Designing a Context Dependent Movie Recommender: A Hierarchical Bayesian Approach

Daniel Pomerantz

Master of Science

School of Computer Science

McGill University

Montreal,Quebec

2009-07-15

A thesis submitted to McGill University in partial fulfilment of the requirements of the degree of Masters of Science

ACKNOWLEDGEMENTS

I would like to thank everyone that helped during my graduate studies. I want to thank my supervisor Gregory Dudek for all his support, encouragement, and general kindness. I'd also like to thank all the members of the Mobile Robotics Lab who were very helpful whenever I had a problem, be it math, computers, or otherwise. Finally, I'd like to thank my family and friends for all their support throughout my graduate studies and life.

ABSTRACT

In this thesis, we analyze a context-dependent movie recommendation system using a Hierarchical Bayesian Network. Unlike most other recommender systems which either do not consider context or do so using collaborative filtering, our approach is content-based. This allows users to individually interpret contexts or invent their own contexts and continue to get good recommendations. By using a Hierarchical Bayesian Network, we can provide context recommendations when users have only provided a small amount of information about their preferences per context. At the same time, our model has enough degrees of freedom to handle users with different preferences in different contexts. We show on a real data set that using a Bayesian Network to model contexts reduces the error on cross-validation over models that do not link contexts together or ignore context altogether.

ABRÉGÉ

Dans cette thèse, nous analysons un système de recommandations de films dépendant du contexte en utilisant un réseau Bayésien hiérarchique. Contrairement à la plupart des systèmes de recommendations qui, soit ne considère pas le contexte, soit le considère en utilisant le filtrage collaboratif, notre approche est basée sur le contenu. Ceci permet aux utilisateurs d'interpréter les contextes individuellement ou d'inventer leurs propres contextes tout en obtenant toujours de bonnes recommandations. En utilisant le réseau Bayésien hiérarchique, nous pouvons fournir des recommendations en contexte quand les utilisateurs n'ont fourni que quelques informations par rapport à leurs préférences dans différents contextes. De plus, notre modèle a assez de degrés de liberté pour prendre en charge les utilisateurs avec des préférences différentes dans différents contextes. Nous démontrons sur un ensemble de données réel que l'utilisation d'un réseau Bayésien pour modéliser les contextes réduit l'erreur de validation croisée par rapport aux modèles qui ne lient pas les contextes ensemble ou qui ignore tout simplement le contexte.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Introduction

This thesis considers the algorithms and techniques used to recommend items in a context dependent setting. A *recommender system* is an agent that can give suggestions or recommendations as to which items a particular person or persons should use. Before computers, a "recommender system" was normally a friend. However, the 21st century has seen the widespread adoption of automated recommender systems that use artificial intelligence techniques with machine learning to provide personalized recommendations. Nowadays, it is difficult to browse the Internet without in some way using a recommender system. When Google™outputs search results for a given search query, it can in fact be viewed as a recommendation problem. Given a set of *keywords*, Google recommends websites to visit. Recommendations are important because as illustrated in Figure 1–1, users are often presented with a vast number of choices.

A *Personalized* recommender system is a recommender system that provides customized recommendations for each user. In 1994, Resnick [30] first proposed using a recommendation system to suggest online news articles to users. Since then, personalized recommender systems have become increasingly common with many websites able to provide personalized recommendations for customers. There is a huge potential benefit as a business model: if the customer likes the product that

Figure 1–1: An illustration of the amount of choices a user can be presented with.

has been recommended, then she is more likely to both buy the specific product and continue shopping there in the future. Personalized recommendations have become such an important part of a business model that in 2006, Netflix Inc^{TM}offered a sizable reward to anyone who could design an improved movie recommendation system. Amazon.com uses a recommender system to suggest further products users should buy. It does this by learning what items are normally bought in pairs. There are many other domains for recommenders, including books to read, blogs to read, movies to view, restaurants to eat at, and recipes to cook.

Recommender systems try to predict a *rating* or score for an item. A rating is a numerical value representing how much a user likes an item. There are many factors that determine a rating. Contextual information, such as temporal, emotional, and

physical, is important in determining how much a user likes a product. Contextual information includes, for example, the time of day (temporal), the mood of the user (emotional), and whom the user is with (physical). One problem with many current recommender systems is they fail to take into account contextual information. They do not deal with important questions such as when, where, and with whom you will use the item being recommended. For example, a couple looking to see a movie on a date is recommended movies such as *Finding Nemo* and *Shrek* because they previously watched similar movies with their kids and enjoyed them. Additionally, most recommender systems that do incorporate contextual information do so in a way that prevents users from defining their own contexts because they rely on calculating similarities between contexts. This is problematic when a user creates a new context that is unique to him and the recommender system is not able to find a similar context because of limited information about the new context.

Traditional recommender systems can only answer the question, "What item should I use?" They treat a rating as a function of only the user and the item and ignore an important variable, namely the context variable, that determines the utility. In this thesis, we outline an approach to providing context-dependent recommendations by treating a rating as a function of the user, the item, and the context. We focus on the movie domain, but our algorithmic methods do not require domain specific knowledge. We will use our context-dependent algorithm to answer two questions: "In setting X, what movie should I watch?" and "Given that I gave movie M a score of S in context C, what would I think about it in a different context?" More formally, in the first question our goal is to recommend the items we think the

user would prefer most in a specific context. In the second question, we will seek to predict a rating for a specific item in a specific context, using the rating of the item in a different context. Throughout the rest of this thesis, we will refer to these two problems as "Standard Context Recommendation" (SCR) and "Different Context Evaluation" (DCE).

Adomavicius and Sankaranarayanan [1], [2] explore using context to find similar users and similar items, However, we would like to build a model for every user that is not dependent on other users. This allows each user to have his own definition of a context. For example, while the majority of users may like reading comedies on a vacation, there may be some users that prefer serious novels. Moreover, by designing a single user-based model, we can easily allow users to add their own contexts.

One way to model a user in different contexts is to treat each different user-context pair independently, essentially splitting each user into several users. This is not ideal, however, as it very often happens that *some* of the user's tastes are still the same in different contexts. The recommender system would have to learn the user's tastes from scratch for *each* context forcing users to rate lots of movies in many contexts in order to get a good prediction. However, if we can share the similarities between contexts, we will not require users to rate as many movies.

1.2 Contributions

This thesis provides three main contributions.

1. We provide a summary of previous recommender systems.
2. We propose a model, using a Bayesian Network, to provide content-based recommendations in a context dependent setting.

3. We implement a recommender based on this model and experimentally compare it with several other algorithms.

1.3 Outline

In the rest of this thesis, we will propose a Hierarchical Bayesian model for linking the preferences in each context. Rather than viewing preferences in each context separately, we view each set of preferences as coming from the same unknown multivariate distribution. Then, using Expectation Maximization, we can learn both the original distribution as well as each set of preferences. The Bayesian model is good because it creates probabilistic constraints between contexts. This is useful to avoid over-fitting, but the dimensionality is still quite high. To resolve this, we do two things. First we employ a feature selection technique to select a subset of all features. We also experiment with learning a correlation matrix that can be used to either aggregate contexts together or "borrow" a rating from another similar context.

We use both synthetic data and a real data set gathered from the online movie recommender Recommendz to test our algorithm. While it is possible that context matters in some domains and not others, none of the techniques derived in this thesis are specific to the movie domain. They could be used to recommend books, recipes, news articles, vacations, or any other domain where context is important.

The outline of the rest of the thesis is the following. In Chapter 2, we summarize the previous work performed on recommender systems. This includes work on Collaborative Filtering (CF), Content-Based (CB), Hybrid, and some Context-Dependent recommendations. In Chapter 3, we describe our method for providing context dependent recommendations using a Bayesian Network. We also discuss a

technique using a correlation matrix to aggregate contexts in order to reduce the dimensionality of the data set. In Chapter 4, we show how to apply our algorithm to the domain of movie recommenders. In Chapter 5, we detail the experiments we performed to test our algorithm, both on synthetic data and on a real movie data set gathered. Finally, in Chapter 6, we conclude our work by suggesting some future improvements

CHAPTER 2
Background Information and Related Work

2.1 Overview

To provide context-dependent recommendations, we need to recommend items to a specific user in a specific context. This requires us to predict a rating r of an item i for a given user u in context c. The rating r represents how much the item is liked. In a context independent setting, we start by viewing the usefulness as a function of only the item and user. As the variables are discrete, this is often thought of as a two-dimensional matrix. See Figure 2–1. Each row stores the ratings for a fixed item, and each column stores the ratings for a fixed user. We are given some of the entries in the table and must fill in the rest of the entries. Typically this matrix is very sparse, so estimating unknown entries can be difficult. Once we estimate the unknown values using some form of interpolation, we can select the products that we predict will have the highest ratings. This thesis will focus on ways to improve the predicted rating of an item, since once we calculate this, we can easily make a recommendation by sorting. There are sometimes other factors to consider in making a recommendation, such as the coverage, which is the variety of choices, of the recommendations. but that is beyond the scope of this thesis.

When we add context as an independent variable, we essentially have extended our two dimensional function to three dimensions. We can now view the function as a three (or more) dimensional matrix. The problem is to estimate unknown

	User 1	User 2	User 3	User 4	User 5
Item 1	6	2		5	
Item 2		3	1		2
Item 3	2				
Item 4	10		8	7	
Item 5			1		
Item 6		5		9	10
Item 7			1	8	
Item 8		2			

Figure 2–1: A context independent recommendation matrix. Each user has rated some of the items but not all of them.

entries of a three-dimensional matrix given some of the entries. By adding a third dimension to that matrix, the matrix becomes even more sparse. Before describing how we handle three dimensions, it is necessary to investigate how current techniques estimate functions of two dimensions. In this chapter, we will give an overview of the previous work on recommender systems and give necessary background information relevant to the rest of the thesis.

In a context-independent setting, we estimate the function $r(i, u)$ for an item i and user u. The two most common techniques to predict the rating of an item are *content-based models* and *collaborative-filtering algorithms*. In content-based approaches, we look at previous items already rated by the user and try to predict the usefulness of the item. Collaborative filtering methods are a general class of algorithms that seek to learn patterns in the data. In recommendation systems, collaborative-filtering based techniques determine sets of similar users. Once the system has determined similar neighbors, it can make recommendations based on

the assumption that similar users will like similar items. There are also *hybrid* approaches that combine the above two methods.

In this chapter, we first describe the family of content-based approaches to recommendation. This section is important because the algorithm we propose is content-based, We describe collaborative filtering approaches, which are important to us for aggregating contexts together. We then describe hybrid approaches which use aspects of both methods. Hybrid recommenders look for structure both in the item space and user space. These methods are important because for a context-based recommender, we will also look at two dimensions: "context" and "user." Finally, we outline the previous work on context-dependent recommenders.

2.2 Content Based Recommendation

Content based recommenders predict a rating $r(i, u)$ of an item **i** by looking only at items previously rated by the same user u. In order to do this, the system tries to determine, possibly implicitly, similarities between the items already rated and the item **i**. The utility or rating of an item is determined independently from other users. In a strict content based model, we only look at the *column* of the rating matrix for the specific user. In doing so, we can learn a *user profile*, which is the information needed to represent a user's preferences. Typically the user profile will be stored as a set of numerical values, but it also could include descriptive information such as gender, favorite genre, or location.

To find similarities between items without considering multiple users, we must store various features for each item. Typically, we represent each item **i** as a vector of features. Depending on the application, the vector might be integer, real-valued,

or boolean. There are then several ways to predict a rating for an item-user pair, which are based on learning a profile, which is a set of values representing a user's preferences, for the user. Note that most of these methods can also be applied to the problem of non-personalized recommendations by aggregating over all users.

2.2.1 Linear Model

The simplest model is a linear model, which proposes that every user profile can be modelled as a vector of real numbers that relates to the item vector, which is also a vector of real numbers, in that each element represents how much the user likes the presence of the corresponding feature. Once we learn this weight vector, denoted by $\mathbf{w_u} \in \mathbb{R}^n$, we can make a prediction r_p as to whether a user u will like an item, denoted by $\mathbf{i} \in \mathbb{R}^n$ based on:

$$r_p(u, m) = \mathbf{w_u} \cdot \mathbf{i} \; . \tag{2.1}$$

We can use machine learning algorithms to learn the weights given a set of training data. We do this by calculating a best fit line, which can be done by computing a least squares linear regression [38]. This method minimizes the sum of squared errors between the predicted ratings of the line and the actual ratings under the assumption that all ratings given are exactly the dot product of $\mathbf{w_u}$ and $\mathbf{i}$ plus zero mean Guassian noise. Mathematically, to guarantee optimality we have:

$$r_a(u, m) = \mathbf{w_u} \cdot \mathbf{m} + N(0, \epsilon) \; . \tag{2.2}$$

where ϵ is the amount of noise in the rating. To calculate the least squares regression, we use the following formula:

$$\mathbf{w_u} = \left(X_u^T X_u\right)^{-1} \left(X_u^T Y_u\right) \ . \tag{2.3}$$

where X_u is a matrix in which each row comes from the item vector $\mathbf{i}$ rated by user u and Y_u is a row vector consisting of the numerical ratings given by the user. These weights minimize the squared error of the prediction with the actual data.

