their presented representation of the data allows the use of virtually any machine learning algorithm for collaborative filtering tasks.

2.3.4 Dimensionality reduction

In the aforementioned clustering methods (see Section 2.3.1), it was assumed that there was one single discrete latent variable which was the underlying cause of the observed data. Dimensionality reduction is an analogous technique, only that here we assume a small number of continuous latent variables [31], as shown in the example in Figure 2.3, where `L1` and `L2` are two latent variables or "hidden causes" for the states of `Item 1, 2 and 3`.

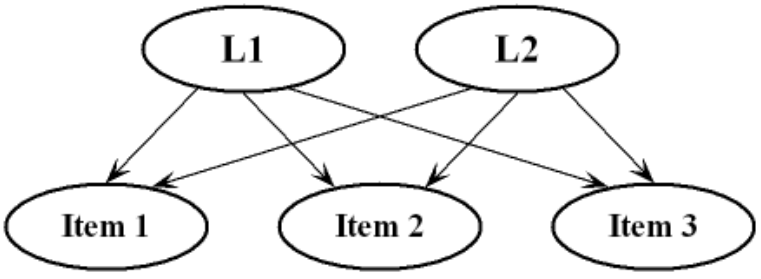


Figure 2.3: Dimensionality reduction model

We wish to utilize this "latent structure" of the ratings, so that users do not have to rate common items to become predictors for each other's preferences [4]. Relationships are captured among customers by reducing the dimensionality of the initial matrices, to increase density and thereby find more ratings [37]. This helps solve two important

problems:

- the problem of *sparsity* - in which we have a matrix with too many empty entries to find useful relationships between users
- the *synonomy* problem - which is more intuitive when dealing with text documents, where it is difficult to capture the similarities between two documents which generally deal with the same topic, if they use different words. In collaborative filtering this problem arises when two users have similar rating patterns, but have not rated any common items.

The dimensionality reduction process can in general be described as mapping a high dimensional input space into a lower dimensional latent space [31].
A brief overview of some dimensionality reduction techniques is given here, and next, applications of these dimensionality reduction techniques within the field of collaborative filtering are described.

Dimensionality reduction techniques

Singular Value Decomposition (SVD) is a dimensionality reduction technique frequently applied in the field of collaborative filtering. The SVD works as follows [37]:

Definition SVD factors an initial matrix R of dimension $m*n$ into three matrices U, S, and V, such that $R = U*S*V'$, where U and V are orthogonal matrices[2] of dimensions $m*r$ and $n*r$, where r is the rank[3] of the matrix r. S is a diagonal matrix[4] of dimension $r*r$ with all the singular values of matrix R as its diagonal entries.

The matrices obtained by performing SVD are particularly useful because it provides the best lower rank approximations of the original matrix R, in terms of the Frobenius norm[5]. It is also possible to make a further reduction by keeping only the k largest diagonal values of the $r*r$ matrix S.

The general idea of this kind of dimensionality reduction is to combine some dimensions so that they depend on more than one term. An example can be the transformation of the vector {(car),(truck),(flower)} into the vector {(1.3452*car + 0.2828*truck),(flower)}.

There are other dimensionality reduction techniques in use, such as *Principal Component Analysis* and *Factor Analysis*. An overview of these methods, as well as of SVD, is given by Marlin in [31].

Applications

In 1988, Furnas et al. [14] proposed Latent Semantic Indexing (LSI) in the information retrieval area to deal with the conflicting challenges of execution time versus prediction quality. LSI uses Singular Value Decomposition to capture latent associations in the

[2] An orthogonal matrix is a matrix which has an equal number of rows and columns, and whose transpose is its inverse

[3] The rank of a matrix is the maximal number of linearly independent rows/columns of the matrix

[4] A diagonal matrix has only entries of zero, except from on the main diagonal (from upper left to lower right)

[5] A matrix norm is a function which assigns a positive length or size to all matrices in the matrix space, and the Frobenius norm comes from an inner product on the space of all matrices

observed data, and has been widely used in information retrieval to deal with synonomy and polysemy[6]. LSI has been utilized in several experiments in Collaborative Filtering.

Sarwar et al. applied LSI to reduce the dimensionality of the customer-product matrix to generate predictions [37]. Neighbourhood-based collaborative filtering algorithms are vulnerable to sparse data matrices, and Sarwar et al. also experimented with the use of a low dimensional representation produced by SVD to compute the neighbourhoods in a neighbourhood-based collaborative filtering algorithm. Due to the fact that the low dimensional representation is much less sparse, they were able to produce more predictions and better predictions.

As previously mentioned, Billsus and Pazzani [4] proposed a method based on dimensionality reduction through SVD of the initial matrix of user ratings. Next, they used the reduced matrix to train a neural network, and used this network to make predictions. However, they claim that with their framework for removing the problem with missing data in the initial data set, virtually any supervised learning algorithm from the machine learning field can be applied instead of neural networks.

In 2006, Symeonidis et al. proposed a novel collaborative filtering algorithm that uses LSI to detect trends and perform recommendations according to them [42].

Even though SVD is the dimensionality reduction technique most frequently applied in Collaborative Filtering, other reduction techniques have also been used. For instance, Goldberg et al. applied Principal Components Analysis to facilitate dimensionality reduction for offline clustering of users and rapid computation of recommendations [16].

2.3.5 Latent class models

In [26], Hofmann and Puzicha introduced a class of algorithms called *latent class models*, where they assumed that there exists some sort of hidden cause(s) for the observed data in some of the variables. In [25], Hofmann used latent class variables in a model-based collaborative filtering technique that among other advantages obtains higher accuracy than the standard memory-based methods. The notions of user communities and item categories are introduced as latent variables, such that for instance a user's belonging to a certain community is the reason why he gave a specific movie a specific rating.

Consider a situation with a number of N users and M items (e.g. movies). To be able to make recommendations to a user based on his preference history, we have to assume that there is an underlying generative process by which users select items. In [26], Hofmann and Puzicha's mixture model formulation assume a latent set of k clusters, which can be thought of for instance as "communities" of users having the same interests. This is shown in Figure 2.4.

Here, the clusters are represented as states in the cluster node `C`, each state in the user node `U` represents one user, and for each item node, the states represent the possible rating values for this item. Formally, each cluster c is a distribution over all the items, assigning probability $P(i|c)$ to each item i, which are the probabilities with which a user in group c will choose each of the items. This means that the clusters can be heavily overlapping. The conditional probability table (CPT) of an item in this model is shown in Table 2.3.

[6]Polysemy is the problem that one single word can have several different meanings

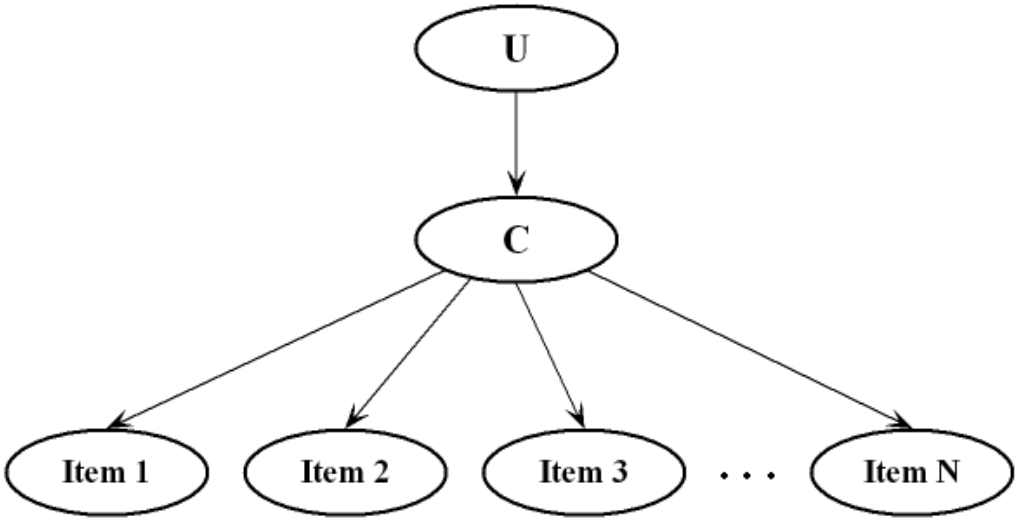


Figure 2.4: Mixture model

Table 2.3: CPT for the variable `Item` k in the mixture model

	`C`		
	group1	group2	...
rating1	p_1	q_1	...
rating2	p_2	q_2	...
...			

Correspondingly, each user is represented by a distribution over clusters, with the probability (or preference) for cluster c denoted by $P(c|u)$. The CPT of the cluster node is shown in Table 2.4.

Table 2.4: CPT for the cluster node `C` in the mixture model

	`User`		
	user1	user2	...
group1	p_1	q_1	...
group2	p_2	q_2	...
...			

The mixture model is an expressive framework for representing the collaborative filtering problem [28]. Although users are grouped into communities, these communities can overlap arbitrarily, and users can have partial membership in many different communities. Similarly, different selections by a single user might require different "explanations" in terms of these communities.

The latent class model (LCM) is a statistical model under the family of mixture models [7]. The predictions are given based on the three variables representing the users, their preferred items, and the preference patterns, as in Figure 2.5.

When there are not explicit ratings, the probability that e.g. a customer u buys a product i can be computed as

$$P(i|u) = \sum_{c' \in C} P(c'|u)\ P(i|c') \tag{2.4}$$

The probabilities $P(i|c)$ are estimated from the observed data, through a learning phase.

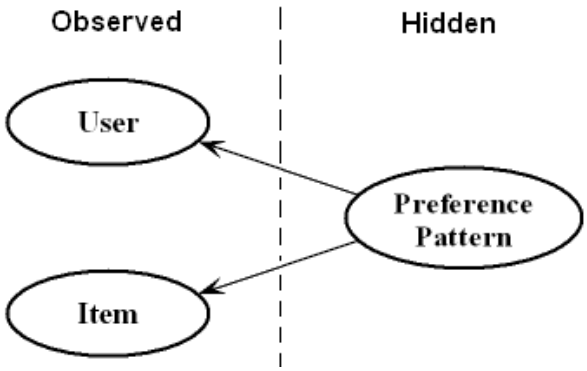
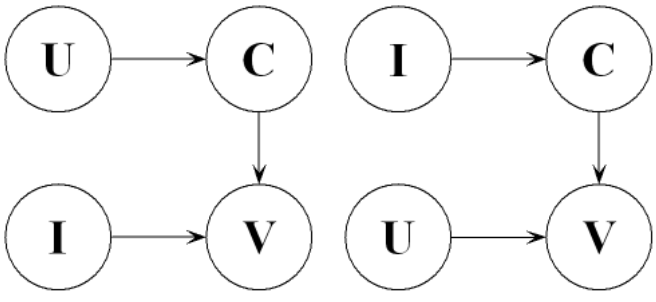


Figure 2.5: The dependency structure of the latent class model (adapted from [7])

In [25] Hofmann used a latent class model approach which is a generalization of a statistical technique called probabilistic Latent Semantic Analysis (pLSA), which was originally developed by Hofmann in the context of information retrieval [22]. The technique is somehow similar to clustering methods in that users are considered as belonging to user communities, but in this case the communities can be overlapping, and users are not partitioned into groups, not even probabilistically [26]. The latent semantic models are in many ways closer related to dimensionality reduction techniques such as SVD and PCA. The main difference compared to these techniques is that pLSA offers a probabilistic semantics and can build on statistical techniques for inference and model selection.

The pLSA model associates a latent variable with every observed rating triplet $\{u, i, v\}$. That means that different ratings from the same user can be explained by different latent causes in pLSA, and user u and item i are rendered conditionally independent given the latent cause. The number of states in the latent variable is considered finite and of size k. If $k = 1$, the model simply assumes that the selection of an item does not depend on the user.

In the case of explicit ratings, Hofmann proposes two different latent class models. This is similar to the latent class model of Cheung et al. in Figure 2.5, but the models of Hofmann are not symmetric models, since he assumes either a latent class for communities of users or a latent class for categories of items. The two different models are shown in Figure 2.6.



(a) User-centric (b) Item-centric

Figure 2.6: Hofmann's latent class models (adapted from [25])

Here, `u` is the user variable, `i` is the item variable, `v` is the rating, and `c` is the latent class. Figure 2.6(a) shows the case in which users are considered as belonging to user

communities, while Figure 2.6(b) shows the case where items are considered as belonging to categories. The proposed model has two ingredients, mixture coefficients - which in the community variant correspond to probabilities $P(c|u)$ - and class-conditional probability distributions $P(v|i,c)$. These distributions correspond to the conditional probability tables shown for the mixture model in Tables 2.4 and 2.3.

In the experiments documented in the article, Hofmann always used the community variant of the model (in Figure 2.6(a)). He also ran experiments with the category variant, but this consistently led to worse results.

Number of hidden states

An important issue in the latent class models is how many states the hidden class should have. As the number of hidden states increases, the set of representable joint distributions over user-item pairs becomes less and less constrained until a fully saturated model is obtained, which can represent any probability mass function over user-item pairs [25]. In practice, the number of states k must be adjusted so that it neither overfits the model nor makes it too inexpressive.
To find an appropriate number of states to use, standard model selection techniques like cross-validation can be used, or it is simply possible to do some experiments with varying k and choose the value of k that yields the best results [7].

2.3.6 Data mining from collaborative filtering models

The extra information that can be extracted from the collaborative filtering process varies with the type of algorithms. Due to the fact that memory-based methods just uses the entire available database for each prediction, they are never able to actually learn anything about relationships that exist between the variables in the database. For model-based methods on the other hand, if the model is properly chosen, the estimated values of the model parameters can provide useful information about relationships and characteristics of users (customers) and items (products) [7]. This type of information can be useful for market analyses and management decision making.

Clustering methods can be used for achieving customer segmentation. Hofmann stated that the decomposition of user ratings in latent class models may lead to the discovery of interesting patterns according to user interest. The communities in his model seemed to constitute themselves around items of either common interest or disinterest.

The ability to automatically discover communities as a part of the collaborative filtering process is a trait that is only shared by a few model-based methods, and can definitely be worth taking into consideration when a decision is to be made about which type of collaborative filtering algorithm to apply for a specific task.

2.4 Evaluation metrics

Researchers within the field of collaborative filtering tend to use varying metrics when listing the results of their experiments. This leads to problems in comparing the various proposed algorithms, since it is hard to identify the best algorithm for a given purpose. Herlocker et al. tried to deal with this problem in their work on collaborative filtering evaluation metrics [20]. They stated that one of the most difficult issues when evaluating

collaborative filtering algorithms is the fact that different algorithms may be better or worse on different data sets. In their survey, they gave an extensive description of the various evaluation metrics applied in the collaborative filtering field, and the overview given here is largely based on their survey.

2.4.1 Accuracy metrics

An accuracy metric empirically measures how close a recommender system's predicted ranking of items differs from the user's true preference ranking. Accuracy metrics may also measure how well a system can predict an exact rating value for a specific item. There are different types of accuracy, including predictive accuracy, classification accuracy and rank accuracy, which are all described here.

Predictive accuracy

Predictive accuracy metrics measure how close the recommender system's predicted ratings are to the true user ratings. For example, MovieLens movie recommender [9] predicts the number of stars (out of five) that a user will give each movie and displays that prediction to the user.

Mean absolute error (MAE) measures the average absolute deviation between a predicted rating and the user's true rating. MAE has been used to evaluate recommender systems in several cases, including by Breese et al. in [5]. Other related prediction accuracy metrics are mean squared error (MSE) and root mean squared error (RMSE) which both squares the error before summing it, such that greater single errors are being punished harder in comparison to smaller errors. Normalized mean absolute error (NMAE) is the mean absolute error normalized with respect to the range of rating values.

The advantages with predictive accuracy metrics are that the computation is simple and easy to understand, and the mean absolute error metric has well studied statistical properties that provide for testing the significance of a difference between the mean absolute errors of two systems.
These metrics are appropriate in the forced prediction case (see Section 2.1.3), however, when a user just wants items that are "good enough", they would not care whether a change in the MAE of 0.01 is achieved. In fact, the MAE is non-intuitive for a user, and not suitable to be handed to a user as a proof of performance of a system. In addition, Herlocker et al. speculated that we might soon be reaching a "magical barrier" where natural variability in the data may prevent us from giving more accurate predictions. This natural variability arises from the fact that the same user may assign a different rating to the same item at different times.

Classification accuracy

In classification of recommendations, the result can be either a good recommendation, or a bad recommendation. Classification metrics measure the frequency with which a recommender system makes correct or incorrect decisions about whether an item is good, and are also referred to as decision support metrics. This type of metrics is thus appropriate when users have true binary preferences, such that they consider an item either relevant/good or irrelevant/bad.

Precision and recall are the most frequently used classification metrics. To measure precision and recall we need a binary rating scale. If the rating scale is numerical, e.g. 1-5 stars, the scale can be transformed into a binary scale by for example converting 1-3 stars into ”bad” and 4-5 stars into ”good”. Precision is defined as the percentage of the selected items that are good items, and represents the probability that a selected item is good. Recall is defined as the percentage of the total number of good items in the system which are selected, and thus represents the probability that a good item will be selected. Precision and recall must be considered together to evaluate the performance of an algorithm completely, and several different measures combining the two has been proposed.

ROC curve-based metrics are an alternative to precision and recall. The ROC model is used to measure to which degree the system can distinguish between signal (good items) and noise (bad items). A ROC curve has the percentage of non-relevant (bad) items selected along the x-axis and the percentage of relevant (good) items selected along the y-axis, such that a random predictor - which is expected to select the same percentage of relevant and non-relevant items - will produce a straight line from the origin to the upper right corner.

Classification metrics make the assumption of binary relevance, and the important issue is to let all relevant items appear before all non-relevant items in the recommendation list; not to distinguish between the degree of ”good” among the relevant items. Classification metrics such as precision and recall can be more intuitive to a user than predictive accuracy metrics; e.g. if an algorithm has a measured precision of 70%, then the user can expect that, on average, 7 out of every 10 items returned will be good items.

Rank accuracy

Rank accuracy metrics measure the ability of a recommendation algorithm to produce a recommended ordering of items that matches how the user would have ordered the same items (cf. top-N recommendations, see Section 2.1.3). Because the predicted rating values create an ordering across the items, predictive accuracy can also be used to measure the ability of a recommender system to rank items with respect to user preference. However, when we want a ranked list, we are not usually interested in exactly how many stars each item would be rated with, as long as the relative ranking of the list is correct, and thus the predictive accuracy metrics are too strict here.

Breese et al. proposed a utility metric for ranked lists, called the *half-life utility metric* [5]. They realized the fact that when a user is presented with a ranked list, he is unlikely to browse all the items in the list. The half-life utility metric attempts to evaluate the utility of a ranked list to the user, and this is done by defining the expected half-life element, that is, the element in the list for which the probability that a user will see it is 50%.

2.4.2 Other performance metrics

Even though accuracy is the metric that has been given the most focus in the collaborative filtering field, these systems must not only provide accuracy, but also usefulness. Usefulness can be defined in terms of how suitable the system is to its users. However, it can be difficult to give a universal definition of suitability. Important issues include coverage, which measures the percentage of the data set for which the system is able to

make recommendations, confidence metrics which help the user to know which systems' decisions he can trust, and novelty/serendipity, which measures in which degree items the user would not consider himself can be recommended.

The user's satisfaction with a system clearly also depends on the performance of the system. Performance metrics of a system include the use of memory, and the computational complexity. Model-based collaborative filtering methods tend to have much smaller memory requirements than memory-based methods as they only require to store a model, not the whole database. However, these models must be learned. The execution of a collaborative filtering algorithm can be divided in two parts, the offline computation and the online computation [25]. The offline component is the part that can be done when the system is not running, e.g. the learning of the models in the model-based methods. The online component is the part of the algorithm that has to be executed while the system is running, when a user request is handed to the system. Since the memory-based methods have to search the entire database for each prediction, they typically require more online computation time than the model-based methods.

2.5 Additional issues

There are a couple of additional issues regarding the collaborative filtering algorithms that are yet to be mentioned, which are addressed here.

2.5.1 Normalization of user ratings

The assumption that all users express their rating on a common scale is invalid in many collaborative filtering scenarios. In the movie rating case for instance, a rating of 3 stars from a user who generally rates all movies with 3-5 stars will not mean the same as a rating of 3 stars from a user who generally rates movies with 1-3 stars. Thus assuming a common rating scale may damage the quality of the recommendations give by the system.

In memory-based methods, the correlation similarity measures such as Pearson or Spearman correlation are normalized to take this into account, while an adjusted cosine similarity measure was introduced by Sarwar et al. to normalize the vector similarity measure (see Section 2.2.1).

In model-based methods, the normalization can be handled by converting the raw user ratings [25]. The usual way to do this is to subtract the mean rating for each user from the user's ratings. This step accounts for the individual differences in the overall "enthusiasm" of the users. The normalized ratings are used for the training of the model, and when a prediction is to be made, the active user's mean rating must be added to the computed prediction to get an output prediction to present to the active user [5].

For users with a small number of ratings, care should be taken when estimating the user-specific mean rating. An example of this problem can be a user who has rated only two movies in the system, both with 5 stars. But if these two movies are actually the best movies this user has ever seen, the system will wrongly assume that this user always rates items very high. Hofmann proposed a smoothing scheme to appropriately smooth the ratings of users with few ratings [25].

2.5.2 The missing at random assumption

When there are data sets with large amounts of missing data, it is important to consider the process that causes the data to be missing [31].

If the probability that a variable is unobserved given the values of all variables is equal to the probability that a variable is unobserved given just the observed variables, then the data is said to be *missing at random* [30]. In this case we can just ignore the causes for the missing data, as the missing data mechanism will not affect the recommendation computations.

However, it is often not the case that data is missing at random. In the movie rating case for example, users only rate movies they have seen, and they tend to watch movies that they think they will like. If the data is not missing at random, ignoring the missing data mechanism can bias maximum likelihood estimates which are computed from the data to make predictions or learn a model.
Chen and George constructed a data set with an extreme non-random pattern of missingness, and performed experiments with this data set [6]. They found that the Bayes approaches substantially outperformed the neighbourhood-based methods in those experiments. It appeared that the Bayes methods were more robust against such extreme missingness because they weight the extent of matches between users rather than correlated patterns.

2.5.3 The first-rater problem

As indicated throughout this chapter, data sparsity is generally a challenge for all collaborative filtering methods. An extreme form of sparsity arises when new products are introduced into the market, for which there exist no previous ratings. This is called *the first-rater problem.* In such cases either some ratings have to be collected, or content-based information has to be utilized for the system to be able to make recommendations. When content-based features are included in the model, the formulation is sometimes called hybrid collaborative filtering [31]. This additional content information can be age or gender of the user, and item-specific information such as an author for books or genre for movies.

The GroupLens researchers experimented with adding personal filtering agents to the collaborative filtering system, to be able to utilize content-based information in the recommendation making process [17], [39]. Basu et al. used hybrid features; collaborative features influenced by content information [2], while Condliff et al. tried to include all the content-based as well as collaborative information in a single probabilistic model [8]. Newer work include Melville et al., who ”content-boosted” a traditional neighbourhood-based collaborative filtering algorithm [32], and Schein et al. who tried to solve the first-rater problem in a movie setting by letting movie actors act as ”surrogates” for movies when the movie information was not available [40].

Cheung et al. pointed out that integrating content-based information with collaborative information when using the latent class model is an interesting direction for further research [7], and Herlocker et al. believed that collaborative filtering must be combined with existing content-based information filtering technology to reach its full potential [20].

Chapter 3

Evaluating Collaborative Filtering Approaches

As one of the goals of this project (see Chapter 1 for a listing of the project goals) is to design a model for collaborative filtering, this chapter uses the information from the previous chapter, as well as the underlying conditions for the project, to evaluate what kind of model would be appropriate in this context. The collaborative filtering algorithms outlined in the previous chapter are reviewed with respect to the data set, and to the problem definition and goals of the project.

3.1 Characteristics of the data set

As described in Section 2.1.5, the sparsity level of a database is given by the ratio of missing entries to the total number of entries. The MovieLens data set which is utilized in this project was introduced in Chapter 1. It contains 97914 explicit ratings on a scale from 1 to 5 given by 925 users on 1682 movies. The sparsity level of this database is thus

$$1 - \frac{97914}{925 * 1682} = \underline{0.937}$$

by the equation presented in Section 2.1.5, which means that for every 1000 entries in the user-movie matrix, 937 of them are missing entries.

The data set contains more information than just the users, movies and ratings. It also contains demographic information about the users, namely age, gender, occupation, and zip code, and movie information, namely title, release date, url to the entry in the Internet Movie Database IMDb (www.imdb.com), as well as 19 genres, which for each of them the movie might either be in that genre or not. It is thus possible for a movie to have several genres. In addition, there are timestamps for the time the ratings were given.

3.2 Collaborative filtering algorithms with respect to the project setting

According to the problem definition and goals in Chapter 1, one of the main objectives of the project is to design a model for doing collaborative filtering of movies on the

MovieLens data set. This section reviews the most important issues of collaborative filtering algorithms outlined in the previous chapter with respect to the project setting, and thereby makes some of the main choices of what kind of model(s) to build.

From the previous chapter, it is clear that sparse databases and lack of enough adequate ratings is a problem for the quality and coverage of recommendations of collaborative filtering algorithms. In Section 2.1.5, it was explained why this is especially a problem for the memory-based methods, and that model-based methods are less hurt by small numbers of commonly rated items, as the relationships between users are computed based on the rating *patterns* rather than ratings on common items. This also implies that it is easier to get 100% coverage (see Section 2.4.2) with the use of model-based algorithms.

As shown in the previous section, the MovieLens data set has a sparsity level of 0.937, which means that more than 93 out of 100 entries in the user-movie rating matrix are empty. This fact, together with the observations emphasized above, implies that the most appropriate in this project is to choose a model-based collaborative filtering algorithm.

To achieve better quality in the recommendations whenever data is sparse, Section 2.3.4 explained how it is possible to utilize "latent structures" of the ratings. Relationships between users are captured by reducing the dimensions of the initial data matrix, so that the density of the data increases. As described, this helps solve both the sparsity problem and the synonomy problem.
The latent structures of the movie ratings can be thought of as hidden causes of why the users rate the way they do. In Section 2.3.5, it was explained how latent class models are able to model such hidden causes. Users are grouped into communities, which can overlap arbitrarily, such that users can have partial membership in many different communities. That means it is possible to use different "causes" for different selections done by the same user.

It is natural to imagine that a database of movie ratings contain latent structures, or hidden "causes" for why the users rate the way they do. It is also likely that there can be different explanations for selections done by the same user. For instance, a user may choose one movie because she is a woman and this is a movie generally liked by women, and the same user may choose another movie because she is interested in sports and the second movie is a movie generally preferred by sports people. Based on these facts and the descriptions given in the previous chapter, latent class models are chosen to build the model(s) of this project.

Hofmann used latent class models to model such hidden causes. There was one "user-centric" model where he grouped users into communities (see Figure 2.6(a) in the previous chapter), to capture the explanations for why users rate as they do. Hofmann also experimented with an "item-centric" model (see Figure 2.6(b)) which grouped items into categories, but concluded that its performance was consistently worse than for the user-centric model.
However, a typical rating explanation scenario within the movie rating area would be that a user u liked movie i because u was a woman, i was a romantic movie, and women generally like romantic movies. Thus, the more intuitive representation of the explanations for ratings would be both a grouping of users into communities and a grouping of items into categories at the same time.

To be able to capture this scenario, a latent class model with both a latent variable for grouping movies into categories, and a latent variable for grouping users into com-

munities is chosen. As this model utilizes elements from both the user-centric and item-centric models of Hofmann (see Section 2.3.5), it will be referred to as the "best of both worlds" (BOBW) model throughout this thesis.

We already know that data sparsity is a general problem in collaborative filtering. In Section 2.5.3, an extreme variant of the sparsity problem, the first-rater problem, was introduced. To achieve better quality in the case where there is a lack of adequate data, several researchers have proposed that collaborative filtering should be combined with existing content-based information filtering techniques to be able to utilize available content information, such as age or gender of the user, or item-specific information such as genre for movies.
Some research has been done in the area, but significant advantages are yet to be established. One of the biggest problems with the inclusion of content information into the collaborative filtering algorithms seems to be how to represent the content information to be able to include it into the collaborative filtering algorithm.

According to the project goals presented in Chapter 1, this project is to do research on how the inclusion of content information into the model can affect the quality of predictions. The need to find better ways to achieve higher quality of the predictions when presented with the first-rater problem, and the sparsity problem in general, described in the previous chapter and evaluated here, justifies the research on this issue. The rather feeble previous proposals to how such information should be included into a collaborative filtering model, seeks for new simpler and more intuitive approaches.

Chapter 4

Model Choices and Implementation

Based on the decisions made in the previous chapter, this chapter gives the details of the designed models, of how the learning and implementation is carried out, and a detailed description of the experimental methodology is provided, as well as the evaluation metrics with which the experiments are to be evaluated.

4.1 Model representation

The latent class models which are to be built, will be represented as Bayesian networks. Using such networks is an intuitive way to model a domain, as for instance an arrow from a node `Category` into the node `Rating` means that the category somehow influences the rating. The use of Bayesian networks for representing the models conforms with the project goal in Chapter 1 which states that the model(s) should be simple, intuitive and easily representable.
In addition, as stated in Chapter 3, the inclusion of additional content features into the model should also be simple and intuitive; and later in this chapter it is shown that this is in fact the case when using Bayesian networks to represent the models.
There are certain useful properties which we can take advantage of when working with Bayesian networks. Among these is the fact that when computing the probabilities for each state in a node, we need only look at the node's own probabilities, and the probability for the combination of states of its parents. This means that the problem of computing probabilities for the whole model can be split into smaller subproblems. How this advantage is utilized, will be made clearer in Section 4.3, where the learning of the models is described.

4.1.1 The rating node

Since the data set with which we are working has a rating scale of 1 to 5 stars, the `Rating` node must be able to model a 5 star rating scale. A labelled node with five states named ”1” through ”5” is possible, but this way the model will not be able to capture the fact that rating value 4 is closer to rating value 5 than what rating value 1 is, as the program can not recognize that simply by the names of the states.

Consequently, there are two options remaining. One is to use a node with five states named ”1” through ”5”, and give the states an ordering according to which state is closer to one state than another, namely an *ordinal* node. The other option is to use a continuous node, and map the 5 scale ratings into the continuous scale. The last option has an advantage when we use normalized ratings, as the input rating in the training set will typically not be an integer. In addition, using a continuous rating scale gives the possibility of outputting predictions on an integral scale, for instance ”3.5”. These two facts results in the use of a continuous node to model the `Rating` node in all the models in this project.