Zhang et. al. [38] describe several other possible ways to calculate these weights including Support Vector Machines (SVM) and Naive Bayes models. SVM techniques can be used for both classification or regression. They seek to find an optimal hyperplane to separate labeled data into different classes according to their labels. Bomhardt [6] uses SVM for an online news recommender. When the data is not linearly separable, one can use Kernel techniques [24], [9] to transform the data into a linearly separable set. Zhang et. al. find that least squares regression and Naive Bayes models have higher recall than SVMs as well as other memory-based recommenders. That is, they correctly select more of the top rated documents when tested using cross validation.

2.2.2 Nearest Neighbor Approaches (Item Similarity)

Another content-based approach that has been used is based on a nearest neighbor approach or nearest k-neighbors approach [32]. In these methods, we calculate the similarity of a previously rated item to the item in question and then take a weighted (by the similarity) average of the ratings in the training data.

There are several ways to calculate the similarity. If one knows the feature vectors, then the similarity can be calculated using the cosine similarity measure[31].

$$sim(\mathbf{i_1}, \mathbf{i_2}) = cos(\theta_{\mathbf{i_1},\mathbf{i_2}}) = \frac{\mathbf{i_1} \cdot \mathbf{i_2}}{||\mathbf{i_1}||_2||\mathbf{i_2}||_2} \ . \tag{2.4}$$

Another method is using the inverse Euclidean distance to calculate similarity. In this case, you have that

$$sim(\mathbf{i_1}, \mathbf{i_2}) = \frac{1}{\epsilon + dist(\mathbf{i_1}, \mathbf{i_2})} \ . \tag{2.5}$$

where $dist(\mathbf{i_1}, \mathbf{i_2})$ is the Euclidean distance between the two vectors and ϵ is a small positive number needed to assure numerical stability when the distance is close to zero. We then predict that the user will give a rating that is the same as the average of the nearest neighbors.

2.2.3 Item Based Collaborative Filtering

If the feature vectors are not known then we must estimate the similarity between the items. One way to calulate the similarity is by estimating the feature vectors based on the users that have rated both items. The idea is to see how often the ratings for two items, $\mathbf{i_1}$ and $\mathbf{i_2}$ "agree." Sarwar [32] describes a method to do this as follows. We look at only the users that have rated both items and heuristically compute the similarity between these items. We view each commonly rated item as a "feature." This makes sense since when we do not directly know the feature vectors, we must view each rating as a feature of the item. Using the estimate of the feature vector, we can use the same techniques as before. One possibility, for example, is to compute a cosine similarity between the two vectors. That is, let v_1 and v_2 represent the rating vector for items one and two calculated by taking the list of ratings for

those items over *all* users and removing any ratings that came from users that did not rate both items. In this case we have:

$$sim(\mathbf{i_1}, \mathbf{i_2}) = cos(\theta_{\mathbf{i_1},\mathbf{i_2}}) = \frac{\mathbf{i_1} \cdot \mathbf{i_2}}{||\mathbf{i_1}||_2||\mathbf{i_2}||_2} \tag{2.6}$$

Sarwar compares several similarity measures and finds that adjusting the similarity ratings by subtracting the user's average rating lowers the average mean absolute error (MAE) by approximately ten percent.

Given a similarity measure, we predict a rating r_p for an item i, by selecting the most similar or k-most similar previously rated items to the item i, and calculating a prediction based on a weighted average of the ratings in the training set.

$$r_p(i) = \frac{\sum_{i \in topk} sim(i_i, i) r(i_i)}{\sum_{i \in topk} sim(m_i, m)} \ . \tag{2.7}$$

It is also possible to take the k most similar items and compute an unweighted average. Sarwar finds item based correlation performs better than slightly better than user based correlations. The main benefit is computational efficiency. Since item-item similarities presumably do not change as much over time once an initial set of ratings are given for the items, the similarities between items can be pre-computed and stored. Note that this approach is not strictly "content-based" as we view ratings from other users. However, we leave it in the content-based approach section as the other users are only used to estimate the feature vectors.

2.2.4 Probabilistic Approach

Garden and Dudek[15] describe a probabilistic approach to prediction. Their particular emphasis is to exploit additional attributes of the items being recommended. We can ask users for explicit feedback regarding an arbitrary set of item-dependent features and select the most useful features. Each item vector is a set of probabilities representing the probability each feature occurs in that amount. For example, if the most useful features to a user are "action" and "comedy," the system stores the probability of each feature being present in "low," "medium," and "high" amounts. For each feature f and quantity q, we calculate an expected rating by averaging the ratings given by the specific user of every item in which the user has said feature f occurs with amount q. We define the function $i(f)$ to be the amount of feature f occurring in item i.

$$E_{f,q} = \frac{\sum_{i \in I_{f,q}} r(i)}{n}$$

where $I_{f,q}$ is the set of items in which the user said feature f occurred in amount q. Based on this, we calculate an expected rating of the item dependent on the feature.

$$E_f = \sum_q P(i(f) = q) \times E_{f,q}$$

Finally, we can generate an expected rating by averaging all E_f

$$E(i) = \frac{\sum_i E_f}{|I|}$$

where $|I|$ is the number of features used. By choosing the features based on feedback from the users, Garden lowers the mean absolute error of recommendation by approximately seven percent. However, this kind of recommender is not able to make predictions on as many items because it requires more input.

2.2.5 Problems With Content-Based Approaches

In practice, there are a few common road blocks to making good content-based recommenders. The most common is the "new user problem." Before a user has rated a sufficient number of items, it is difficult to make good predictions, since the algorithm has very little data to train on. A user will not, however, continue rating items without positive feedback and good recommendations. A second problem is overspecialization. This is related to the "new user problem" and occurs when a user has only rated a fraction of the types of items he likes. In this case, only one specific type of item will be recommended. This may lead to the user becoming dissatisfied because all recommendations are for similar items. A user looking for a good adventure book to read may not be pleased that he has been recommended fifty *Goosebumps* books and nothing else. Ziegler et. al.[40] and Bradley and Smyth[7] examine ways to improve recommendation diversity. Normally a recommender system will select the top N predicted movies and recommend them. Another approach that can improve diversity of recommendations is adding the items one at a time to the recommendation list, only allowing them to be added if they are distinct from other recommended items using a distance metric. A final significant hurdle to content-based recommendations is that they involve estimating an item feature vector. In some domains, such as text classification, this is natural, but in other domains such

as a music recommender, where features are not as obvious, this is difficult to do in practice.

2.3 User Based Collaborative Filtering

Collaborative filtering systems make predictions based on ratings from similar users. Goldberg et. al. [16] in their design of Tapestry are often credited with the genesis of computer-based collaborative filtering systems. Intuitively, this is like asking a friend with similar tastes for a recommendation and works on the assumption that similar users will like similar items. In order to make a prediction, a CF system needs to do two things: 1)determine a set of similar users and 2)combine the users predictions. There are two main approaches to finding similar users: 1)memory-based approaches and 2)model-based approaches. We outline these methods here.

2.3.1 Finding Similar Users: Memory-Based Approaches

Memory-based systems use metrics or heuristics to compute the similarity between users. This idea is the same as the item similarity problem described above except we calculate similarity of columns of Figure 2–1 instead of rows. The difference is we compute similarities between *users* instead of *items*. Sarwar et. al.[32] outline one way to do this, which is analogous to the item similarity approach. In item similarity we take the vector of ratings for each item, keeping only those users who rate both items. When calculating user similarity, we take the vector of ratings for each user, keeping only those items rated by both users. There are then several metrics to calculate the similarity between the two users. Two commonly used metrics are cosine similarity and Pearson Correlation. In cosine similarity we have:

$$sim(\mathbf{u_1}, \mathbf{u_2}) = cos(\mathbf{u_1}, \mathbf{u_2}) = \frac{\mathbf{u_1} \cdot \mathbf{u_2}}{||\mathbf{u_1}||_2||\mathbf{u_2}||_2} \tag{2.8}$$

This is the same thing as calculating the angle between the two vectors. The main problem with this is it does not take into account the distribution that the user rates items with. For example, some users may rate items higher than others. Thus for one user, a certain rating may be considered good, but for others, the same rating would be poor. To deal with this, we subtract the mean score of the user. The other problem with using cosine similarity and not subtracting the mean is that often the set of items rated by both u_1 and u_2 is quite small. This causes the angle between the vectors to be very small. As a simple example, if there is only one overlapping item, the angle between the two vectors will always be 0, even if one user gave the item a low score and the other gave it a high score. If we subtract the mean of each user's ratings, we can partially mitigate this problem. This amounts to computing the Pearson Correlation coefficient. Let S_{u_1,u_2} denote the set of items rated by both u_1 and u_2 and let $\overline{r}_{u_i}$ denote the mean rating given by user i, then the Pearson similarity is defined as:

$$sim(\mathbf{u_1}, \mathbf{u_2}) = \frac{\sum_{s \in S} \left(r(u_1, s) - \overline{r}_{u_1}\right)\left(r(u_2, s) - \overline{r}_{u_2}\right)}{\sqrt{\sum_{s \in S} \left(r(u_1, s) - \overline{r}_{u_1}\right)^2 \sum_{s \in S} \left(r(u_2, s) - \overline{r}_{u_2}\right)^2}} \tag{2.9}$$

This also addresses the fact that different users have different scales for rating items. This approach can still, however, be misleading in practice because the angle between two vectors depends on the dimension of the vector space. Breese et. al. [8] suggest using a default value for unrated items. This increases the accuracy of the similarity

measure on user pairs that have relatively few items rated in common. This approach is particularly useful for cases where the mere *selection* of an item to be rated is enough to learn something. For example, in a web site recommender, we might infer information such as how long a user stayed on a website as a rating. In this case, we guess that if the user did not visit a website at all, then he does not like it. By giving all websites a default value, we fill in the missing data with an estimation. This normalizes all data to have the same dimensionality.

Equation 2.9 normalizes based only on the mean. Subtracting the mean for each user does not necessarily normalize all ratings, however. For example, one user may give ratings uniformly throughout all possible numbers whereas another user may only give exceptionally high or low ratings. Resnick et. al. [30] suggest normalizing the data by assuming the ratings follow a Gaussian distribution and normalizing based on variance as well. Jin et. al.[22] go a step further an remove the Gaussian assumption, using a "decoupling" method. Decoupling involves learning a distribution for the ratings instead of assuming a Gaussian. By looking at the distribution of ratings, we can determine what ratings are "high" for the user. For example, if a user rates all items between five and ten, then an item with a rating of five is not preferred, but if he rates all items between one and five, then a five is a preferred item. By storing probability distributions for each rating, there is more freedom than assuming a Gaussian distribution. Jin[21] compares the two approaches and shows that while the Gaussian method performs better than without any normalization, the best approach is to have a more flexible distribution.

The decoupling method assumes ratings are static over time, but this is not always the case. For example the rating a user gives to an item may dependent on the time of day he is using the recommender or the number of ratings he has already given. Bell et. al.[3] describe a way to normalize all ratings to take into account various effects on movies. They calculate a weight for each relationship that allows then to normalize the ratings. By preprocessing the data in this way, they can achieve a 10% reduction in mean squared error. This approach is very open-ended in that it allows for the effect of any relationship to be removed.

2.3.2 Finding Similar Users: Model-Based Approaches

Memory-based approaches is rely on heuristics that have intuitive meaning, but no mathematical guarantees. Model based approaches learn structure from the data based on machine learning approaches and very often involve statistical approaches. Since the goal of collaborative filtering is to determine "similar neighbors," one natural idea is to cluster users into different neighborhoods. Breese [8] describes a Naive Bayes approach to clustering. Given the user's class, each rating by a user is independent of one another. The centers and members of each cluster are then learned by Expectation Maximization. While intuitive, this method, does not, however, lead to improved recommendations.

One of the reasons clustering is difficult is we have to measure the distance of elements in one cluster from another, which involves to a similarity measure or metric similar to that in the previous section. However, the number of items rated by both users is often quite small, making these techniques infeasible. Ungar and Foster [35] propose creating clusters of both items and users. In this way, he seeks

to find "types of users" who like "types of items." By grouping items together, there is more overlap between each user. We now find similarities between users who rate *similar* items similarly instead of when they are the *same* item.

To cluster items and users, we view each user as being assigned to a certain user class with probability U_c and each item as being assigned to a certain item class with probability I_c. For every user/item class pairing, there is a probability $P(u_c, i_c)$ that a user in that class will like an item in that class. At first glance, this also appears to be a straightforward case of Expectation Maximization. With initial guesses for all probabilities, it seems as if we can assign users and items to the most likely classes. Based on these assignments, we could recalculate our initial guesses and repeat these two steps until the algorithm converges. However, Ungar and Foster[35] show that this does not work. The problem is each user has to always belong to the same class. That is, if we have several observations from one user, then every one of those observations is forced to belong to the same user class. Otherwise we are losing the important link of a rating to its owner since we are treating each rating as coming from a different user. To solve this, we have to break our clustering into multiple steps. First, we cluster only users, then only items, and then repeat. By using clusters of clusters, the overlapping data is much less sparse. According to [35], this algorithm on the CDNow website resulted in increased customer purchases.