Instead of states, a continuous node has for each combination of its parents' states a normal distribution representing the probability density for the various outputs for the given combination of the parents' states. The normal distribution is given by a mean value and a variance value. Figure 4.1 shows a normal distribution (the solid line) with mean 1 and variance 1. The x-axis represents the rating values, while the y-axis represents probability density.

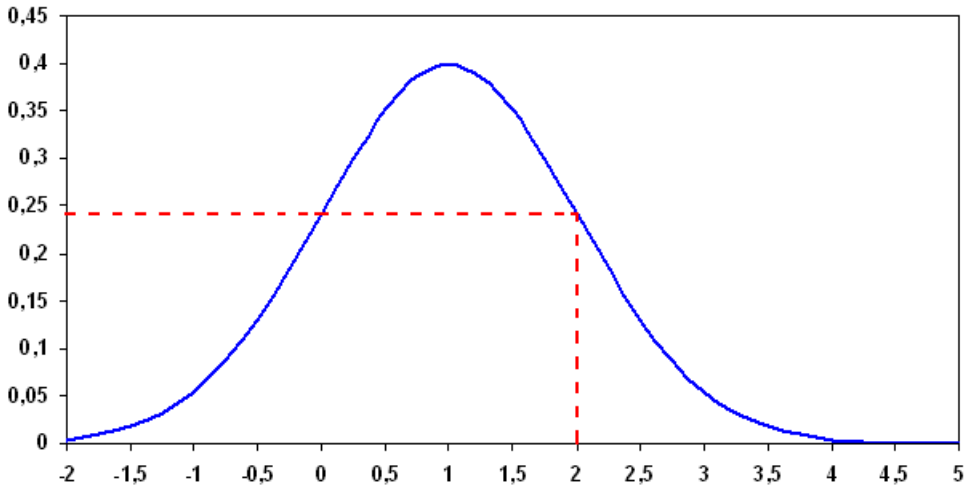


Figure 4.1: Normal distribution with mean 1 and variance 1

We see that with this normal distribution there is a probability density of a little less than 25% for a rating value of 2 (emphasized by the stapled line in the figure).

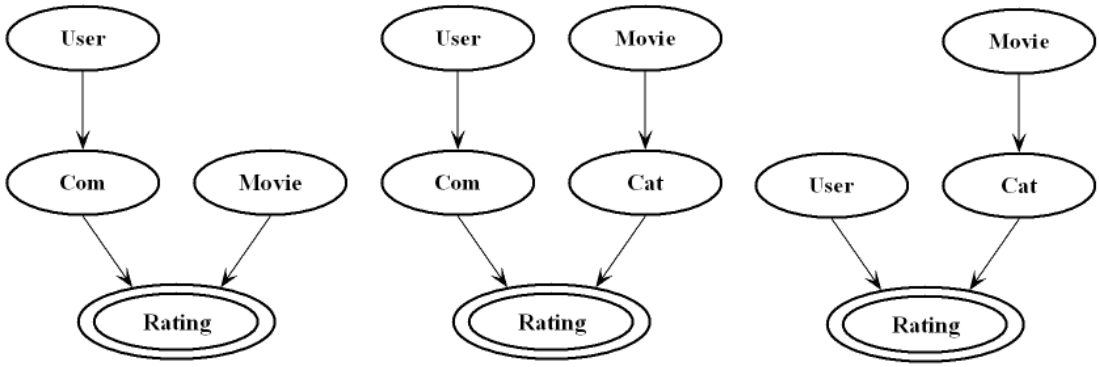
4.2 Model structure

This section describes the structure of the models that were built, that is, the attributes included, the relationships between the attributes, and the possible states of each attribute.

4.2.1 ”Best of both worlds” model vs. user-centric and movie-centric models

As described in Section 2.3.5, Hofmann proposed two different latent class models for explicit ratings, namely a user-centric model with a latent class describing user communities, and an item-centric model with a latent class describing item categories [25]. As commented in Chapter 3, latent class models will be used throughout the

project, and thus the straightforward model containing only a user, movie and rating node without any latent classes is not tested here. By including both a latent class for grouping users into communities and a latent class for grouping movies into categories in the same model as explained in Chapter 3, it will hopefully capture the elements from both the user-centric and item-centric models. It is therefore referred to as the "best of both worlds" (BOBW) model, and is shown in Figure 4.2(b). A given part of the BOBW model testing will be to perform experiments to compare it to the user-centric model, shown in Figure 4.2(a), and the item-centric model - from now on referred to as the movie-centric model - shown in Figure 4.2(c).



(a) User-centric (b) BOBW (c) Movie-centric

Figure 4.2: User-centric, BOBW, and movie-centric models

In these three models, the `User` node has 925 states, one for each user in the database, the `Movie` node has one state for each of the 1682 movies, and the number of states in `Community` and `Category` is varied, although for simplicity, for all models, the number of states is always equal in the two latent nodes. As explained in the previous section, `Rating` is going to be modelled as a continuous node, and hence has a mean and variance value for each combination of its parents, rather than states.

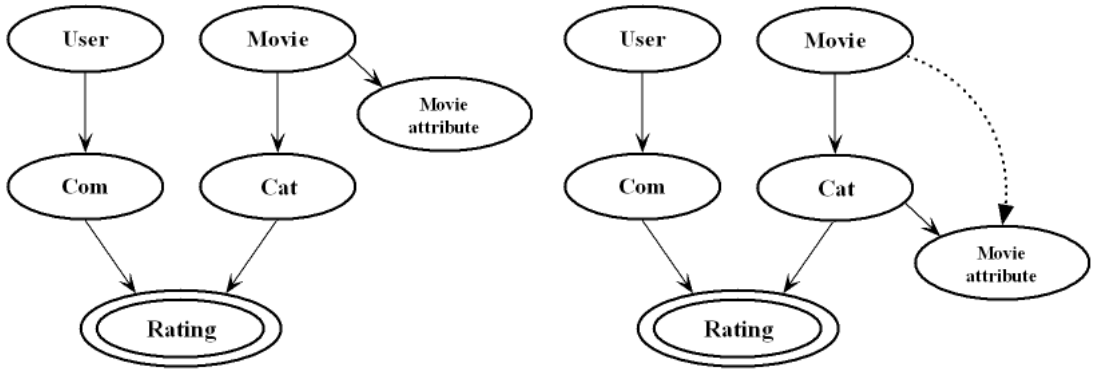
Comparisons will be given of the performance of the three different models in Figure 4.2. Nevertheless, the BOBW model is the "main" model of this project, and is the one used in the rest of the experiments.

4.2.2 Including additional features

As described in Chapter 3, the data set contains demographic information about the user, as well as additional information about the movies. As given from the project goals in Chapter 1, as well as the evaluation in the previous chapter, this project is to do research about including additional content features into the models.
When running the full data set through an attribute selection process[1], it turned out that user age, zip code, and movie release year were the most significant attributes, except from user ID and movie ID. Hence, experiments were executed including each of these three in the BOBW model. The experiments included one additional feature at a time, to be able to see each additional feature's contribution to the performance of the model. The director and leading role actor were also included, as explained below.

[1] Weka, version 3.4.8, was used for this process, with `InfoGain` as attribute evaluator and `Ranker` as search method

When including an additional movie attribute in the model, the "correct" way to include it would be to make it a child node of the `Movie` node, as the observation of movie would give us directly what the value of this attribute is. This model is shown in Figure 4.3(a).



(a) "Correct" inclusion (b) Useful inclusion

Figure 4.3: The inclusion of a movie attribute in the model

However, as both the movie and the movie attribute are observed in our data set, this gives no new information. What we are interested in, is to know more about how the movies should be grouped into categories. So, to let the extra available information (in this case, the movie attribute) help us with that, we instead include the movie attribute node as a child of `Category`, as shown in Figure 4.3(b) (where the stapled line shows the relationship which is no longer a part of the model). That way the additional feature influences the network, and at the same time it does not change any of the existing relations between the other variables. Demographic user features were included in the same way, but with `Community` as a parent instead of `Category`, and therefore influence the grouping of users into communities.

Even though the model in Figure 4.3(b) might not be the obvious choice of how to represent the world we are dealing with here, it is still an intuitive way to think of the additional features. For instance, you would expect the movie release year to have something to do with how the movies are categorized, and similarly, you would expect the user age to affect how users are grouped into communities.

Including additional features into the network in this way would not have been possible for demographic user features in the movie-centric model, and not for movie features in the user-centric model. Thus, the BOBW model has an advantage over these two when it comes to the possibility of inclusion of additional features, as the BOBW model can include both user features and movie features in the same model.

For each additional feature, it is explained below how the data was represented in the model.

Movie release year

In the data set used, the earliest movie release year for any movie is 1922, and the latest year is 1998. To make the movie year attribute easier to handle, it was collected into four groups, namely group 1: 1974 or earlier, group 2: 1975-1989, group 3: 1990-1995, and group 4: 1996 or later. The movie release years not available were simply registered as missing data. This distribution of years into groups was done manually with

a certain feeling of number of movies belonging to each group, without any mathematical basis. However, a distribution of movie years into 9 groups was also tested, and the four groups distribution was consistently better, and is thus used throughout the experiments, giving a `Year` node with four states. The BOBW model with the `Year` node included is shown in Figure 4.4.

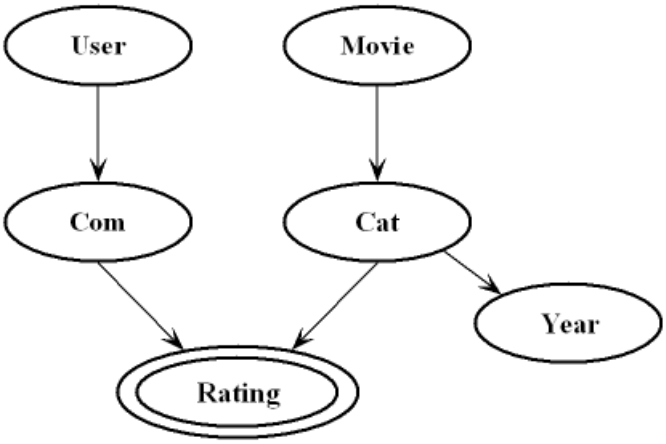


Figure 4.4: BOBW model with movie release year

Directors and actors

In the data set, for each movie, an url to its information on the site of the Internet Movie Database IMDb is given. This information includes who directed the movie, plot outline, names of the actors acting in the movie, and duration of the movie. As mentioned in Section 2.5, Schein et al. did experiments where they let movie actors act as "surrogates" for movies when the movie information was not available [40]. They got interesting results in some situations, and hence the inclusion of actors as an additional feature in this project was worth to be tested. The cast of the movie given at IMDb is ordered by participation in the movie, such that the leading role is listed first. For simplicity only the leading actor/actress was used. The director of the movie was also included in the same model, and the BOBW model including a `Director` and a `Actor` node is shown in Figure 4.5.

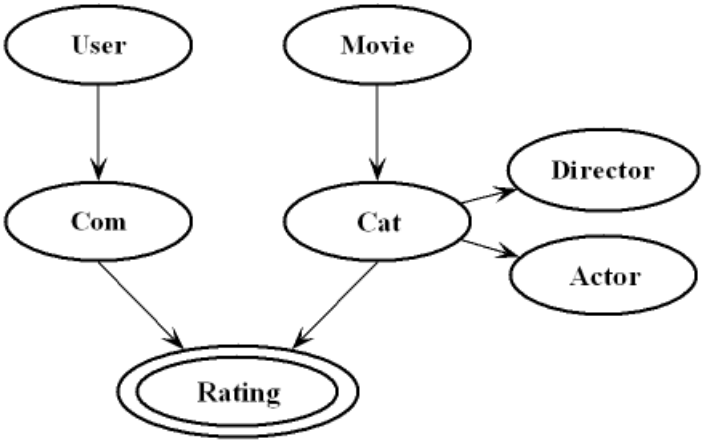


Figure 4.5: BOBW model with directors and actors

Obviously, for 1682 movies, there are bound to be great variation in the director and leading role in the movies; an issue that complicates the model. However, we realize the fact that a director or actor appearing only in one movie will not give any informational value, as we already have the movie ID to identify the relationship between two data cases both containing this director or actor. Consequently, only directors and actors appearing in more than one of the 1682 movies are included. This leaves us with 389 directors and 242 actors, and thus 389 and 242 states in the `Director` and `Actor` nodes respectively. The movies with a director or actor appearing only in the one movie, is registered with missing values for those two attributes.

Zip code

The zip codes in the data set each contain five digits[2]. To simplify the treatment of the zip codes, they were split into ten groups according to the first digit, as the first digit gives an indication of the main area in which the user is situated[3]. The `Zip` node thus has 10 states, and the BOBW model with zip code included is shown in Figure 4.6.

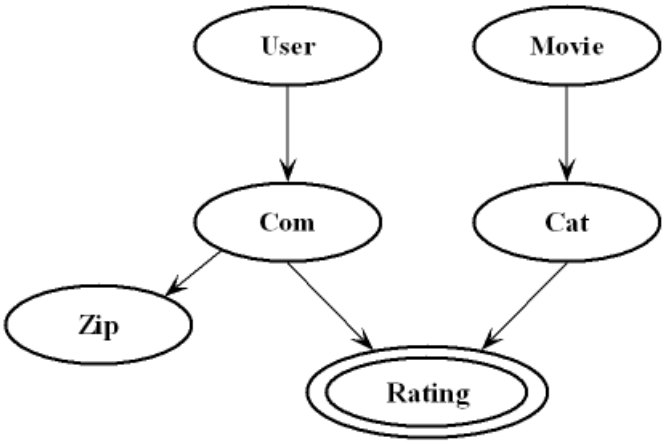


Figure 4.6: BOBW model with user zip codes

It is worth noting here that it is estimated that a quarter to a third of the users in the data set are not from the USA [1]. These users will thus represent ”noise”, as a zip code starting with ”3” in the USA will probably not have a correlation with a British zip code starting with ”3”. However, since the majority of users are American, the zip codes will hopefully still represent valuable information.

User age

The lowest user age for any user in the data set is 7 years, and the highest user age is 73 years. The user age attribute was collected into four groups, namely group 1: 19 years or younger, group 2: 20-34 years, group 3: 35-49 years, and group 4: 50 years or older. Where user age was not available, it was simply registered as missing data. Hence, the `Age` node has four states. The distribution of age into groups was done in

[2]As explained in Chapter 1, the 18 users who had stated zip codes containing letters were removed from the database

[3]For example, in the USA, the states in the west have zip codes starting with 9, the states in the south east have zip codes starting with 3, and so on

the same way as with movie release year explained above. The BOBW model with the `Age` node included is shown in Figure 4.7.

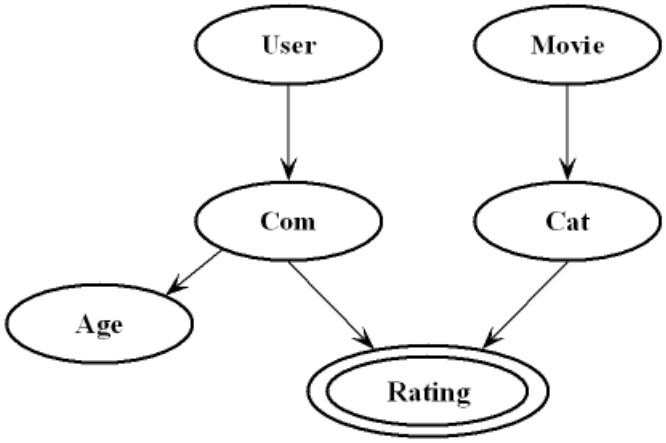


Figure 4.7: BOBW model with user age

4.3 Model learning

To learn the model means to adjust the parameters in the model so that it best fits the training data. One of the most popular algorithms for doing parameter estimation is the Expectation-Maximization (EM) algorithm [11], and it was the algorithm used for the model learning in this project.

The EM algorithm is a general algorithm for finding maximum likelihood estimates[4] for a set of parameters when faced with an incomplete data set, that is, when there are missing values in the data set. That the algorithm is able to handle incomplete data well is obviously important here, as there are two latent variables in the model for which there are no observed data at all.

The algorithm basically alternates between the so-called *expectation step* and the *maximization step*. In the expectation step the data set is "completed" by using current parameter estimates to calculate expectations for the missing values. The *maximization* step use the "completed" data set to find a new maximum likelihood estimate for the parameters. In the next iteration of the algorithm, the new estimate is used to complete the data set in the next iteration of the algorithm.

The plain "best of both worlds" (BOBW) model from the previous section will be used to illustrate the model learning performed in this project. The model is redrawn in Figure 4.8.

For the network in Figure 4.8, we would get the following expression for the joint distribution (using a topological ordering from top to bottom in the graph) (the abbreviations User=U, Movie=M, Community=Com, Category=Cat, and Rating=R will be used where suitable throughout this section):

$$P(U,M,Com,Cat,R) = P(U)P(M|U)P(Com|U,M)P(Cat|U,M,Com)P(R|U,M,Com,Cat).$$

[4] A maximum likelihood estimate is the choice of parameters that makes the observed data most likely, and it is a standard method for statistical inference that can be used to (approximately) maximize the log-likelihood in mixture models like latent class models [25]. That is, the choice of parameters is made which best explains the data.

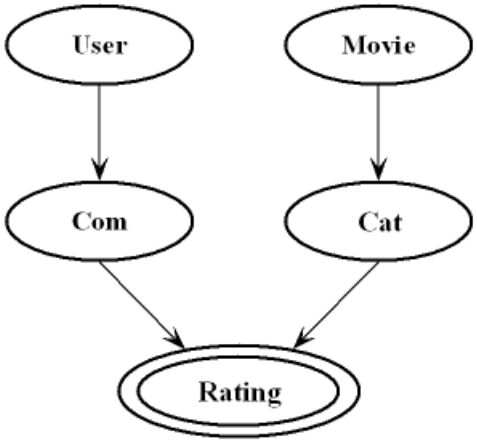


Figure 4.8: Plain BOBW model, redrawn

However, as we are working with a Bayesian network, there are some independence assumptions which can be utilized.
Generally, in Bayesian networks, the probability distribution of a variable depends only on the parent set. The parent set $Pa(v_i)$ of a node v_i is the set of all parents of the node in the network. Thus, the joint distribution in a Bayesian network can be specified as follows.

Definition The joint distribution in a Bayesian network is the product of the conditional probabilities of all variable nodes

$$P(v_1 ... v_n) = \prod_{i=1}^{n} P(v_i \mid Pa(v_i))$$

With the current network structure, this means that $P(M|U) = P(M)$, $P(Com|U,M) = P(Com|U)$, $P(Cat|U,M,Com) = P(Cat|M)$, and $P(R|U,M,Com,Cat) = P(R|Com,Cat)$, and the joint probability distribution of the network in Figure 4.8 can be specified as follows.

$$P(U,M,Com,Cat,R) = P(U)P(M)P(Com|U)P(Cat|M)P(R|Com,Cat) \tag{4.1}$$

To learning the network, we have to estimate these conditional probability distributions so that they best fit the given training data. A simple example is used in this section, and the training data used in the example is given in Table 4.1.

Table 4.1: Training data

Data case	User	Movie	Rating
D_1	1	3	2
D_2	1	2	3
D_3	2	4	5
D_4	2	1	2
D_5	3	2	4
D_6	3	3	4
D_7	4	4	4
D_8	4	2	4

Thus, we have eight given data cases, each with observations for the states in `User`, `Movie`, and `Rating`.

4.3.1 Initialization

Several iterations of expectation and maximization are needed for the learning, but before we start, we have to specify some initial probability distributions, that is, initial values in the conditional probability tables (CPTs) of each node. In this example, `User` and `Movie` each have four states, and are initiated to the uniform values shown in Table 4.2, `Community` and `Category` each have two states, and are initiated to the random values[5] shown in Table 4.3 (conditioned on the state of `User` in the case of `Community`, and on the state of `Movie` for `Category`). The `Rating` node is a continuous node, and hence has mean and variance values which say something about how the normal distribution is given the combination of its parents, instead of straight conditional probabilities. The initial mean and variance values of `Rating` is shown in Table 4.4.

Table 4.2: Initial CPTs for the `User` and `Movie` nodes

State	Probability
1	0.25
2	0.25
3	0.25
4	0.25

Table 4.3: Initial CPTs for the `Community` and `Category` nodes

	State of `User`/`Movie`			
	1	2	3	4
1	0.74	0.43	0.69	0.15
2	0.26	0.57	0.31	0.85

Table 4.4: Initial CPT for the `Rating` node

Category	1		2	
Community	1	2	1	2
Mean	1.0	1.5	3.2	4.6
Variance	1.0	1.0	1.0	1.0

As explained before, `Community` and `Category` are latent (hidden) variables, for which there is no data observed. Had the database been complete (that is, had there been no missing data), then in order to find a new estimate for a distribution $P(X = a)$ for a variable X, we would simply count the number of cases $N(X = a)$ in the training set with $X = a$:

$$P(X = a) \;=\; \frac{N(X = a)}{N} \tag{4.2}$$

[5]For the latent variables it is important that they are not initiated uniformly, as it in that case is impossible to infer anything about belonging of users to different communities, or movies to different categories, as there will be no preferences for one latent class state over another

However, when we have incomplete data, the parameters must be estimated by their *expected* value whenever data is missing, that is, we "complete" the data set as explained above. We know the following regarding the estimated probability $\hat{P}$:

$$\frac{N(X=a)}{N} \;\leq\; \hat{P}(X=a) \;\leq\; \frac{N(X=a,?)}{N}$$

That is, the fraction of cases in which $X=a$ can not be less than the fraction of cases in which we have actually observed $X=a$, and it can not be more than the number of cases in which we have observed $X=a$ plus the fraction of cases in which we have not observed a value for X. To find the estimate $\hat{P}(X=a)$, we have to find the expected count for how many cases which contain $X=a$, and divide it by the total number of cases N, as stated in Equation 4.3.

$$\hat{P}(X=a) \;=\; \frac{E[N(X=a)]}{N} \tag{4.3}$$

4.3.2 User CPT computations

The `User` variable is an observed variable, and there is no missing data in the training data set for this variable, thus, to find the estimates $\hat{P}(U=a)$, we simply have to count the number of cases in the data set in which $U=a$, as specified in Equation 4.2.

We compute $\hat{P}_1(U)$ for the four states:

$$\hat{P}_1(U=1) \;=\; \frac{2}{8} \;=\; \underline{0.25}$$

$$\hat{P}_1(U=2) \;=\; \frac{2}{8} \;=\; \underline{0.25}$$

$$\hat{P}_1(U=3) \;=\; \frac{2}{8} \;=\; \underline{0.25}$$

$$\hat{P}_1(U=4) \;=\; \frac{2}{8} \;=\; \underline{0.25}$$

Since the probabilities add up to 1, for each node we have to compute only $Y-1$ probabilities, where Y is the number of states. The Yth probability can be computed by subtracting the first $Y-1$ probabilities from 1. For instance, $\hat{P}_1(U=4)$ could have been computed by:

$$\hat{P}_1(U=4) \;=\; 1 \;-\; 0.25 \;-\; 0.25 \;-\; 0.25 \;=\; \underline{0.25}$$

4.3.3 Movie CPT computations

The same as above is done for $\hat{P}_1(M)$, and we get:

$$\hat{P}_1(M=1) \;=\; \frac{1}{8} \;=\; \underline{0.125}$$

$$\hat{P}_1(M=2) \;=\; \frac{3}{8} \;=\; \underline{0.375}$$

$$\hat{P}_1(M=3) \;=\; \frac{2}{8} \;=\; \underline{0.25}$$

$$\hat{P}_1(M=4) \;=\; \frac{2}{8} \;=\; \underline{0.25}$$

4.3.4 Community CPT computations

For $\hat{P}_1(Com|U)$, we have to compute the probabilities for each possible parent state, that is, for each state of `User`. Since there are four users, the number of states Y is four, and three probabilities have to be computed.

We start with the computation of $P(Com = 1|U = 1)$. The probability is found by dividing the count for the number of cases in which $Com = 1$ and $U = 1$, on the count for the number of cases in which $U = 1$. However, as described before, `Community` is a hidden variable with no observed data, and thus, as given in Equation 4.3, we have to estimate the *expected* count for how many cases in which $Com = 1$ and $U = 1$.

$$\hat{P}(Com = 1|U = 1) \;=\; \frac{E[N(Com = 1, U = 1)]}{N(U = 1)} \tag{4.4}$$

Since U is observed, the expected count $E[N(Com = 1, U = 1)]$ is found by looking at each case in the training set where $U = 1$, and for each of these compute the probability that $Com = 1$ in that case. We see from the training data in Table 4.1 that there are two cases in which $U = 1$, namely in the two first cases. Thus, the expected count for $Com = 1, U = 1$ is found by computing the probability for $Com = 1$ in these two cases:

$$\begin{aligned} &E[N(Com = 1, U = 1)] \\ &= \hat{P}(Com = 1|U = 1, M = 3, R = 2) \;+\; \hat{P}(Com = 1|U = 1, M = 2, R = 3) \end{aligned} \tag{4.5}$$

From Equations 4.4 and 4.5 we can now find the first estimate $\hat{P}_1$ for the probability that $Com = 1|U = 1$, as stated in Equation 4.6.

$$\begin{aligned} &\hat{P}_1(Com = 1|U = 1) \\ &= \frac{\hat{P}_1(Com = 1|U = 1, M = 3, R = 2) \;+\; \hat{P}_1(Com = 1|U = 1, M = 2, R = 3)}{2} \end{aligned} \tag{4.6}$$

We have to find $P(Com = a|U = b, M = c, R = d)$, and an expression for this probability, given by Bayes' law, is shown in Equation 4.7.

$$\begin{aligned} &P(Com = a|U = b, M = c, R = d) \\ &= \frac{P(U = b, M = c, Com = a, R = d)}{P(U = b, M = c, R = d)} \\ &= \frac{\sum_{Cat} P(U = b, M = c, Com = a, Cat, R = d)}{\sum_{Com,Cat} P(U = b, M = c, Com, Cat, R = d)} \end{aligned} \tag{4.7}$$

1. We start with the computations for the first term in the numerator in Equation 4.6, $\hat{P}_1(Com = 1|U = 1, M = 3, R = 2)$, and by Equation 4.7 we get:

$$\hat{P}_1(Com = 1|U = 1, M = 3, R = 2) \;=\; \frac{\hat{P}_1(U = 1, M = 3, Com = 1, R = 2)}{\hat{P}_1(U = 1, M = 3, R = 2)} \tag{4.8}$$

   (a) We now look at the numerator in Equation 4.8, $\hat{P}_1(U = 1, M = 3, Com = 1, R = 2)$.

$$\hat{P}_1(U = 1, M = 3, Com = 1, R = 2) \;=\; \sum_{Cat} \hat{P}_1(U = 1, M = 3, Com = 1, Cat, R = 2)$$

From Equation 4.1, we get:

$$\begin{aligned}&\sum_{Cat}\hat{P}_1(U=1,M=3,Com=1,Cat,R=2)\\ =&\sum_{Cat}(\hat{P}_1(U=1)\hat{P}_1(M=3)\hat{P}_0(Com=1|U=1)\hat{P}_0(Cat|M=3)\hat{P}_0(R=2|Com=1,Cat))\end{aligned} \tag{4.9}$$

Note that the latest computed estimates $\hat{P}_1(U)$ and $\hat{P}_1(M)$ for user and movie are used in the computation. Further, from Equation 4.9, we get:

$$\begin{aligned}&\sum_{Cat}\hat{P}_1(U=1,M=3,Com=1,Cat,R=2) \;=\\ &\hat{P}_1(U=1)\hat{P}_1(M=3)P_0(Com=1|U=1)P_0(Cat=1|M=3)P_0(R=2|Com=1,Cat=1)\\ +\;&\hat{P}_1(U=1)\hat{P}_1(M=3)P_0(Com=1|U=1)P_0(Cat=2|M=3)P_0(R=2|Com=1,Cat=2)\end{aligned} \tag{4.10}$$

All of these subexpressions are now possible to find from the table entries. $\hat{P}_1(U)$ and $\hat{P}_1(M)$ have been computed earlier, $P_0(Com|U)$ and $P_0(Cat|M)$ can be found from the initial CPT of `Community` and `Category`, and $P_0(R|Com,Cat)$ can be computed from the initial CPTs as follows:

Since `Rating` is a continuous node with a normally distributed mean (see Section 4.1.1), the probability density function for a given rating-value r (computing the y-value in the graph shown in Figure 4.1) can be computed by Equation 4.11 below.

$$f(r,\mu,\sigma) \;=\; \frac{1}{\sigma\sqrt{2\pi}}\; e^{-\frac{(r-\mu)^2}{2\sigma^2}} \tag{4.11}$$

Thus, as an example, for `Community` and `Category` both in state 1, we have a mean value in `Rating` of 1 and a variance value of 1, and the probability P_0 of seeing value 2 in `Rating` is:

$$f(r=2,1,1) \;=\; \frac{1}{1*\sqrt{2\pi}}\; e^{-\frac{(2-1)^2}{2*1^2}} \;=\; 0.2420$$

This is the same normal distribution which was shown in Figure 4.1, and we see from the stapled line of the graph in that figure that the probability density for x=2 was the same as computed here.