Huang et. al.[19] show how we can use graph theoretic techniques to group users and address the "new user" problem (also known as "cold start problem"). We set up a bipartite graph with one set of vertices for the users and the other set for items. There is an edge between the vertex for a user and item if and only if the

user has rated the item. Note that they refer to binary cases, but the algorithm can be extended using edge weights to non-binary variables. Typically, a collaborative filtering algorithm will only consider paths of length three. That is, if user 1 likes item A, user 2 likes items A and B, and user 3 likes items B and C, then the collaborative filtering algorithm will find a path from user 1 to item B. However, it will not find a path from user 1 to item C, which may exist since user 1 is similar to user 2 and user 2 is similar to user 3. If we search for paths along the bipartite graph, we can exploit transitive relationships. The more paths from a user to an item, the more likely it is to be preferred. Since longer paths are less likely to be links, it is useful to weight the paths by an exponential decay. Considering paths of length greater than three leads to much better precision and recall In the book domain. This improvement happens both for new users and regular users.

2.3.3 Voting Scheme

Once we have found a set of similar users, we must generate a prediction. Let S_u denote the set of users similar to user u. The most natural approach to combining these recommendations is an average. In that case we have:

$$r(u,i) = \frac{1}{\|S_u\|} \sum_{s \in S_u} r(s,i) \tag{2.10}$$

In cases where we have computed a similarity metric, we can use this to weight the average. We then have:

$$r(u,i) = \frac{\sum_{s \in S_u} sim(u,s) r(s,i)}{\sum_{s \in S_u} sim(u,s)} \tag{2.11}$$

The approaches outlined in Section 2.2.3 can also be used here to normalize the ratings before averaging.

It is also possible to use a probabilistic approach to combine the most similar users. Rather than calculate some sort of average of the most similar users, Breese [8] suggests calculating an expected rating given similar users U_s.

$$r(u,i) = \sum_{j=0}^{n} j \times P(r(u,i) = j | r(U_s, i))$$

These probabilities can be learned using Bayesian networks.

2.4 Problems With Collaborative Filtering

There are two main impediments to making good predictions using collaborative filtering. The first impediment is that often the ratings matrix is too sparse to generate a good relationship between users. The above approaches all seek to address this and have some success in doing so. Clustering items or users reduces the sparsity by aggregating items or users. The graph based approach creates a denser set of links, extending the meaning of "similar" to include transitivity. The "cold start" problem is a subset of the sparsity problem and also occurs in collaborative filtering.

The other main problem with running CF algorithms is scalability. Since many CF methods are memory based, the techniques do not necessarily scale well to larger data sets. CF systems can potentially have millions of users. Performing similarity calculations on these is slow, even with increased computer speed. Item-item pairings can sometimes improve efficiency in practice, but only in cases where item-item similarities can be precomputed. This is difficult to do, however, in domains where items are added frequently, such as recommending newspaper articles.

2.5 Hybrid

Both content-based and collaborative filtering approaches have strengths and weaknesses. A content-based approach is better for modeling a user with a variety of tastes. A collaborative filtering algorithm may struggle to relate users who have some similar tastes, but not identical tastes. For example, two users might have similar tastes in action movies but different tastes in comedies. Without considering the genre or content of the item, picking the best neighbor is not possible. The drawback of a content-based approach is that the system has to have a method for describing an item and determining its features. In the case of text recommendation or a web page recommendation, this can be done by using a "bag of words" approach[5]. However, it is often difficult to estimate item vectors, particularly when dealing with multimedia.

Some authors have proposed trying to get the "best of both worlds" by combining the two approaches and creating a *hybrid* recommender. The simplest hybrid method is to use two separate systems, one CB-based and one CF-based. We can then combine the two methods using, for example, a linear combination of the ratings. Claypool et. al. [10] take a weighted average of a content based algorithm and collaborative filtering algorithm and immediately see improvement in accuracy. The weights come from the certainty of the recommendation. When the item and user pair have many ratings, the CF algorithm is generally more reliable. A key point with this approach is that the CF and content-based predictions are calculated completely separately. This means that if an improvement is made in one type of algorithm, it can be viewed as a "black box" and improve the hybrid recommendation.

It also easily can incorporate other types of predictors. Pazzani [28] uses a voting scheme to combine other information such as demographics.

While combining two separate predictors sometimes works, ultimately this is simply using a heuristic to decide which recommender to use in which case. There are more complex methods that improve recommendations by creating a hybrid model.

2.5.1 Bayesian Net Hybrid Approach

Both content-based and collaborative filtering approaches suffer from the "new user" problem. Zhang and Koren[39] propose a hierarchical Bayesian model to address this by using a hybrid model that relates each user's preference weights to each other. The model assumes that, as in Section 2.2.1, for any given user, there is a linear relationship between the movie vector and the rating. The model relates each user's weights to each other by assuming there is a common population mean μ and covariance matrix Σ^2. See Figure 2–2. Each user's weight vector, $\mathbf{w_u}$ is a Gaussian random vector with mean and covariance matrix μ and Σ^2 respectively. In other words, $\mathbf{w_u}$ is a random sampling from a multi-dimensional normal distribution. Given the weight vector $\mathbf{w_u}$ for a user u and an item vector $\mathbf{i}$, the rating for an is a linear function with Gaussian noise added:

$$r(u, \mathbf{i}) = \mathbf{w_u} \cdot \mathbf{i} + N(0, \sigma_u) \tag{2.12}$$

where σ_u is a noise factor that is calculated for each user. This is the same equation as in Section 2.2.1. We can use Expectation Maximization to estimate the unknown weights w_u, μ, and Σ^2. At each step when we estimate the weights for a user, we combine terms that depend on both the mean over all users as well as the observed

mean for the specific user. We weight these terms based on the sample variance of the mean for all users and the mean for the specific user. As the number of ratings for a given user increases, the w_u term is more important.

This solution is good for dealing with the new user problem because a new user is initially given a set of weights μ (that of the average user) and the model gradually adjusts these weights to fit the user. Since each user's weights are generated from a normal distribution, which has a support over all real numbers, any particular set of weights is allowed. This means that after rating enough movies, the user's weights under this algorithm will converge with the weights from a standard linear regression.

2.5.2 Content-Boosted CF Recommendations

Melville et. al. [25] use content-based recommendations to boost collaborative filtering recommendations. They address the problem of sparsity of the ratings matrix by making separate content-based predictions and incorporating them into the user rating vector. This allows them to entirely fill out the ratings matrix. By doing this, there is no such thing as a "new item." The idea here is very similar to filling out a user vector with default values but can be more accurate because the artificial values are not the same for all users. However, the new inserted ratings are noisy because they are based on predictions and not actually given, so we weight the boosted predictions based on how confident we are in the content-based predictions. These steps lower the MAE by three percent.

2.6 Incorporating Context

In certain domains, the context or setting that an item will be used in is a factor in its utility. For example, when recommending a location for a vacation, factors such

as the time of year, the person with whom the user is traveling, and amount of time off are very important. The movie domain has similar concerns. The tastes of users generally vary depending on when, where, and with whom they are watching the movie.

Adomavicius and Tuzhilin [1] propose what they refer to as a *reduction based approach*. We can view the contextual problem as a multi-dimensional matrix. One way to make a prediction is by only considering the part of the multi-dimensional matrix specific to that context. Often, however, this requires several contexts to be merged using either machine learning techniques or a human expert. Then, when making a prediction, we only consider ratings from that context. For example, if we need to make a prediction for a movie to watch on a Saturday night, and we already calculated that Saturday night ratings are similar to Friday night ratings, we will only look at the ratings given by users on Friday or Saturday night. We would not look at any other ratings. This allows us to reduce the problem to a standard 2D case. In cases where context does not matter this algorithm converges to standard algorithms when context does not matter. Additionally, they use bootstrapping to improve recommendations by testing whether the standard CF algorithm or their context algorithm is more useful in a certain context.

Ono et. al. [26] design a Bayesian network that incorporates context as a variable. They propose viewing an item rating as a random variable that is generated as follows. First, the user, setting, and movie are random variables. These cause certain *impressions* of a movie, which in turn lead to a rating. To make a recommendation, they estimate $P(r|u, s, m)$, the probability of a rating given the user,

situation, and movie. This approach is an improvement upon standard collaborative filtering approaches.

While the above approaches are useful for determining context-dependent ratings, neither one allows users to make their own context because they rely on sharing information in contexts between users. Thus if a user wants to use a context not listed or rated by only a few users, they will not work as well. Additionally, if a user interprets the meaning of a context differently it will affect the predictions in that context for other users. For example, a parent of teenage children would have a very different taste in the context *with my children* than a parent of toddlers. While the Bayesian model of [26] can theoretically deal with this, an extra parameter has to be added to the Bayesian net to model it.

2.7 Dimensionality Reduction and Feature Selection

One common problem with recommender systems is that the dimensionality of the feature space is very large. This often causes the problem to be ill-posed. The dimensionality can often be lowered because many features are redundant and others are useless. For example, there is a large correlation between the features *Keanu Reaves* and *bad acting* because these features redundant. Other features appear in only a few movies, and can be dropped without much information loss. Goldberg et al. [17] suggest using a gauge set of items to recommend jokes using a recommender called Jester. This is an ideal set of items which all users should be asked about and is calculated by computing which items reveal the most information. Similarly, one could create a gauge set of features, choosing to keep only the features most relevant. Some other possibilities are to reduce the dimensionality based on approaches using

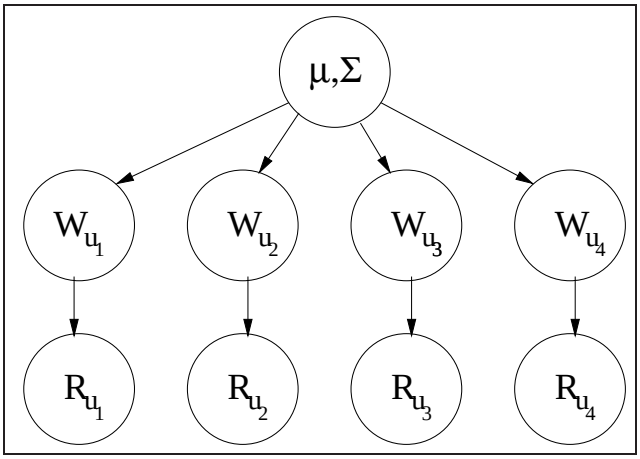


Figure 2–2: Hierarchical model: First a mu and sigma are "chosen" for the entire "population." Then based on this, each users weight is a Gaussian random vector. Finally, given the users preference weights, the rating given to each movie can be determined, but it is not entirely deterministic due to noise. Note that R_i is observed.

information gain, mutual information, Independent Component Analysis (ICA) or Principal Component Analysis (PCA) [27]. For a further analysis of dimensionality reduction approaches, we refer the reader to [36].

2.8 New User Problem

One issue that occurs often in both content-based and collaborative filtering techniques is the "new user problem." One way to address this is by asking users specific questions to learn about them instead of waiting for them to provide the information. Rashid et. al. [29] investigate methods to choose a selection of movies to ask the user about. They compare techniques such as entropy, movie popularity, and random, and find that users are more satisfied with the system that asks questions based on entropy. That is, they give positive feedback by sticking with the system longer.

2.9 Summary

We have outlined several techniques to providing recommendations as well as discussed many of the problems that occur in practice. In Figure 2–3, we have plotted where several of these recommenders lie along two different dimensions. Along the x-axis, the amount of content-based versus collaborative filtering. On the y-axis is the independence of different contexts. For recommender systems that do not consider context, there is zero independence of different context. In the next chapter, we will discuss our contribution, which is creating a context-dependent content-based recommender system using a Hierarchical Bayesian net.

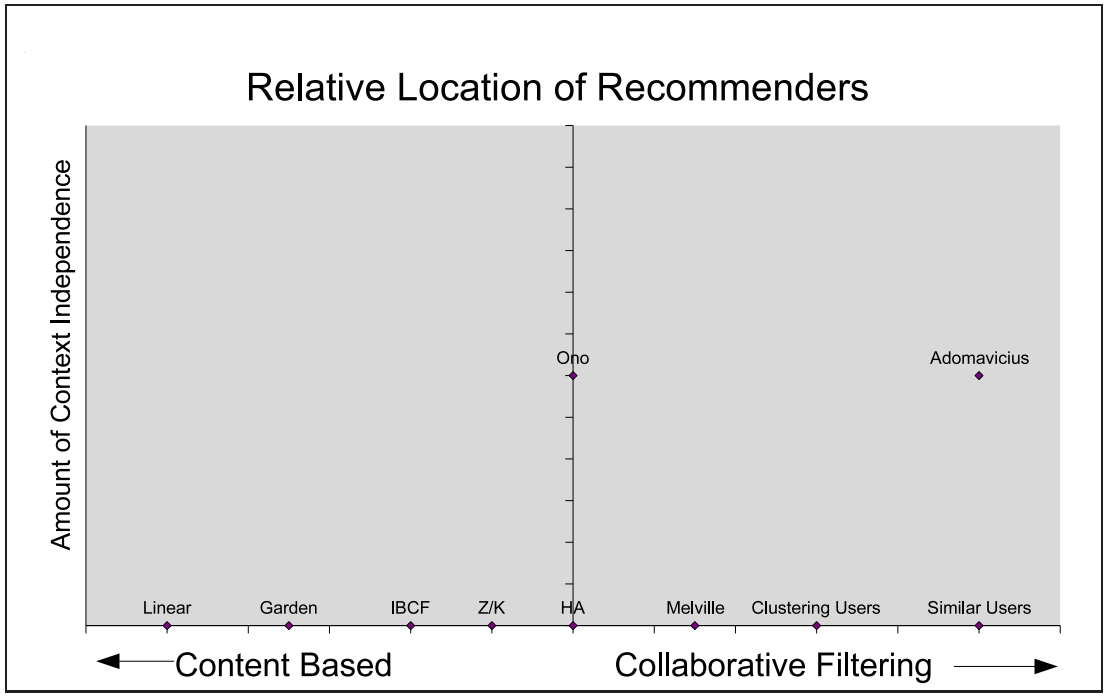


Figure 2–3: A plot showing where several previous recommender systems lie along the "content-based versus collaborative filtering based" dimension as well as the "context dependent or independent" dimension. The further left the algorithm is, the more CB it is. The further right, the more CF. The higher the algorithm is plotted, the more independent the profiles for different contexts are.

CHAPTER 3
Providing Content Based Context Dependent Recommendations

3.1 Overview

One of the problems with many current recommender systems is they do not take into account contextual information such as *when*, *where*, or *with whom* the item is being used. This can cause, for example, a children's book to be recommended to a parent looking for a book to read by himself as a result of past purchases for his child. In this chapter, we describe the approach we use to provide context dependent recommendations. Among the small set of previous context recommenders, most, such as the one proposed by Adomavicius et. al.[1], rely on collaborative filtering techniques which calculate a similarity measure between contexts over *all* users. This requires users to interpret contexts uniformly, but this may not always be the case in practice. Some users may enjoy romantic movies more on dates, but other users may find them to be too cliché. We seek to use a method that allows users to invent their own contexts or interpret provided contexts in individual ways. For example, the context "with family" can have a different meaning for a child than for an adult. For this reason, we will use a content-based approach.

One naive way to model a user's preferences in different contexts is to calculate the preferences of each context separately. The profile for each context can be learned using standard recommendation techniques and only looking at ratings from that context. This would effectively treat each context as a different user. If a user's

tastes are completely different in different contexts then this is the best way to manage context dependent profiles.

The above approach is not feasible because the dimensionality of the solution would be too large. Gathering enough training data to model a user in a context independent setting can be difficult, and we do not want to require users to rate hundreds of items. If we wish to maintain profiles for c contexts, then the amount of data gathering required for such a naive method increases by a factor of c, which is not pragmatic.

Fortunately, we can improve on this if we consider that calculating each context independently ignores the fact that ratings in different contexts do come from the same user. User's preferences in different contexts are often correlated even if they are not exactly the same. If they are correlated, then it is possible to improve our algorithm so that the amount of training data needed stays similar or at least does not increase by as much. Indeed previous context recommender systems in [1] and [26] rely on the assumption that the naive method will not work as they both use methods that exploit dependencies between contexts.

The "new user problem" occurs when a recommender system is unable to make good predictions for a new user because it has not learned enough about the user's preferences. If we do not make profiles in different contexts dependent on each other, then every time a new context is added for a specific user, there would be no information about the preferences in that context. Thus even for users who have inputted many ratings in different contexts, we would have a "new context problem." Given enough data in each context you could treat each context independently, but

we need to exploit dependencies between contexts to avoid creating a "new context problem," which occurs when we do not have enough observed ratings in a given context.

3.2 The Problem of High Dimensionality

Typically a recommender system needs dozens of items in a training set in order to provide recommendations. Swearingen and Sinha have shown that users respond best to a system when it immediately provides recommendations[34]. With that in mind, we require that our algorithm can provide reasonable context recommendations after a training set of limited size per context. If there are c contexts for a user, and the user normally rates n items for a context-independent recommender, then he will only provide on average $\frac{n}{c}$ ratings per context to the recommender. Alternatively, instead of viewing the addition of context information as reducing the training size, one can think of it as increasing the dimensionality of the learning problem by a factor of c. For the remainder of this thesis, we will view the problem as increasing the dimensionality of the solution space as we keep the training set size roughly the same.

If a user's tastes in different contexts are entirely uncorrelated, then the best algorithm would be to treat each context as a separate user, and we will gain nothing from sharing information between contexts. We aim to show, however, that users tastes between contexts are conditionally dependent. For example, if a user hates an actor in one context, without knowing any more information about a second context, we would guess that he also hates the actor. Of course, tastes are not

always positively correlated, but we wish to exploit the positive correlations when they exist.

Working with the assumption that contexts are sometimes correlated, we can deal with the high dimensionality of the solution by creating a "link" or constraint between the profiles of different contexts. By adding these constraints, we reduce the dimensionality of the space. However, if a user's tastes are the same in every context, or if we are in a domain where context does not matter, then the best algorithm will be to ignore context information altogether. Because of these singularities, any model must satisfy the following:

1. Any relationship between contexts must not be fixed. That is, it should allow for almost any possible variations.
2. As the number of ratings in a context goes to infinite, the predictions in each context should converge to what they would under the "separate users" algorithm.
3. We should be able to make a "decent" prediction in a context with only a few ratings.

In the next sections, we describe the techniques we use to satisfy these three constraints.

3.3 Bayesian Learning: Gaussian with Prior

The presence of soft dependencies suggests we might wish to use a probabilistic model. We can assume that for each context c, there is a set of numerical preferences or weights $\mathbf{W_c}$. We model $\mathbf{W_c}$ as a vector of random variables and seek to learn

the probability distribution of the variables. Once we learn the distribution, we can select the most likely values for the weights.

One way to exploit dependencies between contexts is by assuming a prior probability distribution of $\mathbf{W_c}$ based on information from other contexts and then adjusting our distribution as we gather more ratings in the context. We use the Gaussian distribution function as our prior because it is a well studied parametric distribution that occurs often in practice and has support over all real numbers. We let $\mathbf{W_c}$ be a multivariate (multidimensional) Gaussian distribution and make a prior guess as to what the mean and covariance are. We then alter our estimation based on the training data.

More generally, we are trying to estimate a parameter or parameters θ. We have a prior probability distribution $p(\theta)$. Note that in our specific case, $p(\theta)$ is the multidimensional Gaussian function, and θ is thus composed of parameters $\mu \in \mathbb{R}^n$ and $\Sigma^2 \in \mathbb{R}^{n \times n}$. We have several observations D and know $p(D|\theta)$. We are trying to estimate the likelihood, which is $p(\theta|D)$. This will allow us to use the Maximum Likelihood Estimate (MLE). Using Bayes' rule, we have:

$$p(\theta|D) \propto p(D|\theta)p(\theta) \tag{3.1}$$

We can solve for the value of θ that maximizes this probability and estimate it using this solution. If we assume that the observed ratings are a dot product of the item vector $\mathbf{i}$ and weight vector $\mathbf{W_c}$ with the addition of zero-mean Gaussian noise, then we can find a closed form solution for the likelihood function.

Thus we view the context problem in the following way: For a given context c, we need to estimate a set of preference weights $\mathbf{W_c}$. Before gathering any data, we assume that the probability distribution of $\mathbf{W_c}$ is a multivariate Gaussian with mean μ and covariance matrix Σ^2. We observe data D, which are the ratings provided by the user. We assume that the observed ratings are a dot product of the item vector $\mathbf{i}$ and weight vector $\mathbf{W_c}$ with the addition of zero-mean Gaussian noise with standard deviation σ_ϵ. In this case, we can estimate that the likelihood of the parameters is maximized by the following formula. Note that Σ_s^2 refers to the sample covariance matrix.

$$\hat{\mathbf{W}}_\mathbf{c} = \left(\Sigma^{-1} + n\Sigma_s^{-1}\right)^{-1}\left(\Sigma^{-1}\mu_0 + n\sigma_\epsilon^{-1}\bar{x}\right) \tag{3.2}$$

To use these formulas, we must have a prior estimate on μ and Σ^2. A natural idea is to use the sample mean and covariance matrix computed via a context independent linear model. Once we are able to estimate a prior mean μ and covariance matrix Σ^2, we can provide context dependent recommendations.

3.4 Hierarchical Bayesian Model

Unfortunately, we do not know μ and Σ^2 ahead of time. If we make these random variables as well, then we can create a Hierarchical Bayesian net. Hierarchical Bayesian models have been used to link several events which are probabilistically inter-dependent, particularly in cases where the training set is small ([20], [23]). Rather than assume we know μ and Σ^2 ahead of time, as in the Gaussian Prior method, we will learn μ and Σ^2 as well. This method is similar to how Zhang and Koren [39] address the "new user" problem using a Hierarchical Bayesian net. They

use the model to give predictions without requiring a user to rate many movies. Unlike Zhang and Koren, we wish to provide predictions to a user in a specific context. Instead of each branch of the Bayesian network corresponding to a *user*, we design one Bayesian network for each user and let each branch correspond to a specific context. By incorporating an a priori average weight and variance into our model, we avoid ill-posed problems that otherwise would occur frequently in a contextual recommender system because the dimensionality of the solution space is so large. The dimensionality of the solution is the same as when we treat each context independently, but we have added a soft constraint to regularize the system. This regularization assures that we do not have drastically different preferences in different contexts unless we have enough data to support that.

The setup of our generative Bayesian model is described below and shown in Figure 3–1.

1. For each user, a vector $\mu \in \mathbb{R}^n$ and matrix $\Sigma^2 \in \mathbb{R}^{n \times n}$ are generated from an Inverse-Wishart distribution as in [39].
2. Then for every context c, a set of weights $\mathbf{W_c}$ is generated from a normal distribution with mean μ and covariance matrix Σ^2.
3. Finally, for item $\mathbf{i}$, in context c, a rating is generated from a normal distribution with mean $\mathbf{W_c}^T \cdot \mathbf{i}$ and variance σ_ϵ^2.

Theoretically, this is done for every movie in every context. However, we are only able to observe the ratings of *some* of the items. The task for the recommender is to estimate other, unobserved ratings and select the items with the highest ones. In some domains, such as the domain of web articles, the item vector can be observed

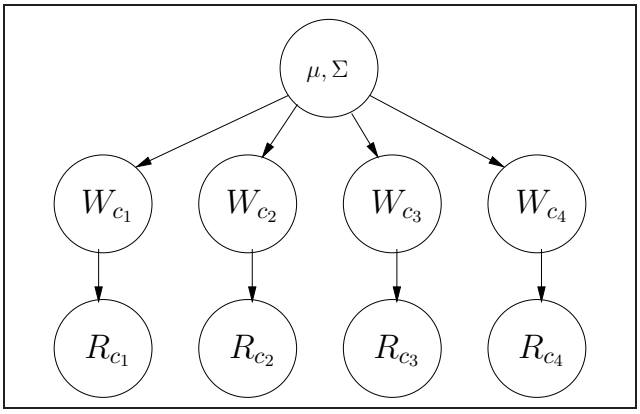


Figure 3–1: Hierarchical model: First a mu and sigma are "chosen" for the user. Then based on this, each context's weight vector is a Gaussian random vector. Finally, given the users preference weights, the rating given to each movie can be determined, but it is not entirely deterministic due to noise. Note that R_i is observed. This is the same as in Figure 2–2 except each branch corresponds to one context instead of one user as the entire tree refers to one user instead of the entire population.

precisely by counting words. In other domains, such as the movie domain, we can not directly observe the item vector $\mathbf{i}$. We are only able to estimate the movie vector based on user input that is subject to variation as well. While conceivably we could use only objective features such as actors, this would be nonetheless be tedious to add to the data set since such a database is not readily available. As well, considering only actors is ignoring much information about the movie's contents.

In the movie domain, an additional complication is that even an "objective" feature can be subjective when we consider that typically the "amount" of the feature is significant. For example, what constitutes a large presence in an item of a feature to one user is not necessarily a large presence to another user. This also varies as a function of the movie and the viewer's expectations. A user watching a Disney movie will have a different threshold for what he considers a lot of violence from a user

watching an action movie. Finally, this approach would not necessarily generalize easily to other domains with subjective features such as a restaurant recommender. Thus we will assume that we are forced to estimate the item vector $\mathbf{i}$.

The Bayesian model has several unknowns: the weight vectors $\mathbf{W_c}$ for each context and the mean μ and covariance matrix Σ^2 that each weight vector is chosen from. We also have to estimate each item vector $\mathbf{i}$, but we view this computation as independent of the Hierarchical Bayesian net.

3.5 Learning the Weights of the Bayesian Net

Yu etl al.[37] describe a procedure using Expectation Maximization (EM) for estimating the weights. $\mathbf{W_c}$, of a Bayesian net given ratings R, which we will summarize here. We need to estimate the weights for each context. If we know the generative μ and Σ^2, then we can estimate the weights $\mathbf{W_c}$ of each branch using a linear regression with a known prior as in Section 3.3. We assume that the prior is a multidimensional Gaussian because Gaussians occur often in practice. Note that this method is compatible with other prior distributions, but the final formulas will of course be different. If we know the weights $\mathbf{W_c}$ of each branch, then we can estimate μ and Σ^2 using the technique of maximum likelihood estimation. These situations are typically solved by expectation maximization. After making an initial guess for μ and Σ^2, we estimate the weights. Then using these new weights, we adjust our estimates of μ and Σ^2. We repeat this until either all the variables stabilize or a fixed number of iterations. We do not present the derivation of the formulas here but merely present the resulting algorithm and refer the reader to [37] and [39] for further on the derivation:

1. Make an initial guess for μ and Σ^2
2. E step: For each context c_i, estimate $P(\mathbf{W_c}|R, \mu, \Sigma^2)$ where R is the set of ratings given by the user.
3. M step: Reestimate μ and Σ^2 using the new user weights.
4. Repeat steps 2 and 3 until all variables stabilize.