We continue with the computations, and using Equation 4.10, the numerator $\hat{P}_1(Com=1,U=1,M=3,R=2)$ is:

$$(0.25*0.25*0.74*0.69*0.2420)+(0.25*0.25*0.74*0.31*0.1942) \;=\; 0.0105$$

(b) The denominator of Equation 4.8, $\hat{P}_1(U=1,M=3,R=2)$, is not dependent of the state of `Community`, as it sums over each state a of $Com=a$. That means that when we do these computations for all states of `Community`, we can replace the denominator with a constant α, and in this case, since `Community` has only two states, we get the following expression:

$$\begin{aligned}&\frac{\sum_{Cat}\hat{P}_1(U=1,M=3,Com=1,Cat,R=2)}{\alpha}\\ +\;&\frac{\sum_{Cat}\hat{P}_1(U=1,M=3,Com=1,Cat,R=2)}{\alpha}\\ =\;&1\end{aligned}$$

The constant α is thus given by:

$$\begin{aligned}&\alpha\\ &= \sum_{Cat} \hat{P}_1(U=1, M=3, Com=1, Cat, R=2)\\ &+ \sum_{Cat} \hat{P}_1(U=1, M=3, Com=2, Cat, R=2)\end{aligned}$$

However, as we are only showing the computations for $Com = 1$ here, we do not have the value for $\sum_{Cat} \hat{P}_1(U=1, M=3, Com=2, Cat, R=2)$, and hence we have to compute the denominator $\hat{P}_1(U=1, M=3, R=2)$ explicitly.

$$\begin{aligned}&\hat{P}_1(U=1, M=3, R=2)\\ &= \sum_{Com,Cat} \hat{P}_1(U=1, M=3, Com, Cat, R=2)\\ &= \sum_{Com,Cat} \hat{P}_1(U=1)\hat{P}_1(M=3)P_0(Com|U=1)P_0(Cat|M=3)P_0(R|Com, Cat)\end{aligned}$$

This expression is computed in the same way as done with the numerator, and the computation is therefore not shown. The result of the computation gives:

$$\hat{P}_1(U=1, M=3, R=2) \;=\; 0.0145$$

Now we can find $\hat{P}_1(Com=1|U=1, M=3, R=2)$ from Equation 4.8:

$$\begin{aligned}&\hat{P}_1(Com=1|U=1, M=3, R=2)\\ &= \frac{\hat{P}_1(U=1, M=3, Com=1, R=2)}{\hat{P}_1(U=1, M=3, R=2)}\\ &= \frac{0.0105}{0.0145} \;=\; 0.7241\end{aligned}$$

2. The second term in Equation 4.6, $\hat{P}_1(Com=1|U=1, M=2, R=3)$, now has to be computed in the same way as the first term. Those computations are not shown here as they are the same as above. The computation yields:

$$\hat{P}_1(Com=1|U=1, M=2, R=3) \;=\; 0.8549$$

Now we are finally ready to compute $\hat{P}_1(Com=1|U=1)$ in Equation 4.6.

$$\begin{aligned}&\hat{P}_1(Com=1|U=1)\\ &= \frac{\hat{P}_1(Com=1|U=1, M=3, R=2) \;+\; \hat{P}_1(Com=1|U=1, M=2, R=3)}{2}\\ &= \frac{0.7241 \;+\; 0.8549}{2} \;=\; \underline{0.7895}\end{aligned}$$

Since `Community` has only two states, $\hat{P}_1(Com=2|U=1)$ is computed by subtracting $\hat{P}_1(Com=1|U=1)$ from 1:

$$\begin{aligned}&\hat{P}_1(Com=2|U=1)\\ &= 1 \;-\; \hat{P}_1(Com=1|U=1)\\ &= 1 \;-\; 0.7895 \;=\; \underline{0.2105}\end{aligned}$$

The same steps have to be performed to compute $\hat{P}_1(Com|U=2)$, $\hat{P}_1(Com|U=3)$, and $\hat{P}_1(Com|U=4)$, but the computations are not shown. The resulting CPT of `Community` after one iteration is shown in Table 4.5.
The accuracy of the value for $\hat{P}_1(Com|U=1)$ computed above suffers from not including enough decimals in the subcomputations, and to make the following computations more accurate, Table 4.5 shows the CPT values resulting from not rounding off in the subcomputations. The value for $\hat{P}_1(Com|U=1)$ is hence not exactly the same as the one computed above (0.7895 vs. 0.7892).

Table 4.5: Resulting CPT for the `Community` node after one iteration

	State of `User`			
	1	2	3	4
1	0.7892	0.2676	0.6474	0.1311
2	0.2108	0.7324	0.3526	0.8689

When comparing the CPT entries of `Community` in Table 4.5 with the initial probabilities listed in Table 4.3, we see that for user 1, the probability of being in community number 1 increases, while for the other users, the probability of being in community number 2 increases. Thus it seams like user 1 is being "isolated" in his own community. If we look at the ratings for each user in the training set in Table 4.1, we see that user 1 has given an average rating of 2.5, while user 2, 3, and 4 have given average ratings of 3.5, 4, and 4 respectively. Hence user 1 is isolated towards another community than the other users because he has a low rating average in comparison with the others, and community 1 thus seems to be for "low-raters", while community 2 seems to be a "high-rating" community.

4.3.5 Category CPT computations

To find the estimates for $\hat{P}_1(Cat|M)$, we have to do the same as with $\hat{P}_1(Com|U)$. As these computations steps are exactly the same as with $P(Com|U)$ above, they are omitted here. The resulting CPT of `Category` after one iteration is shown in Table 4.6.

Table 4.6: Resulting CPT for the `Category` node after one iteration

	State of `Movie`			
	1	2	3	4
1	0.9048	0.0680	0.4311	0.0044
2	0.0952	0.9320	0.5689	0.9956

When comparing the CPT entries of `Category` in Table 4.6 with the initial probabilities given in Table 4.3, we see that movie 1 has a high probability of ending up in category 1, while movie 2 and 4 are almost certainly in category 2, and movie 3 has almost the same probability of ending up in both categories. If we again take a look at the training set in Table 4.1, and focus on the ratings for each movie, we find that movie 1 has an average rating of 2, while movie 2, 3, and 4 have average ratings of 3.67, 3, and 4.5 respectively. Hence it seems like "bad" movies are categorized in category 1, "good" movies are categorized in category 2, and averagely rated movies are placed somewhere in between.

If the number of states in the `Community` and `Category` nodes had been larger, it might have been possible to observe more interesting groups, such as a community for "people who likes sci-fi movies".

4.3.6 Rating mean and variance computations

The remaining part of the first EM iteration is to compute new mean and variance values for the `Rating` node. We start with the mean values.

Computing new mean values
Generally, to find a mean value for a continuous node given a combination of its parents, for each observed value for the continuous variable, we have to weight it against the probability of that combination of its parents given all observed data [10]. Thus, in this case, Equation 4.12 shows how to find the mean value for the `Rating` node, for `Community` in state i and `Category` in state j.

$$\hat{\mu_{ij}} \;=\; \frac{\sum_{k=1}^{n}(r_k * \hat{P}(Com = i, Cat = j|D_k))}{\sum_{k=1}^{n}\hat{P}(Com = i, Cat = j|D_k)} \tag{4.12}$$

In this equation, n is the number of data cases, and $\hat{P}$ is the estimated joint probability for state i in `Community` and state j in `Category` given the observed data case D_k.

Consequently, there are four mean values which need to be computed here, one for each combination of states in `Community` and `Category`. The computation will be shown only for μ_{11}, that is, the mean value given $Com = 1$ and $Cat = 1$.

As we see from Equation 4.12, for each data case k, we have to estimate the joint probability of $Com = 1$ and $Cat = 1$ given the observed data D_k:

1. We start with the data case D_1, where we have observed $U = 1$, $M = 3$, and $R = 2$ (see Table 4.1). Hence we must find an estimate for $\hat{P}_1(Com = 1, Cat = 1|U = 1, M = 3, R = 2)$. We use the chain rule to rewrite the expression, and get:

$$P(Com, Cat|U, M, R) \;=\; P(Com|U, M, R)P(Cat|U, M, Com, R) \tag{4.13}$$

   (a) We start with the first factor, $\hat{P}_1(Com = 1|U = 1, M = 3, R = 2)$, in Equation 4.13.

$$\begin{aligned}
&\hat{P}_1(Com = 1|U = 1, M = 3, R = 2) \\
&= \frac{\hat{P}_1(U = 1, M = 3, Com = 1, R = 2)}{\hat{P}_1(U = 1, M = 3, R = 2)} \\
&= \frac{\sum_{Cat}\hat{P}_1(U = 1, M = 3, Com = 1, Cat, R = 2)}{\sum_{Com,Cat}\hat{P}_1(U = 1, M = 3, Com, Cat, R = 2)} \\
&= \frac{\sum_{Cat}\hat{P}_1(U = 1)\hat{P}_1(M = 3)\hat{P}_1(Com = 1|U = 1)\hat{P}_1(Cat|M = 3)P_0(R = 2|Com = 1, Cat)}{\sum_{Com,Cat}\hat{P}_1(U = 1)\hat{P}_1(M = 3)\hat{P}_1(Com|U = 1)\hat{P}_1(Cat|M = 3)P_0(R = 2|Com, Cat)}
\end{aligned}$$

   $\hat{P}_1(Com|U)$ and $\hat{P}_1(Cat|M)$ are given from the new CPTs in Tables 4.5 and 4.6, and computations for the other subexpressions are shown earlier. The computation yields the result:

$$\hat{P}_1(Com = 1|U = 1, M = 3, R = 2) \;=\; 0.8345$$

(b) We proceed with the second factor of Equation 4.13.

$$\begin{aligned}
&\hat{P}_1(Cat = 1|Com = 1, U = 1, M = 3, R = 2) \\
&= \frac{\hat{P}_1(U = 1, M = 3, Com = 1, Cat = 1, R = 2)}{\hat{P}_1(U = 1, M = 3, Com = 1, R = 2)} \\
&= \frac{\hat{P}_1(U = 1, M = 3, Com = 1, Cat = 1, R = 2)}{\sum_{Cat} \hat{P}_1(U = 1, M = 3, Com = 1, Cat, R = 2)} \\
&= \frac{\hat{P}_1(U = 1)\hat{P}_1(M = 3)\hat{P}_1(Com = 1|U = 1)\hat{P}_1(Cat = 1|M = 3)P_0(R = 2|Com = 1, Cat = 1)}{\sum_{Cat} \hat{P}_1(U = 1)\hat{P}_1(M = 3)\hat{P}_1(Com = 1|U = 1)\hat{P}_1(Cat|M = 3)P_0(R = 2|Com = 1, Cat)}
\end{aligned}$$

whose computation yields the result:

$$\hat{P}_1(Cat = 1|Com = 1, U = 1, M = 3, R = 2) \;=\; 0.4857$$

Equation 4.13 can now be used to compute the joint belief for $Com = 1$ and $Cat = 1$ given the observations in data case D_1:

$$\begin{aligned}
&\hat{P}_1(Com = 1, Cat = 1|U = 1, M = 3, R = 2) \\
&= \hat{P}_1(Com = 1|U = 1, M = 3, R = 2)\hat{P}_1(Cat = 1|U = 1, M = 3, Com = 1, R = 2) \\
&= 0.8345 * 0.4857 \;=\; \underline{0.4053}
\end{aligned}$$

2. The same computations must be performed for each data case k. The results of the computations for all the eight data cases are shown in Table 4.7.

Table 4.7: Joint beliefs for $Com = 1$ and $Cat = 1$ for each data case

Data case	$\hat{P}_1(Com = 1, Cat = 1)\|D_k$
D_1	0.4053
D_2	9.222E-3
D_3	5.401E-7
D_4	0.1968
D_5	6.756E-4
D_6	6.995E-3
D_7	7.777E-6
D_8	1.273E-4

Equation 4.12 is now used to compute the mean μ_{11}.

1. We start with computing the numerator, $\sum_{k=1}^{n}(r_k * \hat{P}(Com = 1, Cat = 1|D_k))$, of Equation 4.12.

$$\begin{aligned}
&\sum_{k=1}^{n}(r_k * \hat{P}_k(Com = 1, Cat = 1|D_k)) \\
&= (2 * 0.4053) \;+\; (3 * 9.222E-3) \;+\; (5 * 5.401E-7) \;+\; (2 * 0.1968) \\
&+ (4 * 6.756E-4) \;+\; (4 * 6.995E-3) \;+ (4 * 7.777E-6) \;+ (4 * 1.273E-4) \\
&= 1.2631
\end{aligned}$$

2. Then the denominator, $\sum_{k=1}^{n} \hat{P}(Com = 1, Cat = 1|D_k)$, of Equation 4.12 is computed as the sum of the joint beliefs in Table 4.7.

$$\begin{aligned}
&\sum_{k=1}^{n} \hat{P}(Com = 1, Cat = 1|D_k) \\
&= 0.4053 \ + \ 0.9.222E-3 \ + \ 5.401E-7 + \ 0.1968 \\
&+ 6.756E-4 \ + \ 6.995E-3 \ + 7.777E-6 \ + 1.273E-4 \\
&= 0.6191
\end{aligned}$$

Now we are finally ready to compute the estimated mean $\hat{\mu_{11}}$, using Equation 4.12.

$$\begin{aligned}
\hat{\mu_{11}} \ &= \ \frac{\sum_{k=1}^{n}(r_k * \hat{P}(Com = 1, Cat = 1|D_k))}{\sum_{k=1}^{n} \hat{P}(Com = 1, Cat = 1|D_k)} \\
&= \frac{1.2631}{0.6191} \ = \ \underline{2.0402}
\end{aligned}$$

The three other estimated means are computed in the same way, and the joint beliefs for each data case for all combinations of states in `Community` and `Category` are given in Table 4.8, and the estimated mean values $\hat{\mu_{12}}$, $\hat{\mu_{21}}$, and $\hat{\mu_{22}}$, as well as $\hat{\mu_{11}}$, are listed in Table 4.9.

Table 4.8: Joint beliefs for `Community` and `Category` for each data case

	$Cat = 1$	$Cat = 1$	$Cat = 2$	$Cat = 2$
Data case	$Com = 1$	$Com = 2$	$Com = 1$	$Com = 2$
D_1	0.4053	0.1576	0.4291	0.0080
D_2	9.222E-3	5.911E-3	0.9155	0.0694
D_3	5.401E-7	9.645E-6	0.0726	0.9274
D_4	0.1968	0.7835	0.0166	3.146E-3
D_5	6.756E-4	1.466E-3	0.6135	0.3843
D_6	6.995E-3	0.0150	0.6013	0.3767
D_7	7.777E-6	1.768E-4	0.1159	0.8839
D_8	1.273E-4	3.360E-3	0.1156	0.8809

Table 4.9: Mean values for the `Rating` node after one iteration

Category	1		2	
Community	1	2	1	2
Mean	2.0402	2.0475	3.3978	4.2365

If we compare the mean values found here to the initial mean values in Table 4.4, we see that the two first means initiated to 1 and 1.5 both increased beyond 2. In the training data, shown in Table 4.1, we see that there are no ratings below 2, and thus these two low-initiated groups will together collect the lowest ratings given. The two groups with mean value 3.3978 and 4.2365 both take a step towards the overall mean rating of 3.5.

Computing new variance values

Generally, to find an estimated variance value for a given mean and combination of

the parents' states, for each observed value for the continuous variable, we weight its squared difference from the given mean against the probability of the combination of its parents given all observed data [10]. Thus, in this case, Equation 4.14 shows how to find the estimated variance value $\hat{\sigma_{ij}}$ for the `Rating` node, when `Community` is in state i and `Category` is in state j.

$$\hat{\sigma_{ij}} = \frac{\sum_{k=1}^{n}((r_k - \hat{\mu_{ij}})^2 * \hat{P}(Com = i, Cat = j|D_k))}{\sum_{k=1}^{n}\hat{P}(Com = i, Cat = j|D_k)} \tag{4.14}$$

Here n is the number of data cases, $\hat{\mu_{ij}}$ is the estimated mean value when $Com = i$ and $Cat = j$, and $\hat{P}$ is the estimated joint probability of state i in `Community` and state j in `Category` given the observations in data case D_k.

Consequently, there are also four different variance values which need to be computed, one for each combination of states in `Community` and `Category`. The computation will be shown only for $\hat{\sigma_{11}}$, that is, the variance value given $Com = 1$ and $Cat = 1$. We have already computed the joint beliefs $\hat{P}(Com = 1, Cat = 1|D_k)$ for each data case k, and thus the computation of the variance is straightforward.

1. We use Equation 4.14, and start with computing the numerator, $\sum_{k=1}^{n}((r_k - \hat{\mu_{11}})^2 * \hat{P}(Com = 1, Cat = 1|D_k))$.

$$\begin{aligned}
&\sum_{k=1}^{n}((r_k - \hat{\mu_{11}})^2 * \hat{P}(Com = 1, Cat = 1|D_k) \\
&= ((2 - 2.0402)^2 * 0.4053) \ + \ ((3 - 2.0402)^2 * 9.222E-3) \ + \ ((5 - 2.0402)^2 * 5.401E-7) \\
&+ ((2 - 2.0402)^2 * 0.1968) \ + \ ((4 - 2.0402)^2 * 6.756E-4) \ + \ ((4 - 2.0402)^2 * 6.995E-3) \\
&+ ((4 - 2.0402)^2 * 7.777E-6) \ + ((4 - 2.0402)^2 * 1.273E-4) \\
&= 0.0395
\end{aligned}$$

2. The denominator in 4.14 is the sum of the joint beliefs in Table 4.7, which was computed above with a result of 0.6191.

Now we are ready to compute the estimated variance $\hat{\sigma_{11}}$:

$$\begin{aligned}
\hat{\sigma_{11}} &= \frac{\sum_{k=1}^{n}((r_k - \hat{\mu_{11}})^2 * \hat{P}(Com = 1, Cat = 1|D_k))}{\sum_{k=1}^{n}\hat{P}(Com = 1, Cat = 1|D_k)} \\
&= \frac{0.0395}{0.6191} \ = \ \underline{0.0638}
\end{aligned}$$

The three other variances are computed in the same way, and the estimated variance values $\hat{\sigma_{12}}$, $\hat{\sigma_{21}}$, and $\hat{\sigma_{22}}$, as well as $\hat{\sigma_{11}}$, are listed in Table 4.10.

Table 4.10: Variance values for the `Rating` node after one iteration

Category	1		2	
Community	1	2	1	2
Variance	0.0638	0.0867	0.5995	0.2388

We see from this table that the third group "covers" ratings from the widest range, while the ratings ending up in the two first groups seem to be very close to the means of those groups. If we again take a look at the training data in Table 4.1, we see that

the two ratings of value 2 are probably placed almost entirely in the two first groups, while the ratings of 3, 4 and 5 are largely covered by the two last groups.

This hypothesis can be verified by looking at the redrawn version in Table 4.11 of the joint beliefs of `Community` and `Category` for each data case. If we look at the numbers written in boldface, we see that the two first groups largely cover data case 1 and 4 where the ratings are 2, the fourth group largely covers data cases 3, 5, 6, 7, and 8, where the ratings are 4 or 5, and the third group covers rating values from the whole scale, although it covers medium ratings more.

Table 4.11: Joint beliefs for `Community` and `Category`, analysis

			$Cat = 1$	$Cat = 1$	$Cat = 2$	$Cat = 2$
U	M	R	$Com = 1$	$Com = 2$	$Com = 1$	$Com = 2$
1	3	2	**0.4053**	**0.1576**	**0.4291**	0.0080
1	2	3	9.222E-3	5.911E-3	**0.9155**	0.0694
2	4	5	5.401E-7	9.645E-6	0.0726	**0.9274**
2	1	2	**0.1968**	**0.7835**	0.0166	3.146E-3
3	2	4	6.756E-4	1.466E-3	**0.6135**	**0.3843**
3	3	4	6.995E-3	0.0150	**0.6013**	**0.3767**
4	4	4	7.777E-6	1.768E-4	**0.1159**	**0.8839**
4	2	4	1.273E-4	3.360E-3	**0.1156**	**0.8809**

4.3.7 Proceeding after the first iteration

One iteration in the EM algorithm is completed, and the algorithm continues with the second iteration. The CPTs of `User` and `Movie` will never change, as these nodes do not have any parent nodes, and there are no missing data for these variables. Hence, we can never give a better estimate of their probabilities than the estimates already computed in the first iteration.

For the other three nodes on the other hand, the next iteration will give new and better estimates, now based on the estimates computed in the first iteration. Thus, the process is exactly the same in the next iterations as in the first iteration, the probabilities in the computations are just replaced to represent the most recently acquired probabilities.

By always utilizing the most recent probabilities, for every iteration the algorithm gets one step closer to the accepted result. The process of recomputing probabilities as shown above continues until the relative change in log likelihood is below some threshold (typically 10^{-3}), or until a predefined number of steps have been performed.

4.4 Model implementation

In the previous section the steps of the EM learning process was shown with a simple network with only four users, four movies, and two states in each of the hidden classes. However, in the MovieLens database which is used in this project, there are 925 users and 1682 movies, and the number of communities and categories are to be varied between one and 50 states. Consequently, the process from the last section gets equally more extensive. For this reason, a tool was used to simplify the computations.

The Hugin Developer is a tool for building, learning, and running Bayesian networks [12]. It has a graphical user interface, but for networks like the ones in this project containing a lot of states or nodes, the use of a programming interface towards Hugin is a better choice. The Java interface of Hugin Developer, version 6.7, was used in this project.

However, in Hugin there is no support for doing EM learning when the network contains continuous nodes. Therefore, this part of the learning process had to be implemented manually. The implementation steps were as follows.

1. **Set up the network structure using Hugin**
   Corresponding to the steps performed in Section 4.3.1, although all the CPT entries in each node were initialized randomly, except from the variance values in `Rating`, which were all initialized to 1.
2. **Use the training set file to do a predefined number of EM iterations with Hugin, learning all non-continuous nodes**
   Even though Hugin is not capable of doing learning of the continuous node, and therefore nothing happens with the CPT of this node, the evidence of this node is still taken into account when the other nodes are learned. This step corresponds to the steps performed in Section 4.3.2 through Section 4.3.5.
3. **Use Hugin to generate the joint community and category belief table**
   That is, to find the values in Table 4.8, instead of computing the beliefs as done in Section 4.3.6, first the evidence for the `User`, `Movie`, and `Rating` nodes for a case in the training set are entered into the Hugin network, and the marginal for all combinations of `Community` and `Category` for the given combination of evidence is read.
4. **Compute the new mean values for `Rating`**
   Corresponding to the step "Computing new mean values" in Section 4.3.6.
5. **Compute the new variance values for `Rating`**
   Corresponding to the step "Computing new variance values" in Section 4.3.6.

4.5 Experimental methodology

All the experiments in this project were performed on the MovieLens data set. Although there is a great ongoing interest in the Netflix Prize contest, and thereby much ongoing use of the Netflix data set, there is more documented research on the MovieLens data set. This section describes how the experiments were carried out. The descriptions include how the data set was split into training and test sets, and choices taken during the testing, as well as two baseline methods used for comparison.

4.5.1 Training and test sets

A test set containing one rating from each of the 925 users was extracted from the 97914 data cases, and this exact same test set was used for all the tests.

Five different training set types were used for each model. None of them contained any of the 925 data cases from the test set, and for each new training set type, additional data cases were picked randomly from the remaining cases. The first training set type contains one rating per user, and is referred to as *1perUser* throughout this report.

Three more training set types were constructed with five, 10, and 19 ratings per user, referred to as *5perUser*, *10perUser*, and *19perUser* respectively. These training sets were constructed by adding cases such that each training set is a superset of the smaller training sets[6]. The last training set type is referred to as *AllBut1*, and contains all of the 96989 cases which remain after removing the 925 test cases.

In addition, ten different training sets each containing 400 randomly selected data cases from the remaining 96989 cases in the initial set were used for some of the experiments in the next chapter. When using these training sets, the same test was run with each of the sets, and the result was computed as the average. This was done to take into account the fact that 400 ratings randomly selected from a set of 96989 cases might not be fully representative for the total data set. The same issue arises with the training sets mentioned above as well, and the test sets. The method presented by Kohavi in [29] can be used to compute the variances of test results, but as his work regards classification accuracy, and we deal with prediction accuracy here (see the next section), his methods are not applicable in our case. It is also the fact that the more cases in a randomly selected training set, the more representative these cases would be for the total data set. The average of the ten 400 training sets are from here referred to as the *400R* training set.

4.5.2 Experience tables

All the conditional probability tables also contain *experience tables*, describing how many times each variable has seen each combination of its parents, or for a variable without parents, how many data cases it has seen. This means that if a network containing two nodes `A` and `B` both with three states where `A` is the parent of `B` is learned with the data cases $A = 1$,$A = 2$, the experience of `A` will be 2, while the experience of `B` for $A = 1$ will be 1, and the experience of `B` for $A = 2$ will be 1.

The experience tells the model how much weight to put on the a priori entries in the CPTs. That is, if the experience tables contain non-zero entries when the learning starts, the initial entries in the CPTs will not be totally overwritten. In this case this is an advantage when using the smallest training sets, as for instance in the *1perUser* set, the maximum number of movies in the training set is 925 movies out of the total 1682 movies. Hence, if we start with an experience entry of zero in the experience table of the `Movie` node, there will be some movies with probability zero after the learning. That is, we say that "it can never happen that this movie appears in the data set". Therefore, to take into account that there will indeed be situations where movies not in the training set will appear in the test set, the experience table of `Movie` is set to 1 initially.

Tests have shown that setting all the initial experience tables to 1 gives better results than starting with zero-entries in some experience tables. This is due to the fact that giving all the weight to the training set will lead to some degree of overfitting, and this overfitting is somehow *smoothed* when giving a little weight to the initial random values. Consequently, for all the experiments listed in this report, all the experience tables were initialized to 1.

[6]That is, all the cases in the *1perUser* set are also in all the other training sets, all the cases in the *5perUser* set are also in the *10perUser* and *19perUser* sets, and so on.

4.5.3 Number of iterations

As indicated in Section 4.3, several EM iterations are normally performed during the learning process. It is possible to repeat the steps until the CPT entries does not change more than a given limit, but the EM learning is very time consuming when dealing with as large data sets as in this project, and thus a change limit will make the learning time more unpredictable, as the number of iterations will change for each run. Therefore, the learning is stopped if it has not reached its limit within a predefined number of iterations.

The change limit was set to 10E-3, and for comparison between the various models, maximum 10 iterations of training the non-continuous nodes were executed, and thereafter one iteration training the continuous `Rating` node. Although some of the models are far from converging after this number of iterations, the comparisons between models will still give approximately the same results, and it is far less time-consuming.

Several tests were done to compare the combination and number of executed iterations, and the general trend is that all results are better when run with more iterations, but that the results for different models are comparable for all kinds of combinations (within certain limits) of iterations. That is, the performance increases similarly for all model types when the number of iterations is increased.

4.5.4 Baseline comparisons

For each of the five main training sets, the results when always predicting the active user's mean rating were recorded, and these results were used as a baseline for the other models. The user mean baseline is from here referred to as the *userMean* baseline.

For the *400R* training set, the users' mean ratings make little sense, as there will be none or only a few users with more than one rating. Therefore, the baseline here was computed as always predicting the overall mean rating of all ratings in the training sets. This baseline method is from here referred to as *overallMean* baseline, and it was also computed for all the other training sets.

The two baseline methods will be listed in the next chapter together with the corresponding results of the model tests.

4.6 Evaluation metrics

In section 2.4, a thorough description of various evaluation metrics utilized in related research was given. With the focus of this project, the accuracy metrics from Section 2.4.1 are the only relevant metrics. We have a movie rating scenario, with the use of the MovieLens database, and the goal is to predict how many stars (out of five) that a user will give each movie, that is, we have a forced prediction case, and thus predictive accuracy metrics are required.

Mean Absolute Error (MAE) is a predictive accuracy metric frequently used in related work, and it is used as the metric here to make comparisons to other work simpler. As explained in Section 2.4, the MAE is computed as the average difference between the predicted rating and the actual user rating for each movie.

Chapter 5

Experiments and Results

In this chapter, the results from the experiments are listed and commented.

All the experiments described in this chapter were executed according to the experimental methodology outlined in Section 4.5, and measured according to the evaluation metrics presented in Section 4.6.

To do the comparison of unnormalized vs. normalized ratings in Section 5.4, experiments are run with both types of ratings. Note that for the smallest training sets, namely *400R* and *1perUser*, normalized ratings are not used, as the average number of ratings per user is one or less, and thus the normalization makes little sense.

Note that all results listed in this chapter are based on the run with the "optimal" number of states in the latent variables for each model. The optimal number of states for the models are given in Section 5.3. MAEs for the experiments with non-optimal number of states are listed in Appendix A.

5.1 BOBW model vs. user- and movie-centric models

As described in the previous chapter, in Section 4.2.1, the novel "best of both worlds" (BOBW) model was to be tested in comparison with the user-centric and movie-centric (item-centric) models of Hofmann [25].

The MAE results from these experiments for unnormalized and normalized ratings, are given in Table 5.1. The MAEs from the baseline methods are also listed in the table.

Table 5.1: MAE for BOBW, user-centric, and movie-centric models

	1perUser	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
overallMean	0.919312	0.921249	0.920879	0.921847	0.927561
userMean	1.075675	0.885837	0.862378	0.844893	0.834019
Plain BOBW, unnorm.	0.932810	0.869040	0.853880	0.834596	0.815412
User-centric, unnorm.	1.125432	1.091004	1.044608	1.015055	0.853764
Movie-centric, unnorm.	1.075675	0.904360	0.893687	0.885494	0.867540
Plain BOBW, norm.	-	0.876835	0.839755	0.806763	0.780215
User-centric, norm.	-	1.037703	0.970470	0.948753	0.828182
Movie-centric, norm.	-	0.917982	0.907078	0.900165	0.874922

From the results in the table, we see that the novel BOBW model has a lower MAE than the user-centric and movie-centric models in all experiments, which confirms the hypothesis that the prediction would improve by introducing latent classes both for grouping users into communities and movies into categories in the same model.

It is also clear that except from the *AllBut1* experiments, the user-centric model is consistently worse than the movie-centric one. This might seem strange, when we remember that Hofmann's item-centric model was consistently worse than his user-centric model [25].
We must however keep in mind that the movie-centric model may have an advantage here, since all training sets but the *AllBut1* contain an equal amount of ratings from each user. The user-centric model on the other hand, might have to deal with movies with none or a only a very few ratings even in the *19perUser* case. In the only training set where the movie-centric model does not have this advantage (the *AllBut1* set), it is outperformed by the user-centric model both with normalized and unnormalized ratings.

5.2 Including additional features

According to the project goals presented in Chapter 1, and as described in Section 4.2, models with built-in demographic user features and movie features were to be tested. Each of the four models shown in Section 4.2.2 were therefore tested in comparison to the plain BOBW model, to investigate the effect of including additional features. The tests were run on all the six training set sizes, to see if including additional features gives more advantages when the total amount of available data is low.

The results for each of the four models are given in this section, in Tables 5.2 through 5.5. Boldface numbers mean that the additional feature model has a lower MAE than the plain BOBW model on the respective training set. All the models are compared to the plain BOBW and the baseline methods.