In order to estimate $P(\mathbf{W_c}|R, \mu, \Sigma^2)$, for each context c we keep track of the variance of the weights, denoted by Σ_c as well. By keeping track of the variance or certainty of our approximation to each weight vector, we can better estimate the covariance Σ^2 of the entire setup. The formulas for estimating $P(\mathbf{W_c}|R, \mu, \Sigma^2)$ are:

$$\mathbf{W_c} = \left(\left(\Sigma^2\right)^{-1} + \frac{S_{xx,c}}{\sigma_\epsilon^2} \right)^{-1} \left(\frac{S_{xy,c}}{\sigma_\epsilon^2} + (\Sigma^2)^{-1}\mu \right) . \tag{3.3}$$

$$\Sigma_c^2 = \left((\Sigma^2)^{-1} + \frac{S_{xx,c}}{\sigma_\epsilon^2} \right)^{-1} . \tag{3.4}$$

where σ_ϵ^2 is the variance of the noise once the weights are determined (assumed to be known), $S_{xx,c}$ is the sample covariance for the specific user in the specific context. This is computed by taking the matrix composed of all the different feature vectors of items $\mathbf{i}$ that the user rated and multiplying it by its transpose. $S_{xy,c}$ is the matrix created by taking the vector of items $\mathbf{i}$ that have been rated by a specific user in a specific context and multiplying it by the actual ratings given.

In step 3, the mean and covariance matrices are estimated by:

$$\mu = \frac{1}{|C|} \sum_{c_i \in C} \mathbf{W_c} , \qquad \Sigma^2 = \frac{1}{|C|} \Sigma_c^2 + (\mathbf{W_c} - \mu)(\mathbf{W_c} - \mu)^T . \tag{3.5}$$

where C is the set of all contexts for the user. These are the maximum likelihood estimators. Using this algorithm, we can provide context dependent recommendations.

3.6 Correlation Between Contexts

The Hierarchical Bayesian model proposed above assumes that the ratings in one context are *conditionally independent* from ratings in a different context given the mean and covariance for the user. In reality, some contexts are very similar to one another and we can reduce the dimensionality of the problem by merging these similar contexts. This allows us to increase the size of the training set per context by reducing the number of degrees of freedom. This is similar to the what Adomavicius et. al. do in [1], but different from that work since they aggregate contexts over *all* users. Our approach is to merge different contexts for different users, which allows us to merge the contexts *date* and *friday night* for an adult user but not a child user. In this way, we achieve personalized results. To merge contexts, we compute the similarity between any two contexts. For notational simplicity, we will refer to this as a correlation matrix. This is a symmetric matrix where the i^{th} row and j^{th} column is the similarity between the i^{th} and j^{th} contexts.

We calculate the similarity between two contexts for a given user by considering the items that are rated in both contexts, using techniques commonly used in collaborative filtering algorithms. We make vectors v_1 and v_2 for each context out of these item ratings and there are then several ways to calculate the similarity:

- **Mean Absolute Difference**: This is the average difference between the two vectors. That is,

$$sim(c_1, c_2) = \frac{abs(v_1 - v_2) \cdot abs(v_1 - v_2)}{n} \tag{3.6}$$

where n is the number of items rated in both contexts by the same user.

- **Adjusted Pearson Simlarity**: Rather than calculate the similarity based on absolute difference, we can calculate it based on the dot product between the two vectors. We first normalize the ratings by subtracting the mean for the user before calculating the cosine similarity.

$$sim(c_1, c_2) = \frac{(v_1 - \mu_u) \cdot (v_2 - \mu_u)}{\|(v_1 - \mu_u)\|\|(v_2 - \mu_u)\|} \tag{3.7}$$

- **Rank Pearson Similarity**: Here we also compute the Pearson similarity, except we first normalize the ratings by considering the *rank* of each item instead of the actual number given. This may be a more accurate way to compute similarity as it will take into account any asymmetries in the data that occur. These often happen, for example, if a user is more likely to give ratings on extremes (e.g. very good or very bad) or if a user only uses a subset of the rating possibilities[18]. Here the two vectors, r_1 and r_2 are calculated by counting the position of the item with respect to the median rating, allowing negative positions for items below the median. We normalize based on the median so that we have a symmetrical distribution of similarities. Without this adjustment, all similarities would be greater than zero.

$$sim(c_1, c_2) = \frac{(r_1) \cdot (r_2)}{\|(r_1)\|\|(r_2)\|} \tag{3.8}$$

There are also other techniques we can use based on collaborative filtering algorithms. One idea is to give default ratings as in [19], [8], or [25]. We did, however, not test these possibilities for this thesis.

With a metric for measuring the distance between contexts, we can generate the context correlation matrix:

$$\begin{pmatrix} sim(c_1, c_1) & sim(c_1, c_2) & ... & sim(c_1, c_n) \\ sim(c_2, c_1) & sim(c_2, c_2) & ... & sim(c_2, c_n) \\ ... & ... & ... & ... \\ sim(c_n, c_1) & sim(c_n, c_2) & ... & sim(c_n, c_n) \end{pmatrix}$$

Note that it is possible to perform any of the above methods over *all* users as in [1] instead of separately for each user. This can stabilize the values of the similarity since each pair of contexts may have only a few movies in common for a specific user. The downside, however, of relying too much on this approach is we lose the independence of the each user's definition of each context and by relying on this we would make it less feasible for users to create their own contexts.

3.7 Reducing the Number of Contexts

Given a correlation matrix, there are several techniques we can use to reduce the dimensionality of the solution space. The problem is similar to feature selection and dimensionality reduction, but it is important to remember the sparsity of the data. Theoretically, it is possible to do Principal Component Analysis (PCA) to determine which contexts are more relevant. This would essentially be creating "hybrid contexts" and viewing each rating as a mixture of ratings from various contexts. Since a user does not, however, typically rate the exact same list of movies

in all contexts, this would force us to discard any items that were not rated in every context thus potentially leaving us with no items. This differs from using PCA for feature selection as in most applications of PCA all features are observable (such as image processing where a feature is a pixel). PCA is typically used when we have too *much* data as opposed to our case where we have too *little*.

A method based on information gain is not helpful because these techniques involve ignoring data. It is more useful to reduce the dimensionality of the solution space by increasing the number of data points per context than by ignoring data. This is because the estimates for μ and Σ^2 actually improve as we have more contexts to sample from. The problem is the estimates for each $\mathbf{W_c}$ are based on small samples and thus are very susceptible to noise and over-fitting. To make a prediction in context c_i that has only a few observed ratings, one idea is to reduce the dimensionality of the solution by removing all contexts that have small correlation. While this would indeed reduce the dimensionality, it would not avoid over-fitting as our estimate for $\mathbf{W_c}$ would still be based on the same small number of observed ratings. A more appropriate solution is to combine the context having only a limited number of ratings with a context with more ratings. Thus given a correlation matrix, we combine contexts that are highly correlated.

In this work, our approach is to greedily merge the most highly correlated contexts. We continue to merge contexts until no two contexts have a correlation more than a fixed amount.

3.8 Summary

In this section, we described our mathematical model for providing context dependent recommendations. We described a Gaussian prior model and an extension on this using a Hierarchical Bayesian model. We also discussed ways to combine contexts. In the next section, we will discuss the experimental setup and implementation details that we used to test these models.

CHAPTER 4
Experiments and Methodology

4.1 Overview

In this section, we describe the experiments we ran in order to test our algorithms in a real-time application. We start by describing Recommendz, the recommender system that we used to gather data. We then describe many of the algorithmic choices we made that did not involve context. This includes how we estimate the item vector, choose the appropriate features, and estimate the amount of noise per user.

4.2 Real Data Set: Recommendz

We gathered data using the online website Recommendz. This site is accessible at `http://www.recommendz.com` and has been used in previous research [14], [12]. The site has over 3000 users and has been running since 2003. See Figure 4.2, Figure 4–3, and Figure 4–4. The site has the capability of recommending both blogs and movies, but we limit our experiments to the movie domain. Unfortunately, many of the ratings in the Recommendz database are in context-independent settings and are not useful to these experiments. However, we did gather context-dependent data from fifteen users who each rated on average about thirty movies. This is a small number of movies compared with other studies, but is useful for demonstrating the effectiveness of the algorithm on a small training set.

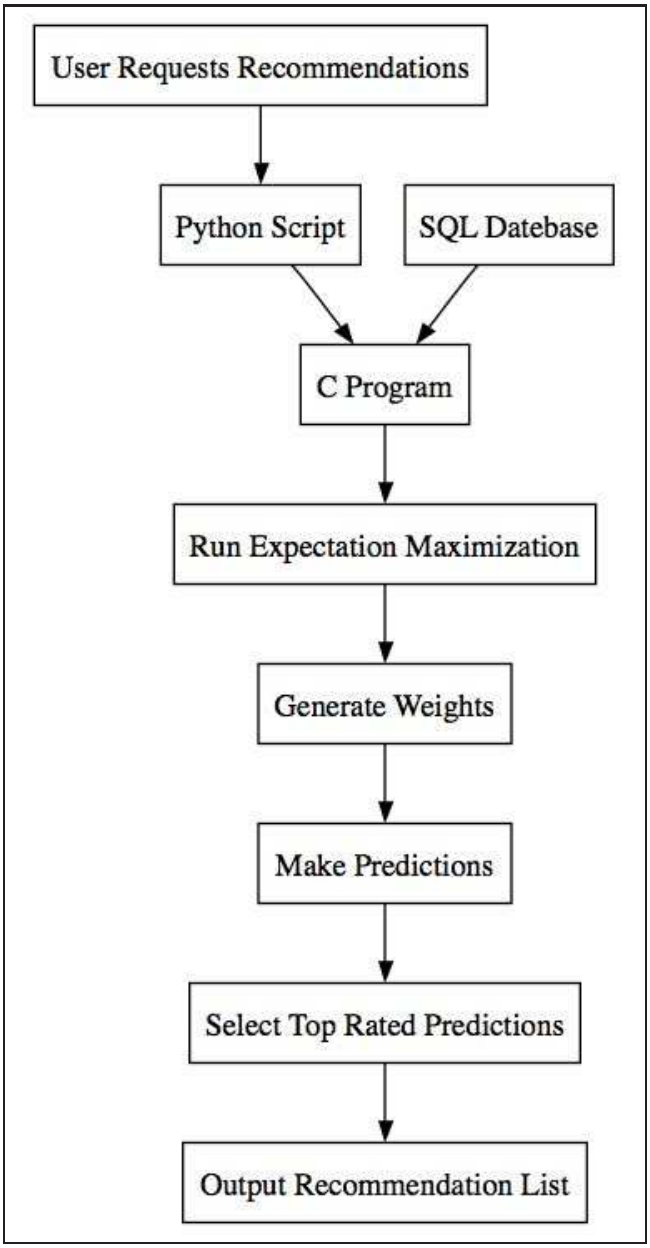
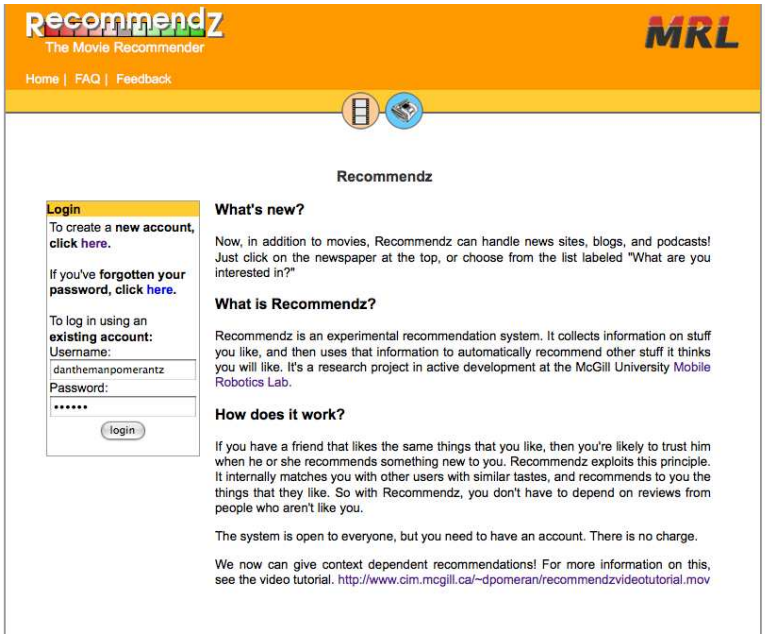
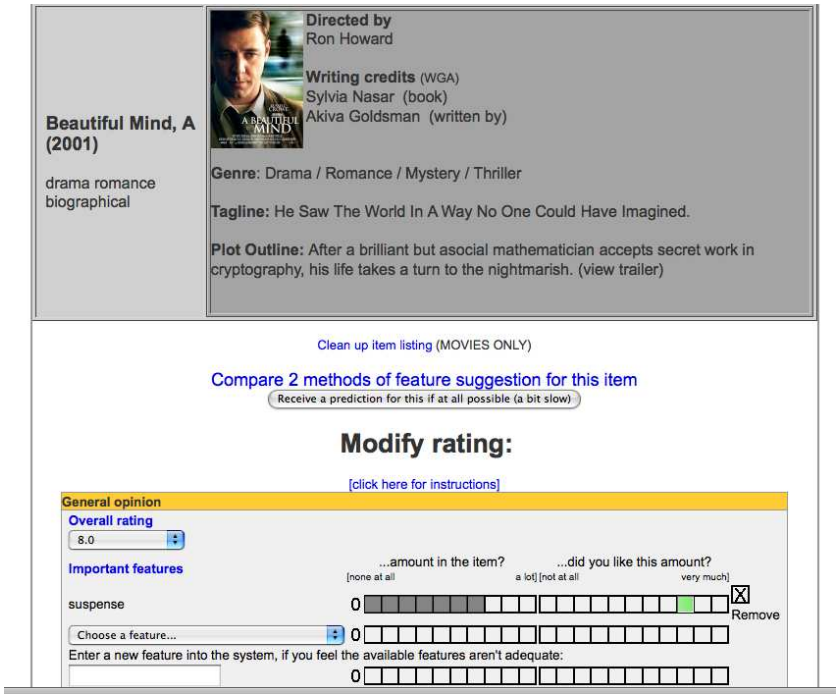


Figure 4–1: A flowchart showing the method used to generate recommendations on Recommendz.



(a) Recommendz Website



(b) Providing a Context Independent Rating

Figure 4–2: Giving a review on the Recommendz website.

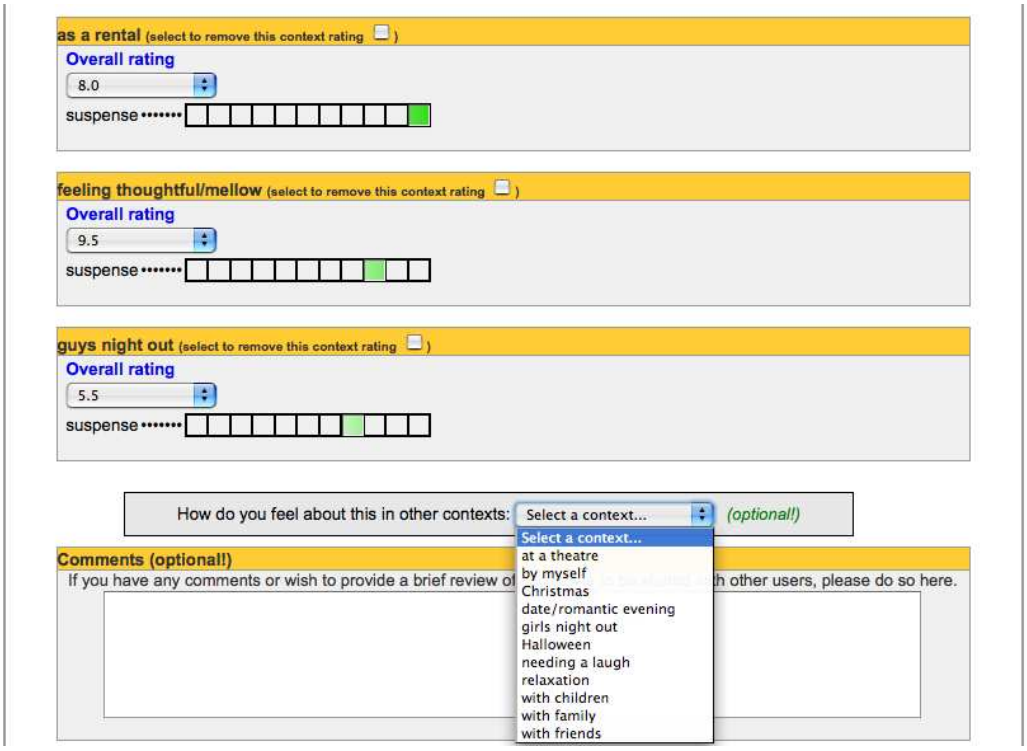


Figure 4–3: Providing a Context Dependent Rating. The user is able to give different ratings for the movie in different contexts

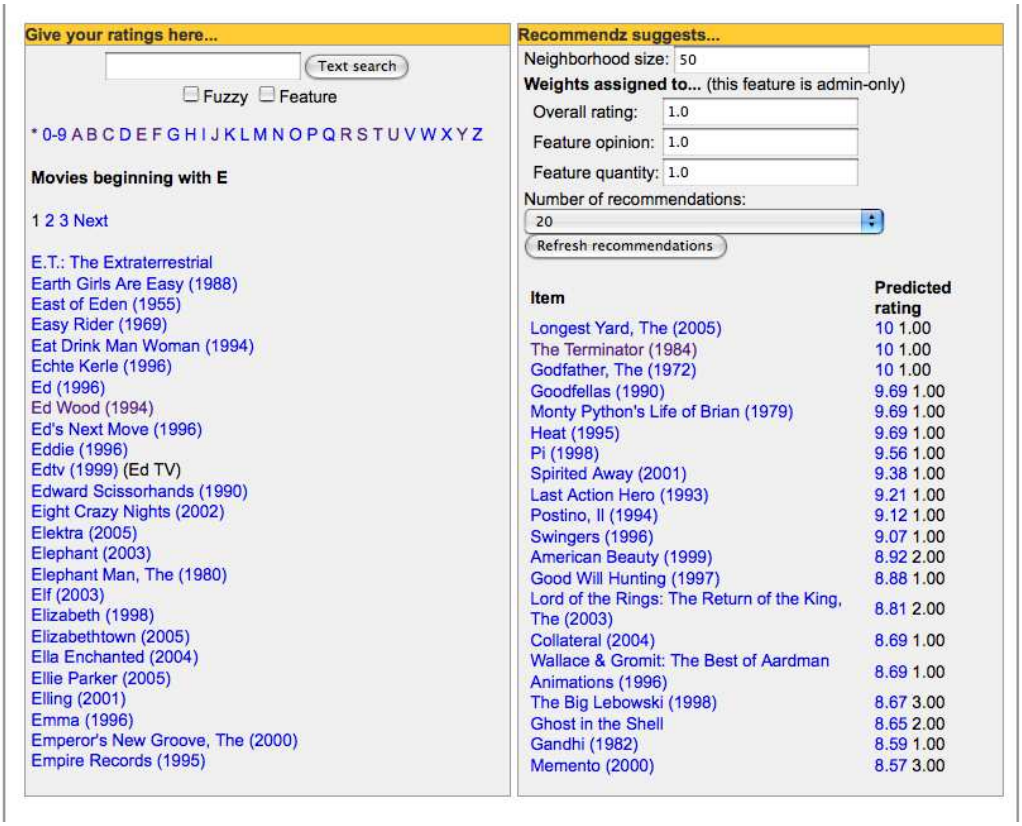


Figure 4–4: Getting a recommendation: On the left, the user can select the movie they wish to rate. On the right, they receive recommendations.

The Recommendz website is run on a Zope server. The front end is implemented in Python and it calls a C-program when it makes a movie prediction. The database is stored using an SQL database. See Figure 4–1. When a user rates a movie on the Recommendz system, he is required to give a numerical preference score on a scale of 1-10 for the movie along with a set of features that he thought was important in forming his assessment of the movie and the amount of each feature present (on a scale of 1-10). These features are then used to estimate the movie feature vector. There are approximately 1500 movie features in the database thus there is a vast amount of data to parse.

4.3 Feature Selection

With over a thousand features, we need a good way to reduce the dimensionality of the input space. The number of parameters in our model is given by FC where F is the number of features in the database and C is the number of contexts needed to estimate. Since there are often hundreds or even thousands of features and several contexts, solving for FC unknowns is impractical. We compared two separate techniques to reduce the dimensionality. In one approach, we only consider the features for which there is an observation for the user. That is, when making a prediction for user u, we consider only the features that user u has explicitly selected as an relevant feature. This works if we assume that a user provides a good summary of the features that are important to them.

In the second method, we run PCA on the entire movie data set to determine the most useful feature dimensions to consider. PCA is a well established and efficient

technique used in machine learning to reduce the dimensionality of the input. Unfortunately, this does not allow us to continue storing a probability distribution for each feature because PCA projects the feature space onto a new feature space composed of "hybrid features," which does not easily extend to a probabilistic method. Another disadvantage of using "hybrid features" is they do not allow us to give any justification for a rating, which could be useful in allowing users to understand *why* we have recommended certain movies. If we do not use hybrid features, we can notify the user which features she has highest preference towards, possibly allowing her to alter these preferences if she disagrees. With hybrid features, this is not possible as the hybrid preferences are not easily interpreted by a human.

4.4 Estimating the Movie Vector

For each feature, rather than storing a Boolean value or a number that represents the amount of the feature in the movie, we store a probability distribution. We do not want to assume that every user perceives movies in an identical way. What appears to one user as a lot of a feature may appear to another user as a small amount. See Section 3.4. The distribution is stored as three numbers which represent the probability of the feature being present in a *low*, *medium*, or *high* amount. Garden [14] experiments with how much detail to store and finds that three is the optimal discretization in the movie domain. While we would like to keep the same number of bins as possible numerical scores so as to not lose information, when more than three bins are used, there are too many parameters to estimate effectively due to the sparsity of the data. When fewer than three bins are used, however, not enough detail is available in the distribution to be of help.

If we have selected n features to use, our final movie vector is then stored as a $3n$ dimensional vector where each dimension represents the probability of a feature occurring in a specific quantity.

Given a sequence of observations $o(f, m, u)$ we first normalize the data to match the user. Typically there is a large asymmetry in the data as many users, for example, put most feature amounts between six and nine. For each user u, we estimate a μ_f and σ_f which are the distributions of numerical amounts of feature presence for the user. We use the sample mean and standard deviation for the estimation. Whenever we record an observation $o(f, m, u)$ that is less than $\mu_f - \sigma_f$, we record this as an example of movie m having a *low* amount of feature f. If the amount is greater than $\mu_f + \sigma_f$, then we record an example of movie m having a *high* amount of feature f. Otherwise if the observation is within σ_f of μ_f, we record the example as being *medium*. Note that each user has a different mean and standard deviation and our movie vector is produced by combining all observations. After doing this, we can calculate the probability distribution of each feature of each movie.

Table 4–1 gives an example of the observations. Each user has a somewhat different distribution of ratings. Jane gives very high ratings, Joe gives very low ratings, Jim gives middle ratings, and John gives diverse ratings. Our algorithm deals with this by considering the distribution of observations per user instead of over all users. Thus "romance" will be recorded as "low" in *Superman* by Jane and feature "action" will be recorded as "high" in *The Matrix* by Joe even though Joe's rating is lower than Jane's. This is because while Jane's rating is a 7, which is well below her average rating, but Joe's rating is a 4, which is well above his average.

Table 4–1: A sample table of feature observation amounts.

User	The Matrix Action	The Matrix Romance	Superman Action	Superman Romance	User Mean
Jane	10	9	8	7	8.5
Joe	4	3	2	1	2.5
Jim	8	6	4	4	5.5
John	10	9	6	2	6.75

After compiling observations over all user, features, and items, we can estimate the item vector.

We illustrate this pictorially in a box plot in Figure 4–5. For each user, a different area is highlighted as central, and we assign all feature amounts in this range to be "medium." Note that the range that we consider "medium" for John is much wider than the range for Jane.

Storing three separate numbers per feature allows us to model more complex user preferences that can not be modeled in a typical linear model. By using three degrees of freedom per feature, we can store more complex patterns such as if the user prefers a feature to be present a lot or not at all, but does not like if there is only an average amount of the feature. Note that the Hierarchical Bayesian algorithm ignores the fact that these three dimensions of representing "low," "medium," and "high" are linked, meaning it does not take advantage of the correlation. While this adds to the dimensionality of the solution space, it allows a more diverse set of users preferences. If there are F features, there each movie will now be stored by $3F$ numerical values.