Table 5.2: MAE for BOBW with year

	400R	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
overallMean	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
userMean	-	1.075675	0.885837	0.862378	0.844893	0.834019
Plain, unnorm.	0.953167	0.932810	0.869040	0.853880	0.834596	0.815412
Year, unnorm.	**0.951738**	**0.922062**	0.898248	**0.852811**	0.848426	0.822256
Plain, norm.	-	-	0.876835	0.839755	0.806763	0.780215
Year, norm.	-	-	0.880694	0.853637	0.823365	0.815273

From the results listed in the tables, we see that the plain BOBW is generally better than the models with additional content features. However, there are several experiments with the smaller training sets in which some of the models with additional features perform better than the plain model. For unnormalized ratings, the year model performs better than the plain model on the *400R*, *1perUser*, and the *10perUser* sets, the age model does better on the *1perUser* set, while the director and actor model does better on the *5perUser* set. The age model also does better than the plain model on the *5perUser* set with normalized ratings. The zip model does not do better than the

Table 5.3: MAE for BOBW with directors and actors

	400R	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
overallMean	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
userMean	-	1.075675	0.885837	0.862378	0.844893	0.834019
Plain, unnorm.	0.953167	0.932810	0.869040	0.853880	0.834596	0.815412
Dir&Act, unnorm.	0.961737	0.944499	**0.868048**	0.867744	0.866141	0.858265
Plain, norm.	-	-	0.876835	0.839755	0.806763	0.780215
Dir&Act, norm.	-	-	0.890810	0.873727	0.854088	0.825682

Table 5.4: MAE for BOBW with zip code

	400R	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
overallMean	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
userMean	-	1.075675	0.885837	0.862378	0.844893	0.834019
Plain, unnorm.	0.953167	0.932810	0.869040	0.853880	0.834596	0.815412
Zip, unnorm.	0.960050	0.935835	0.885922	0.903373	0.886425	0.892053
Plain, norm.	-	-	0.876835	0.839755	0.806763	0.780215
Zip, norm.	-	-	0.885647	0.867712	0.859142	0.818227

Table 5.5: MAE for BOBW with user age

	400R	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
overallMean	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
userMean	-	1.075675	0.885837	0.862378	0.844893	0.834019
Plain, unnorm.	0.953167	0.932810	0.869040	0.853880	0.834596	0.815412
Age, unnorm.	0.954806	**0.924703**	0.876684	0.856282	0.851369	0.855781
Plain, norm.	-	-	0.876835	0.839755	0.806763	0.780215
Age, norm.	-	-	**0.871266**	0.850863	0.828411	0.809665

plain model on any of the training sets, but as up to one third of the users are not from the same country as the rest (see Section 4.2.2), it is a great possibility that this fact means that the zip codes lead to more noise than valuable information.

Thus, it seems like if the right content features are included, it might lead to better accuracy when there is little data, and if the ratings are not normalized. It is an interesting thing to notice though, that when we have a training set with one or less rating per user, namely the *400R* and *1perUser* sets, it is always better to just predict the overall mean rating of the training set. Thus, it is not enough data to infer any latent structures or rating patterns, since the variation for each user is high enough to lead to overfitting when we try to learn anything about a user which has one rating or less.

5.3 Number of states in the latent nodes

For each of the training sets, the models were tested with a number of states in `Community` and `Category` varying from 2 up to 50. Note that with only 1 state in a latent variable, the result of its child will not depend on the latent variable's parent, and thus that variant is not included in the tests here. Since the optimal number of states in the latent variables may differ according to the model type, all the models were tested with varying numbers of states in the latent variables.

The number of states which minimized the MAE is shown for each model on each training set, with unnormalized ratings in Table 5.6, and normalized ratings in Table 5.7 respectively.

Table 5.6: Optimal number of states for minimizing MAE, unnormalized data

	400R	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
User-centric	-	30	25	2	5	2
Movie-centric	-	all	25	20	2	2
Plain BOBW	10	12	2	3	2	12
BOBW with year	2	12	12	5	12	5
BOBW with dir&act	50	5	2	2	10	5
BOBW with zip	10	30	5	5	5	10
BOBW with age	10	10	5	2	5	10

Table 5.7: Optimal number of states for minimizing MAE, normalized data

	5perUser	*10perUser*	*19perUser*	*AllBut1*
User-centric	12	15	15	10
Movie-centric	50	12	12	10
Plain BOBW	25	10	12	10
BOBW with year	30	25	25	5
BOBW with dir&act	2	2	2	2
BOBW with zip	5	5	15	12
BOBW with age	12	5	25	25

As we see from the tables, the tendencies are somehow different when using normalized and unnormalized ratings. When using unnormalized ratings (shown in Table 5.6), it seems like the optimal number of states depended more on the training set than the model type. For instance it looks like the *10perUser* training set "preferred" a low number of states, while the two smallest training sets give the best result with a higher number of states.

An interesting observation in this table is the optimal number of states for the plain BOBW model and BOBW with year with the *400R* training set (10 and 2 states respectively). It is natural that the more information given in the training set, the more states the model will need to represent it. Though, it is worth noting that the difference between the MAEs with varying states is not very big, for instance, for the BOBW model with year, the MAE with 2 states was 0.951738 (as listed in the table), while for 15 states it was 0.954261. Thus, it can be just the fact that the learning might have been prematurely ended which leads to this somehow surprising observation.

With normalized ratings (shown in Table 5.7), on the other hand, it seems like the optimal number of states was more dependent on the model type than the training set, but also the larger training set, the more number of states gave the best result.

Only the results from number of states from 2 to 50 were treated in the tables here. However, tests were also run with only 1 state in each of the latent variables. The results showed that for all the experiments with the *400R* and *1perUser* training sets this gave a lower MAE than the listed experiments. The fact that having only one state in the latent variables means that the predicted rating does not depend neither of the user nor the movie, is important here. If we look at the baseline method *overallMean*, which also do not depend on the user or movie, we can see that this baseline method always did better than all other methods on the two smallest training sets. Thus the observation that one state in each of the latent variables gives the best result with small training sets is quite logical.
Note that with only 1 state in the `Category` node, the movie-centric model turns into a model which always predicts the user average, namely the same as the *userAverage* baseline. This is because the rating prediction given will only depend on the user, and the best prediction the model can give will be the user-specific mean rating.
The last thing worth noting regarding 1 state in the latent variables, is that the user-centric model achieved a very good result here, namely a MAE of 0.778526 (see Appendix A.2 for a listing of MAEs in all the experiments with the user-centric model). With only one state in `Community`, the user-centric model turns into a model which makes its predictions independently of the user, and thus it seems like for this training set it would have been better to always guess the movie-specific mean rating instead of making personalized recommendations.

It is also worth noting here that the log likelihood of the model and the lowest MAE were not necessarily coincident. When run to convergence, the log likelihood increases with the number of states, but the MAE was more often lowest with a smaller number of states.
This observation illustrates the fact that too many states can lead to an overfitting of the model to the training data, which does not necessarily give the best model for prediction on the test data.

5.4 Unnormalized ratings vs. normalized ratings

As explained in Section 2.5.1, the assumption that all users rate on a common scale might not be a correct assumption. A possible way to handle this, is to *normalize* each user's ratings by subtracting the user-specific mean rating from each rating in the training set. When testing the model, the user-specific mean rating is added to each predicted rating for that user.

An interesting issue with normalization is how many ratings per user should be present in the training set before the normalization of ratings gives better performance than the unnormalized version. This is why all the previously described tests were tested both with normalized and unnormalized ratings. A summary of which of the two rating mappings that did best in each combination of model type and training set is given in Table 5.8. The results with the optimal number of states listed in the previous sections of this chapter were used here as well. An `N` entry in the table means that normalized ratings had the smallest MAE, and `U` means that unnormalized ratings gave the smallest MAE.

Table 5.8: Comparison unnormalized vs. normalized ratings

	5perUser	*10perUser*	*19perUser*	*AllBut1*
User-centric	N	N	N	N
Movie-centric	N	U	U	U
Plain BOBW	U	N	N	N
BOBW with year	N	U	N	N
BOBW with dir&act	U	U	N	N
BOBW with zip	U	N	N	N
BOBW with age	N	N	N	N

From the table, we see that the general tendency is that the more data available, the better is the normalized version compared to the unnormalized one. Exactly how many ratings per user there have to be before normalized ratings are better depends on the model type, but for the experiments with the *19perUser* and the *AllBut1* training sets, the normalized version is always best, except from in the movie-centric model, where the tendency is the exact opposite.

The reason why normalized ratings are better with larger data sets should be quite obvious. As we remember from Section 2.5.1, when there is little data available, for instance one or two ratings for each user, it would be somehow dangerous to normalize the ratings, as we will "trick" the model into thinking that those few ratings are representative for the user-specific mean rating. To draw user-specific rating scale conclusions from such few ratings thus causes the quality of the predictions to drop if we have many cases where only a user's ratings of his favourite movies end up in the training set, or only the movies the user likes the least.

5.5 "Optimization"

The combination of model type, number of states in the latent variables, training set, and unnormalized/normalized ratings from the test results listed in the previous sections, was the plain BOBW model with 10 states, the *AllBut1* training set, and normalized ratings.

To perform an experiment as "objective" as possible, five different test sets with 1 rating for each of the 925 users were randomly generated, and for each test set, the remaining 96989 data cases were used in the training set. Thus, this experiment was done with five different training sets, and the results listed here are the average results for each of these training sets.

As described in Section 4.5.3, the results of an experiment are generally better for a greater number of iterations than that of the previously described experiments. Thus, to run the optimized experiment, we use five iterations of non-continuous node training, followed by one iteration of continuous training, repeated several times.

The results after each round of five and one iterations are collected to be sure that the best results are recorded. Only the first six rounds are shown here, since the MAE increased after the sixth round. The average MAE and log likelihood over all the five training sets after each round are shown in Table 5.9. The log likelihood change is

computed as (LL=Log likelihood):

$$\frac{\text{New LL} - \text{Old LL}}{\text{abs(Old LL)}}$$

Table 5.9: Results from the "optimized" run

	MAE	Log likelihood	Likelihood change
After 1 round	0.822826	-1416860	-
After 2 rounds	0.789034	-1410596	4.4E-3
After 3 rounds	0.773489	-1402118	6.0E-3
After 4 rounds	0.765789	-1399440	1.9E-3
After 5 rounds	0.761574	-1395492	2.8E-3
After 6 rounds	0.759697	-1394077	1.0E-3

As we can see from the results listed in the table, the best average MAE was achieved after the sixth round of five and one EM iterations, when the change in log likelihood was at 1.0E-3 (keep in mind that each "round" consists of five iterations with learning of the non-continuous nodes, and one iteration with learning of the continuous node). The average MAE at this point was 0.759697.

Now, to do a little comparison, we can for instance look at the M.Sc project of Doron Harlev [18], which worked with the same data set as here, namely the (small) MovieLens data set. He built a Movie Advisor, and performed testing of several different algorithms, of which most of them did significantly better with reduced coverage. Note that all the models presented in this project have 100% coverage, that is, they can make predictions for all users and all movies, no matter how many common ratings exist in the database. Thus, to compare with Harlev's work, we have to look at the performance of his algorithms at 100% coverage.
With a pure Pearson memory-based algorithm, the MAE was approximately 0.78 when the coverage was 100%. He experimented with a method based on a grouping of movies into genres, which achieved a MAE of approximately 0.82. A combination of the genre method with the Pearson algorithm gave a MAE of approximately 0.81, and a "hybrid genre" algorithm also using the genres to perform a grouping of users achieved about the same, but with a coverage of 98.1%.
In his test set, Harlev used five entries from each of 10% of the users, which gives him a test set containing 470 data cases, while the rest of the data cases were put in the training set. That means he worked with a bigger training set than we have, which should give his methods a head start.
Thus, we can conclude that the BOBW model presented in this project produce better results than all of Harlev's algorithms.

It is also worth noting that Harlev used only one training set/test set division, and therefore the results might have been varying if this division was changed. The training set/test set division of the five used in this experiment which performed best, had a MAE of 0.731828, while the worst-performing division gave a MAE of 0.784289.

Chapter 6

Conclusions and Further Work

6.1 Conclusions

The main objective of this project was to use collaborative filtering algorithms to build models for doing movie rating prediction, based on previous work within the area.

We have given a description of the most important existing algorithms and techniques within the collaborative filtering field. The information gathered in that survey was used to make the necessary choices of a model for recommending movies.

We have proposed a novel latent class model with two latent variables, one for grouping users into communities, and one for grouping movies into categories. The model was based on Thomas Hofmann's latent class models, but seeks to utilize the best from both his user-centric and item-centric models. It has been shown here that the new model has a higher accuracy in all experiments than both of the models of Hofmann, and that the general predictive accuracy of the model is very good compared to other related works.

The inclusion of additional content features into the models has been an important focus in this project. It has been shown that the proposed latent class model is very suitable for including such features, in a simple and intuitive way. It has been proven that content features can improve accuracy in cases where there is little data available. We have also seen that when there is one rating or less available from each user, it is better to utilize a method which makes predictions independently of the user and movie.

It has not been put a lot of effort into the representation of the content information which has been included into the models. The movie release year and user age features have been split subjectively into four groups each, and the inclusion of director and actor could probably have been done in several other ways. When it comes to the zip codes, the fact that approximately 30% of the users are not from the USA, it is obvious that there will be some degree of misinterpretation of zip codes.
With these remarks in mind, the models with additional content features should have a lot more potential, and it is a strength that these models did as well as they did in the experiments here.

During the process, a rather simple method for implementing EM learning when dealing with continuous nodes, has been introduced. As an effect of the process, we have been able to make several interesting observations. One of these is the use of normalized ratings. It has been shown that when the training set has a certain size (19 ratings

per user or more in this project), the accuracy is always better when the ratings are normalized. However, when there is little data available, normalizing the ratings draws false conclusions about rating scales, and thus the performance will be poorer than without normalization of ratings.

We have also been able to make some observations about the optimal number of states in the latent variables. When dealing with unnormalized ratings, the optimal number of states depends more on the training set than on the model type, while with normalized ratings, the model type is the most important factor.

6.2 Further work

This project has covered many aspects within the collaborative filtering area, but there are always possible improvements and additions.

There were several available content features which were not utilized in the models of this project, mainly due to time restrictions. More research should be done on which type of features which can actually be valuable, for instance which demographic user features that actually have a correlation with how the users rate, and therefore can help improve accuracy if included.
As indicated in the conclusions, the representation of the additional content features should have a great improvement potential. One common way to do such discretization is the method of Fayyad and Irani [13].
Another interesting experiment would be to see how combinations of several content features in one model at the same time would affect the accuracy.

In this project, the models have been tested on the MovieLens data set. However, it would be interesting to do experiments with other data sets, including the Netflix Prize contest data set, to be able to compare the predictive accuracy of the models to the recent work of some of the most prominent researchers within collaborative filtering algorithms. The data set used in the Netflix Prize contest contains much of the same type of available information as the MovieLens data, thus the model(s) built in this project would be appropriate to utilize also in the Netflix contest.

The optimization of some parameters could be given more effort, for instance when it comes to the effect of the number of EM iterations. In [23], Thomas Hofmann proposed a method called *tempered EM*, which minimizes a regularized risk function instead of the empirical risk. Hofmann showed that the models trained with tempered EM consistently outperformed the models trained with EM with early stopping.

The only performance focus in this project has been on prediction accuracy. However, if a recommender system is to be used in real-life, there are other considerations to take. Among these is the scalability of the model. As the implementation is at this point, it is not possible to scale it to a thousand times more users, movies or ratings. Further work should choose implementation methods which allow for such scaling.

Finally, since the use of additional content features to improve accuracy is a relatively unexplored area, it would be interesting to do research on inclusion of such features in combination with other collaborative filtering algorithms than latent class models.