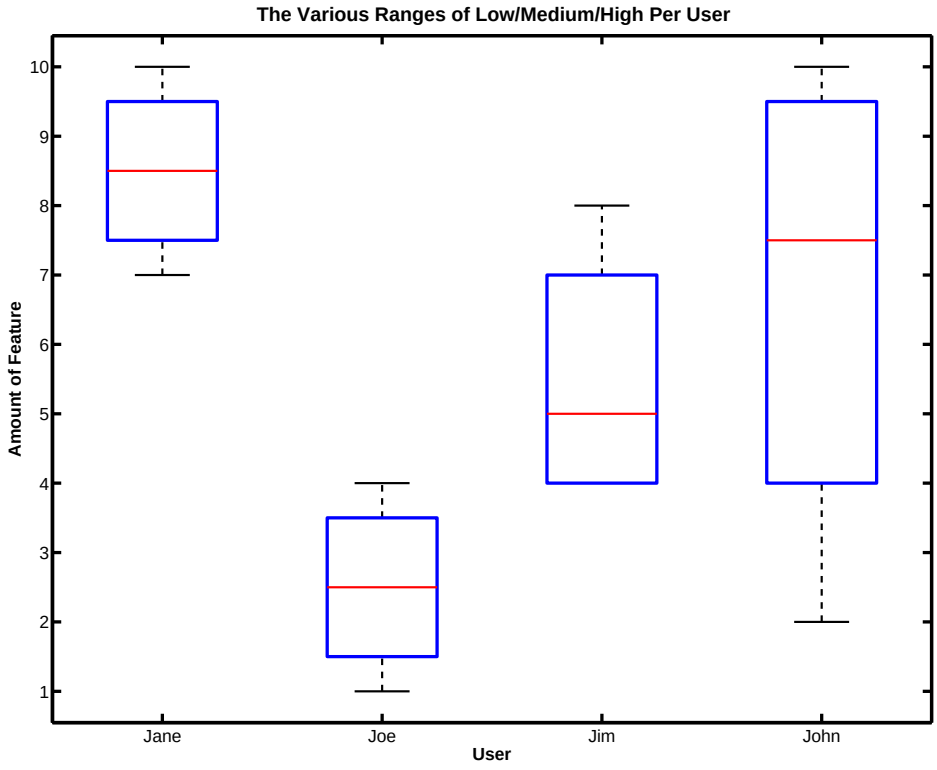


Figure 4–5: An illustration of the different distribution of feature observations. Each box represents the part of the feature presence distribution that counts for "medium" for the given user. The area below the box counts as "low," and the area above counts as "high." The line through the middle of each box represents the average presence value given by the user. Notice that both the mid-points and heights of the boxes are different for each user.

4.5 Estimating the noise per user: σ_ϵ

We need to estimate the amount of noise σ_ϵ created in going from the weights and item vectors to a rating. Zhang and Koren[39] assume that σ_ϵ is given. Yu et. al. [37] propose solving for σ_ϵ during the EM process as well during the M step by measuring the error on the training data. It is even possible to create a different $\sigma_{\epsilon,c}$ for each context. This will only add to the complexity of the problem and thus we assume that while the variation from the linear model is a function of the user, it is not a function of context. Since σ_ϵ represents the amount of noise added to the linear model, we estimate σ_ϵ heuristically by setting it equal to the standard deviation of the error on the training data using *non-context* linear weights, which is the noise in the non-context linear model. This is computed using the least-squares linear regression outlined in [38]. We then leave σ_ϵ constant throughout the EM algorithm.

4.6 Computing an initial μ and Σ^2

The EM algorithm requires that we calculate initial guesses for μ and Σ^2, which we denote by μ_0 and Σ_0^2, respectively. To calculate μ_0, we perform a least square regression in a context-independent setting. Let X represent the movie matrix for the user and Y represent the ratings given.

$$\mu_0 = \left(XX^T\right)^{-1}\left(X^TY\right) \ . \tag{4.1}$$

To calculate the covariance matrix Σ^2, we consider all movies rated in multiple contexts for a user. Based on this we calculate the sample variance and adjust this number by the number of values needed to represent a movie, which is $3|F|$.

$$\Sigma_0^2 = \frac{Var(U)}{3|F|} \ . \tag{4.2}$$

where U is the set of the absolute difference between movies rated in multiple contexts. Since the linear solution has a dimension of $3|F|$, there are $3|F|$ possible terms that can cause variation. For the initial value of Σ_0^2, we assume each dimension is equally likely to cause variation and thus normalize. Note that the initial guesses are merely a heuristic and if the assumption does not hold, the EM algorithm will still converge.

4.7 Summary

In this section, we described our experimental setup using the website `http://www.recommendz.com`. We described how we estimate the item vectors, which involves feature selection and user-specific normalization to estimate a probability distribution function for each feature in each movie. We also described how we calculate the initial values needed to run EM. In the next section, we describe the experiments we ran to test our methods.

CHAPTER 5
Experimental Results

5.1 Overview

Here we describe the experiments we performed related to context-dependent predictions and recommendations. We asked fifteen users to provide on average thirty movies in a context dependent setting. Note that if a user rates the same movie in two different contexts, we count this as two ratings. This means that if a user has provided thirty ratings, he most likely has rated fewer than thirty movies. Since our data set is relatively small, we also tested our algorithms on a synthetic data set.

The experiments were designed to test three things:

- Does context matter at all?
- If so, are different contexts even correlated?
- Is it possible to treat each different context as a separate profile?
- Is the Hierarchical Bayesian Model a useful way to make context recommendations?
- Finally, can we learn anything from computing a correlation matrix?

In the rest of this section, we will describe how we answered these questions.

5.2 Do Some Preferences Change in Different Contexts?

The first question we answered is whether or not context matters at all. In order to answer this, we analyzed the variation of ratings given in different contexts. As there may be an inherent variation in ratings from day to day, we also compared

these results with a control group we asked to re-rate the same movies, in context independent settings, on different days. On average, the re-rating was performed three years after the original ratings were given. This is a much larger gap than between giving context ratings. The goal of this was to determine whether the variation of ratings in different contexts was because users preferred different movies in different contexts or because our data gathering technique caused variation when users rated the same movie several times in different contexts. We found that on average, the ratings in context dependent settings had a variation of 1.50. When users re-rated movies, there was an average difference of .679. This difference is statistically significant when using a t-test at the $\alpha < 0.01$ level. This shows that users do consistently rate movies similarly from day to day. Thus we conclude that the variation in the context ratings is due to users having a variety of tastes in different contexts and not a temporal instability in ratings. This shows that it is necessary to consider context in making a prediction.

5.3 Are Preferences in Different Contexts Conditionally Dependent on Each Other?

Having shown that user tastes do change in different contexts, we next test whether there is any relationship at all between item preferences in different contexts. This tests whether the best solution is to treat each context independently, essentially as a separate user. We performed a simple data analysis on our context ratings. For every movie that was rated in multiple contexts by the same user, we looked at how many pairs of context ratings fell into the same half of the data when ranked in order by rating. For example, if a user whose average rating is 5.0 rates a movie in three different contexts, giving it scores of 1,2, and 10, we count it as one pair the

same (1 with 2) and 2 pairs different (1 with 10 and 2 with 10). Note that if the rating is exactly the median we do not count it towards either group. If there is no correlation between contexts, then there should be a fifty percent split of the two. Our results show agreement 66.74% of the time. A standard t-test reveals a t-score of 7.22 which says there is almost zero chance of this happening if the true proportion is 0.5. Based on this, we can conclude that there is some correlation between a user's ratings in one context from another. If a movie is rated x in one context, then if we guessed that it is also rated x in another context, we have a better than random chance of agreement. Thus while it is important to consider context, we can improve recommendations by considering information from other contexts.

5.4 Predicting Ratings

Knowing that it is important to both consider context and share information between contexts, we seek to test the Hierarchical Bayesian (HB) algorithm in a context-dependent setting. To do this, we compared it with several baseline content-based algorithms. They are the following:

- Least Squares Linear Regression Ignoring Context (LSRIC). In this baseline algorithm we ignore the information we have regarding context and treat all contexts the same.
- Least Squares Linear Regression Separate Users (LSLSU). In this algorithm we deal with context by breaking each user into several users.
- Item Based Collaborative Filtering (IBCF). In this algorithm, we select the N closest neighbors that the same user has rated and average.

- Gaussian Prior (GP). This is a linear algorithm with Gaussian prior, the same thing as aborting the EM algorithm after one step.

5.4.1 Synthetic Results: Hierarchical Bayesian Algorithm

In order to obtain a larger amount of suitable data within a controlled environment, we first ran our algorithms on a synthetic data set. To generate a synthetic item $\mathbf{i}$, we created a vector in n dimensional space by uniformly selecting a subset of the features to be high. We generated users according to the assumptions made in the Hierarchical Bayesian algorithm. That is, we first selected a mean $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma^2 \in \mathbb{R}^{n \times n}$ from an Inverse-Wishart distribution. We then generated weight vectors $\mathbf{W_c} \in \mathbb{R}^n$ for each context c_i as samples from a multi-variate normal distribution with parameters μ and Σ^2. We then simulated a context-dependent ratings by taking the dot product of the weight vector $\mathbf{W_c}$ and the item vector $\mathbf{i}$ and adding zero-mean Gaussian noise.

In Table 5–1 we have listed the mean and median error. The best algorithm is GP and the second strongest is HB. Both algorithms perform better than any of the baseline algorithms. The fact that GP performs better than HB suggests that perhaps using EM is not necessary. Rather it is sufficient to take the initial guess. This makes sense as if can accurately predict the initial mean, then no improvement will come from EM and in fact we may lose accuracy. On a synthetic data set, we can correctly estimate the mean. These results show, however, that HB and GP are both significantly better than the baseline algorithm and are thus good for making context dependent predictions.

Table 5–1: The results of the recommender when using cross validation on a synthetic data set. Based on 348 synthetic ratings. The HBA algorithm improves over Linear with p-value .0387 for the mean and .0249 for median.

Algorithm (SCR)	**Mean Absolute Error**	**Median Absolute Error**
LSRIC	2.3065	1.4822
LSLSU	4.7025	5.4182
IBCF	2.6282	2.0703
GP	**1.9061**	**1.1083**
HB	2.0344	1.2281

5.4.2 Real Data: Answering for a New Movie

For each algorithm, we computed 5 numbers. The mean error overall, the median error overall, mean error per user, median error per user, and F-Score in selecting the top quarter. The difference between mean error overall and mean error per user is due to the fact that some users have rated more movies than others. In per user error, we count each user equally. This normalizes the data so that each user contributes equally to the scoring metric, a normalization which may or may not be desired. While MAE is an intuitive error measure, F-Score, which depends on both the precision and recall of the algorithm, is considered by some to be a more accurate measure of error in recommender systems because in many applications the most important criteria for a recommender system is that it recommends the top movies [18]. In evaluating the F-Score, we divided our rankings into percentiles. Unfortunately, this forces a discretization of the data since the set of items are not actually divided into discrete percentiles. For example, it is preferable to put a movie that is supposed to be in the top quartile into the second quartile than the

Table 5–2: The results of the recommender when using cross validation to answer the first question. Bold items mean optimal relative performance. Based on 262 ratings. Results significant with p-value of .0283.

Algorithm (SCR)	**MAE**	**Median**	**MAE PU**	**Median PU**	**F-Score**
LSRIC	2.2449	2.1061	2.134	2.166	.2893
LSLSU	2.8094	2.2500	2.802	2.281	.2989
IBCF	2.2807	2.0855	2.150	2.051	.0755
GP	2.1733	1.827	2.038	1.746	.2763
HB	**2.1381**	**1.6257**	**2.0103**	**1.6533**	**.4079**

bottom quartile. However, F-Score counts both as an error. The numerical results are presented in Table 5–2 and illustrated in a bar graph in Figure 5–1.

The mean absolute error (MAE) is fairly similar in all cases. However, for the problem of recommendations, MAE is not considered the best form of measurement as the results are often skewed by the uneven distribution. For example, whether a movie is rated 4.0 or 1.0 is largely irrelevant for a task of recommendation. In any event, the strongest performance is using the Bayesian model. In Figure 5.4.2, a histogram of both the Bayesian algorithm and the linear algorithm's error is given. The error in the Bayesian algorithm is very right skewed compared to the linear algorithm.

It is somewhat difficult to measure statistical significance. The reason for this is we do not know the underlying distribution. However, if we assume a normal distribution, then we can do a paired t-test. For median error, the Hierarchical Bayesian algorithm performs significantly better than others. The fact that HB has a much smaller median than mean indicates that there are some predictions that are far off, but the majority of ratings are closer. The ratings adjusted per user show a