Appendix A

MAE for all experiments

A.1 User-centric model

Table A.1: MAE for the user-centric model for varying number of states in the latent variables, unnormalized data

States	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	1.181684	1.128424	1.051004	1.011694	0.816245
2	1.209915	1.115552	1.044608	1.018096	0.853764
5	1.166196	1.093654	1.065369	1.015055	0.887894
10	1.127316	1.106752	1.063911	1.021598	0.905283
12	1.138596	1.100768	1.057332	1.039241	-
15	1.129247	1.100413	1.054653	-	-
20	1.140807	1.099311	1.053298	-	-
25	1.136490	1.091004	-	-	-
30	1.125432	1.094661	-	-	-

Table A.2: MAE for the user-centric model for varying number of states in the latent variables, normalized data

States	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	1.104963	1.024755	0.966165	0.778526
2	1.082301	1.025377	0.992925	0.853518
5	1.048670	1.013199	0.973671	0.854530
10	1.067690	0.993892	0.984911	0.828182
12	1.037703	1.008266	0.973352	0.861287
15	1.055134	0.970470	0.948753	-
20	1.049459	0.995221	0.950785	-
25	1.061225	-	-	-
30	1.049674	-	-	-

A.2 Movie-centric model

Table A.3: MAE for the movie-centric model for varying number of states in the latent variables, unnormalized data

States	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	1.075675	0.885837	0.862378	0.844893	0.834019
2	1.075675	0.925061	0.901140	0.885494	0.867540
5	1.075675	0.943461	0.916461	0.915288	0.885956
10	1.075675	0.936412	0.921976	0.911318	0.893849
12	1.075675	0.906227	0.922680	0.909270	-
15	1.075675	0.922547	0.919275	-	-
20	1.075675	0.905862	0.893687	-	-
25	1.075675	0.904360	0.901132	-	-
30	1.075675	0.906012	-	-	-

Table A.4: MAE for the movie-centric model for varying number of states in the latent variables, normalized data

States	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.885837	0.862378	0.844893	0.834019
2	0.956791	0.917761	0.901744	0.901147
5	0.949186	0.934933	0.936613	0.917703
10	0.952057	0.932778	0.907882	0.874922
12	0.937171	0.907078	0.900165	0.906314
15	0.938429	0.923863	0.901171	-
20	0.936957	0.913094	-	-
25	0.935729	-	-	-
30	0.935719	-	-	-
40	0.924431	-	-	-
50	0.917982	-	-	-

A.3 Plain BOBW

Table A.5: MAE for plain BOBW for varying number of states in the latent variables, unnormalized data

States	*400R*	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
2	0.998809	0.953145	0.869040	0.907244	0.834596	0.819403
5	0.966124	0.940625	0.895159	0.853880	0.894183	0.841091
10	0.953167	0.941683	0.876609	0.865014	0.872972	0.842940
12	0.955989	0.932810	0.881344	0.859761	0.849017	0.815412
15	0.956739	0.938199	0.900646	0.876378	0.850773	0.864176
20	0.954929	0.937107	0.907081	0.882356	0.873484	-
25	0.956129	0.941526	0.911341	0.894834	-	-
30	0.955407	0.944785	0.912044	0.901629	-	-
40	0.955417	0.948449	0.923265	0.921747	-	-
50	0.956793	0.949075	0.929801	0.926539	-	-

Table A.6: MAE for plain BOBW for varying number of states in the latent variables, normalized data

States	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.885837	0.862378	0.844893	0.834019
2	0.927807	0.885531	0.850771	0.789940
5	0.887744	0.903566	0.827175	0.803616
10	0.896072	0.839755	0.825965	0.780215
12	0.882300	0.844910	0.806763	0.806922
15	0.888168	0.843040	0.818513	-
20	0.880273	0.844422	0.823843	-
25	0.876835	0.845253	-	-
30	0.884150	0.846901	-	-
40	0.883906	-	-	-
50	0.882821	-	-	-

A.4 BOBW with year

Table A.7: MAE for BOBW with year for varying number of states in the latent variables, unnormalized data

States	*400R*	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
2	0.951738	0.919655	0.909921	0.918976	0.852213	0.845651
5	0.962552	0.970048	0.924601	0.852811	0.954680	0.822256
10	0.955926	0.927971	0.911270	0.883635	0.867618	0.837343
12	0.956576	0.922062	0.898248	0.891039	0.848426	0.853537
15	0.954261	0.931300	0.899753	0.898396	0.861263	-
20	0.952277	0.933960	0.910006	0.864340	0.878471	-
25	0.954644	0.935706	0.906861	0.894860	-	-
30	0.955447	0.941984	0.900793	0.899195	-	-
40	0.954923	0.943516	0.916214	0.905501	-	-
50	0.954855	0.939525	0.908112	-	-	-

Table A.8: MAE for BOBW with year for varying number of states in the latent variables, normalized data

States	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.885837	0.862378	0.844893	0.834019
2	0.896587	0.864519	0.844698	0.818038
5	0.900567	0.889117	0.864977	0.815273
10	0.881223	0.871154	0.855554	0.824184
12	0.918851	0.872508	0.827777	0.841608
15	0.890042	0.886894	0.828254	-
20	0.896125	0.878022	0.836597	-
25	0.895556	0.853637	0.823365	-
30	0.880694	0.870446	0.847081	-
40	0.883001	-	0.840304	-
50	0.882544	-	-	-

A.5 BOBW with director and actor

Table A.9: MAE for BOBW with director and actor for varying number of states in the latent variables, unnormalized data

States	*400R*	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.927580	0.921733	0.920357	0.921223	0.920747	0.927561
2	0.995238	1.013700	0.868048	0.867744	0.907399	0.927930
5	0.995599	0.944499	0.924426	0.904277	0.939550	0.858265
10	0.975200	0.962452	0.954786	0.936537	0.866141	0.909216
12	0.984565	0.967093	0.948931	0.938911	0.890818	0.905129
15	0.972585	0.958260	0.948661	0.939663	0.895151	-
20	0.970944	0.975094	0.934017	0.936066	0.918254	-
25	0.965084	0.959081	0.947492	0.930854	0.940477	-
30	0.969482	0.955773	0.944934	0.926535	0.930402	-
40	0.961737	0.944832	0.959096	0.935567	0.916327	-
50	0.963777	0.951083	0.959891	0.924715	-	-

Table A.10: MAE for BOBW with director and actor for varying number of states in the latent variables, normalized data

States	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.881729	0.865297	0.845576	0.834019
2	0.890810	0.873727	0.854088	0.825682
5	0.945723	0.942236	0.876422	0.852055
10	0.907323	0.934674	0.939715	0.847276
12	0.934975	0.941923	0.889562	0.880985
15	0.946683	0.920401	0.875288	-
20	0.911275	0.955448	0.906208	-
25	0.921026	0.923454	0.899098	-
30	0.935290	0.940509	0.875772	-
40	0.907621	0.904815	0.952276	-
50	0.916609	0.912120	-	-

A.6 BOBW with zip

Table A.11: MAE for BOBW with zip for varying number of states in the latent variables, unnormalized data

States	*400R*	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
2	0.986706	0.949116	0.998489	0.907276	0.982352	0.935853
5	0.989310	0.983612	0.885922	0.903373	0.886425	0.911790
10	0.983075	0.975435	0.943630	0.947555	0.899772	0.892053
12	0.986046	0.989768	0.911329	0.922577	0.922024	0.911443
15	0.968061	0.967367	0.916438	0.916910	0.906020	0.908248
20	0.966945	0.946087	0.941176	0.914568	0.906352	0.894435
25	0.966034	0.966586	-	0.923028	-	-
30	0.969655	0.935835	-	0.911163	-	-
40	0.964483	0.940692	-	0.930388	-	-
50	0.960050	0.939713	-	-	-	-

Table A.12: MAE for BOBW with zip for varying number of states in the latent variables, normalized data

States	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.885837	0.862378	0.844893	0.834019
2	0.923039	0.873440	0.864312	0.831493
5	0.885647	0.867712	0.865343	0.840249
10	0.971953	0.889950	0.893536	0.864688
12	0.933948	0.929674	0.869976	0.818227
15	0.903044	0.908947	0.859142	0.850470
20	0.915064	0.871039	0.860355	0.855033
25	-	0.897863	0.862700	-
30	-	0.877820	0.863410	-

A.7 BOBW with age

Table A.13: MAE for BOBW with age for varying number of states in the latent variables, unnormalized data

States	*400R*	*1perUser*	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.927580	0.919312	0.921249	0.920879	0.921847	0.927561
2	1.008979	0.942268	0.863215	0.856282	0.974929	0.926179
5	0.966455	0.934627	0.876684	0.863541	0.851369	0.866894
10	0.954806	0.924703	0.954328	0.898860	0.866385	0.855781
12	0.956871	0.943279	0.910159	0.877828	0.874758	0.874023
15	0.957459	0.938243	0.894316	0.892075	0.877150	-
20	0.955715	0.926625	0.895777	0.886121	0.896178	-

Table A.14: MAE for BOBW with age for varying number of states in the latent variables, normalized data

States	*5perUser*	*10perUser*	*19perUser*	*AllBut1*
1	0.885837	0.862378	0.844893	0.834019
2	0.912907	0.910472	0.866832	0.839163
5	0.903170	0.850863	0.875740	0.849793
10	0.885846	0.879918	0.864794	0.821027
12	0.871266	0.852837	0.833375	0.821193
15	0.892974	0.862728	0.842188	0.830313
20	0.893504	0.854272	0.830436	0.813449
25	-	-	0.828411	0.809665
30	-	-	0.839653	-

Bibliography

[1] E-mail contact with Max Harper at GroupLens Research [29.05.2007] `http://www.grouplens.org/contact`.

[2] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as Classification: Using Social and Content-Based Information in Recommendation. Proceedings of the 15th National Conference on Artificial Intelligence, pages 714–720, 1998.

[3] Pavel Berkhin. Survey of Clustering Data Mining Techniques. Technical report, Accrue Software, 2002.

[4] Daniel Billsus and Michael J. Pazzani. Learning Collaborative Information Filters. Proceedings of the 15th International Conference on Machine Learning, pages 46–54, 1998.

[5] John S. Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence, pages 43–52, 1998.

[6] Yung-Hsin Chen and Edward I. George. A Bayesian Model for Collaborative Filtering. Online Proceedings of the 7th International Workshop on Artificial Intelligence, 1999.

[7] Kwok-Wai Cheung, Kwok-Ching Tsui, and Jiming Liu. Extended Latent Class Models for Collaborative Recommendation. IEEE Transactions on Systems, Man, and Cybernetics. Part A: Systems and Humans, 34(1), pages 143–148, 2004.

[8] Michelle Keim Condliff, David D. Lewis, David Madigan, and Christian Posse. Bayesian Mixed-Effects Models for Recommender Systems. Proceedings of the ACM SIGIR Workshop on Recommender Systems: Algorithms and Evaluation, 1999.

[9] Brent J. Dahlen, Joseph Konstan, Jon Herlocker, Nathaniel Good, Al Borchers, and John Riedl. Jump-starting MovieLens: User Benefits of Starting a Collaborative Filtering System with "Dead Data". Technical report, University of Minnesota, 1998.

[10] Sanjoy Dasgupta. Learning Mixtures of Gaussians. Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, pages 634–644, 1999.

[11] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society, Series B, 39(1), pages 1–38, 1977.

[12] Hugin Expert. Hugin Developer [20.05.07]. `http://www.hugin.com/developer/`.

[13] U. M. Fayyad and K. B. Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. Proceedings of the 13th International Joint Conference on Artificial Intelligence, pages 1022–1029, 1993.

[14] G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information Retrieval using a Singular Value Decomposition Model of Latent Semantic Structure. Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 465–480, 1988.

[15] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using Collaborative Filtering to Weave an Information Tapestry. ACM Communications, 35(12), pages 61–70, 1992.

[16] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. A Constant Time Collaborative Filtering Algorithm. Information Retrieval, 4(2), pages 133–151, 2001.

[17] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining Collaborative Filtering with Personal Agents for Better Recommendations. Proceedings of the 16th National Conference on Artificial Intelligence, pages 439–446, 1999.

[18] Doron Harlev. Movie Advisor. M.Sc Project, Tel Aviv University, 2001. `http://www.eng.tau.ac.il/ danar/Miscellaneous/Movie-ad.doc/`.

[19] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An Algorithmic Framework for Performing Collaborative Filtering. Proceedings of ACM 1999 Conference on Research and Development in Information Retrieval, pages 230–237, 1999.

[20] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John Riedl. Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems (TOIS), 22(1), pages 5–53, 2004.

[21] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and Evaluating Choices in a Virtual Community of Use. Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems, pages 194–201, 1995.

[22] Thomas Hofmann. Probabilistic Latent Semantic Indexing. Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 50–57, 1999.

[23] Thomas Hofmann. Unsupervised Learning by Probabilistic Latent Semantic Analysis. Machine Learning, 42(1-2), pages 177–196, 2001.

[24] Thomas Hofmann. Collaborative Filtering via Gaussian Latent Semantic Analysis. Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 259–266, 2003.

[25] Thomas Hofmann. Latent Semantic Models for Collaborative Filtering. ACM Transactions on Information Systems (TOIS), 22(1), pages 89–115, 2004.

[26] Thomas Hofmann and Jan Puzicha. Latent Class Models for Collaborative Filtering. Proceedings of the 16th International Joint Conference in Artificial Intelligence, pages 688–693, 1999.

[27] Rong Jin, Luo Si, ChengXiang Zhai, and Jamie Callan. Collaborative Filtering with Decoupled Models for Preferences and Ratings. Proceedings of the 12th International ACM Conference on Information and Knowledge Management, pages 309–316, 2003.

[28] Jon Kleinberg and Mark Sandler. Using Mixture Models for Collaborative Filtering. Proceedings of the 36th Annual ACM Symposium on Theory of Computing, pages 569–578, 2004.

[29] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. Proceedings of the 14th International Joint Conference on Artificial Intelligence, pages 1137–1143, 1995.

[30] Roderick J. A. Little and Donald B. Rubin. Statistical analysis with missing data. John Wiley & Sons, Inc., 2002.

[31] Benjamin Marlin. Collaborative Filtering: A Machine Learning Perspective. Master's Thesis, University of Toronto, 2004.

[32] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. Content-Boosted Collaborative Filtering. Proceedings of the ACM SIGIR Workshop on Recommender Systems, 2001.

[33] Tom Mitchell. Machine Learning. McGraw Hill, 1997.

[34] Mark O'Connor and Jon Herlocker. Clustering Items for Collaborative Filtering. ACM SIGIR'99 Workshop on Recommender Systems: Algorithms and Evaluation, 1999.

[35] GroupLens Research. MovieLens Data Sets [20.01.07]. `http://www.grouplens.org/node/12/`.

[36] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work, pages 175–186, 1994.

[37] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of Dimensionality Reduction in Recommender System - A Case Study. Proceedings of ACM 2000 WebKDD Web Mining for E-Commerce Workshop, pages 82–90, 2000.

[38] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Item-Based Collaborative Filtering Recommendation Algorithms. Proceedings of the 10th International World Wide Web Conference (WWW10), pages 285–295, 2001.

[39] Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John Riedl. Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System. Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW), pages 345–354, 1998.

[40] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and Metrics for Cold-Start Recommendations. Proceedings of the 25th ACM SIGIR Conference on Research and Development in Information Retrieval, pages 253–260, 2002.

[41] Upendra Shardanand and Pattie Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems, pages 210–217, 1995.

[42] Panagiotis Symeonidis, Alexandros Nanopoulos, Apostolos Papadopoulos, and Yannis Manolopoulos. Scalable Collaborative Filtering based on Latent Semantic Indexing. Proceedings of the IJCAI Workshop on Intelligent Techniques for Web Personalization (ITWP 2006), pages 1–9, 2006.