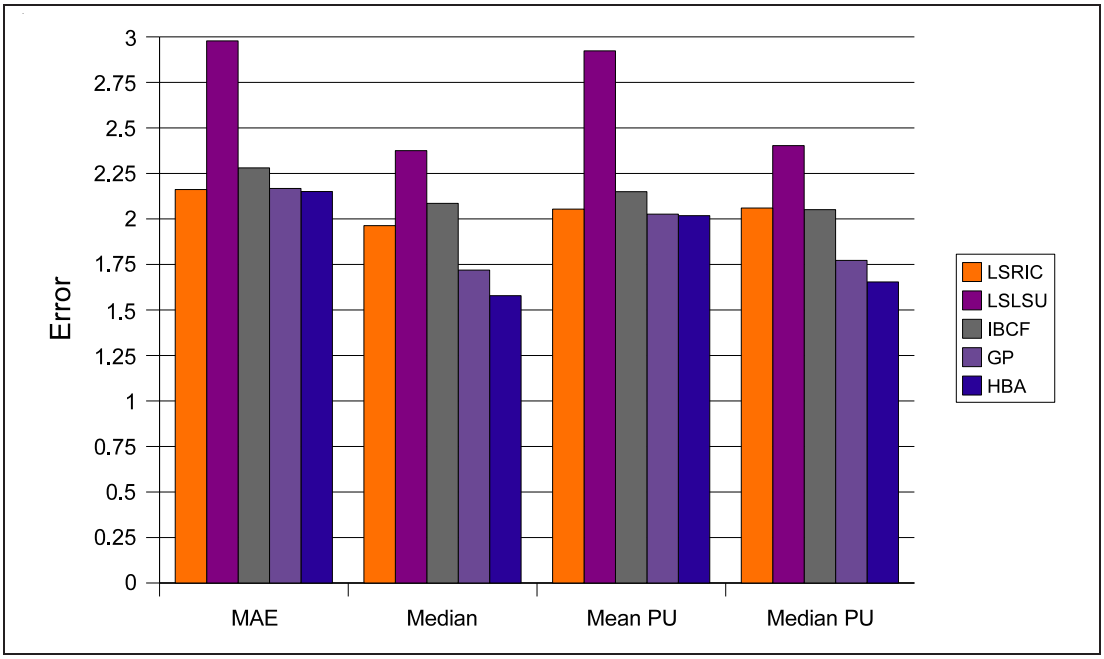
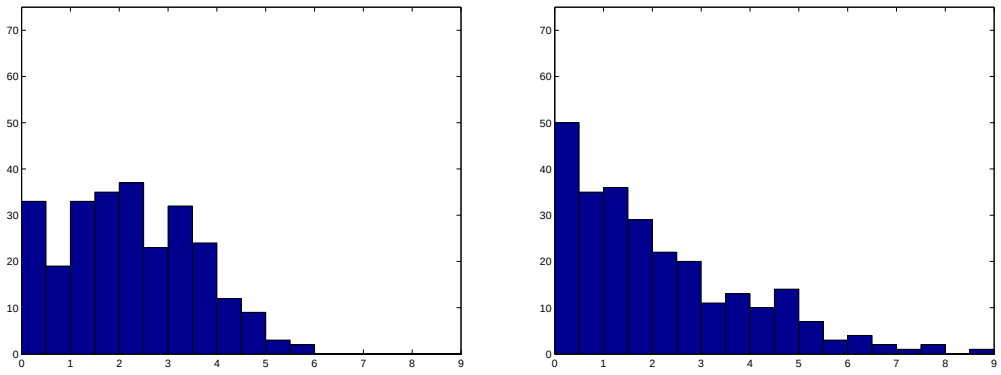


Figure 5–1: A comparison of the errors of the various algorithms for a movie never before rated by the same user. The Bayesian algorithm performs best in all cases. The difference in the median error (both per user and separately) between baseline linear and per user algorithms with the Hierarchical Bayesian algorithm is statistically significant.



(a) Linear Histogram Error (b) Bayesian Histogram Error

Figure 5–2: Two histograms showing the error of the linear algorithm (left) and the Bayesian algorithm (right) in answering for a new item. The Bayesian algorithm has significantly more small errors, but a few large errors. This explains why the median is smaller in the Bayesian algorithm despite a smaller mean.

comparable change, with HB performing best in both, but doing much better in the median.

As stated before, the F-Score breaks the data set into separate, discrete categories, when in reality there is no such thing. However, as it is considered a good estimate of a recommenders ability to select the top movies, we list it here. The Hierarchical Bayesian algorithm performs better than the baseline linear algorithm. The worst approach is the item collaborative one.

The first linear algorithm treats all ratings as coming from the same context. Another way of thinking about this is that it entirely ignores the context information given by the user. This linear model suggests that the utility is a function of just the item and user and not context. The separate user algorithm does almost the opposite as it uses the context information *too* much. The fact that the Hierarchical

Bayesian algorithm performs better than each of these in all forms of measurement shows that it is useful to consider context, but at the same time useful to connect contexts to each other. On the real data set, HB is an improvement over GP, showing that when we can not correctly estimate the initial mean, we can improve the guess using EM.

5.4.3 Real Data: Answering for a New Context

Our second goal is to predict a context rating when the movie has already been rated, but only in one or more different contexts. This question would be useful, for example, in determining which context would be the best to watch a given movie in. We ran the same algorithms on this data set. Additionally, we employed a baseline algorithm of outputting the average rating of the same movie by the same user in different contexts. The results are presented in Table 5–3 and Figure 5–3.

As expected, the algorithms in general show some improvement. This confirms that knowing the utility of an item in a different context is somewhat useful for predicting it in another context. However, as the "guess mean" algorithm performs very poorly relative to the other algorithms, we can conclude that it is not enough to look only at previous ratings for the context. Here the normal linear algorithm is not sufficient either. The Hierarchical Bayesian algorithm has a much smaller median and mean error both per user and overall with p-values 8.61E-5 and .0038, respectively. This shows that the Bayesian algorithm is better at predicting ratings of a movie given that it has already been rated in a different context. The linear algorithm does, however, have the same F-Score as HBA, showing the overall recall and precision of the algorithm does perform similarly.

Table 5–3: The results of the recommender when using cross validation to answer the second question. Bold items mean optimal relative performance. The p-value for the mean is .0038 and median is 10E-4.

Algorithm (DCE)	MAE	Median	MAE PU	Median PU	F-Score
LSRIC	2.1060	1.991	1.9840	1.8796	.4575
LSLSU	2.8094	2.25	2.8017	2.2807	.2989
Guess Mean	2.1250	2.1667	3.785	3.375	.1701
IBCF	2.1626	2.0319	2.0260	1.9837	.3007
GP	2.0810	1.7858	1.961	1.7096	.3289
HBA	**1.8236**	**1.3823**	**1.8071**	**1.5151**	**.4868**

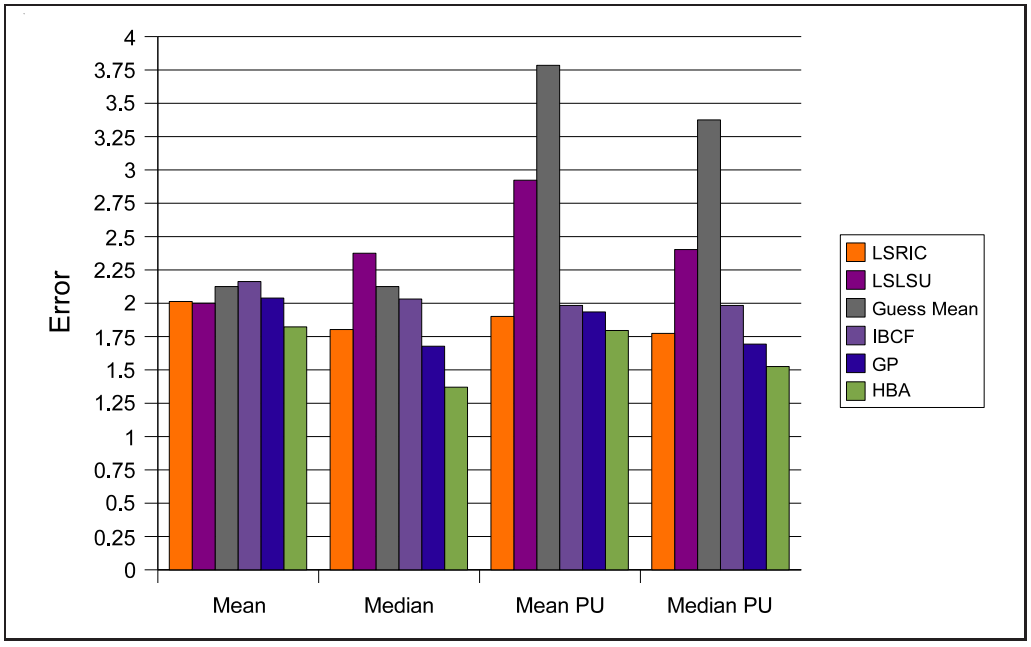
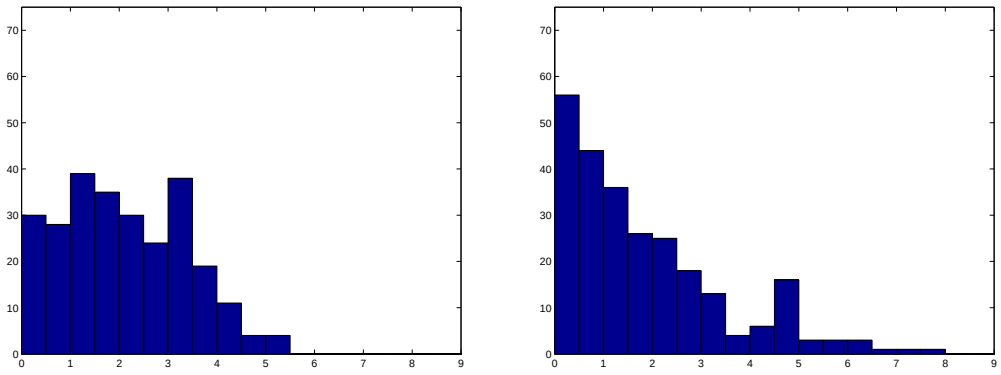


Figure 5–3: A comparison of the errors of the various algorithms at predicting a rating for a movie already rated, but in a different context. The Bayesian algorithm performs best in all cases. The difference in the median error (both per user and separately) between baseline linear and per user algorithms with the Hierarchical Bayesian algorithm is statistically significant.



(a) Linear Histogram Error

(b) Bayesian Histogram Error

Figure 5–4: Two histograms showing the error of the linear algorithm (left) and the Bayesian algorithm (right) in answering for an old item in a new context. Once again, the Bayesian algorithm has many more small errors, but a few large errors.

5.5 Using a Correlation Matrix: Synthetic Results

The second part of our algorithm involves using a correlation matrix to aggregate contexts. We ran our experiments on a synthetic data set again. We compared the Aggregate Contexts Hierarchical Bayesian (ACHB) algorithm with the Hierarchical Bayesian algorithm with no preprocessing. See Table 5–4. By aggregating contexts (ACHB), we improve the recommendations by a statistically significant amount for both the mean and median absolute error, with p-values of .0040 and .0295, respectively, over the standard Hierarchical Bayesian algorithm.

Clearly the more data we have per context, the easier it will be to learn the weights. The results on the synthetic data show that if we have enough ratings per context, we can combine the data from the contexts. This is useful because the amount of ratings needed to estimate the correlation matrix is independent of

Table 5–4: The results of the recommender when using cross validation on a synthetic data set. Based on 348 synthetic ratings. The aggregation causes a statistically significant improvement with p-value .0295 for the mean and .0040 for the median.

Algorithm (SCR)	**Mean Absolute Error**	**Median Absolute Error**
LSRIC	2.3065	1.4822
HBA	2.0344	1.2281
ACHB	**1.7060**	**.9882**

Table 5–5: A comparison of pre-processed data for a new *item* in which contexts have been merged using a correlation matrix with the original data. In each case the Hierarchical Bayesian algorithm was run. This is the case where the movie has not been rated by the user.

Algorithm (SCR)	**MAE**	**Median**	**MAE PU**	**Median PU**	**F-Score**
HBA	2.1381	**1.6257**	2.0103	1.6533	.4079
ACHB	**2.1102**	1.6867	**2.0063**	**1.6615**	**.4156**

the number of features used. Thus in a domain where it is necessary to store more features, we can use the correlation matrix.

Generating synthetic data makes several assumptions, so in the next section, we perform our experiments on a real data set.

5.6 Using a Correlation Matrix: Real Data

There are several possible ways to compute the correlation matrix. We experimented with the approaches outlined in Section 3.6 and found similar results in all cases with rank difference being slightly better. Hence we used it for the following tests. We compared the Hierarchical Bayesian algorithm run on the original data with one run on data in which contexts were aggregated. The results are summarized in Tables 5–5 and 5–6, and Figures 5–5 and 5–6.

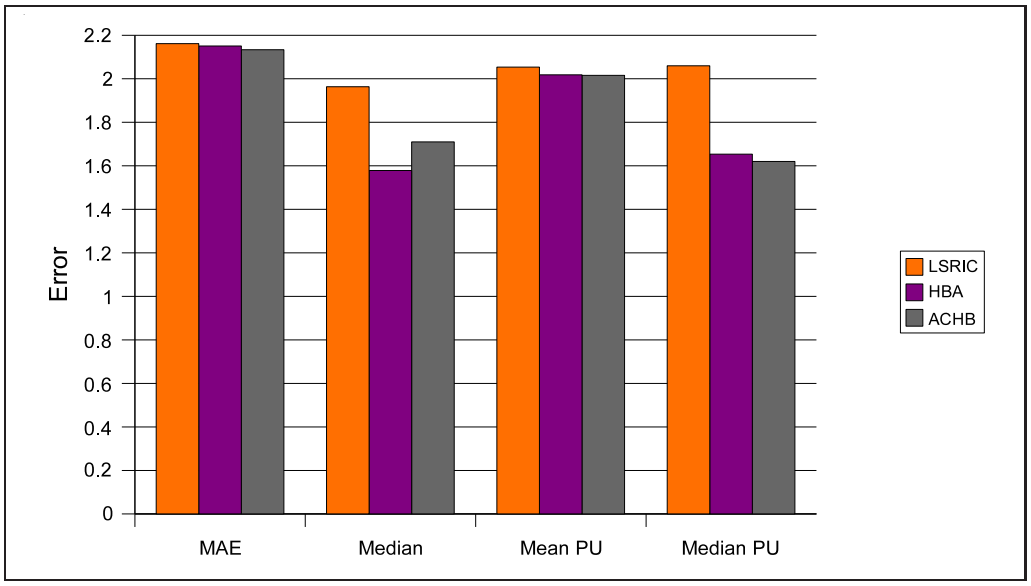


Figure 5–5: A comparison of the errors of the Hierarchical Bayesian algorithm when preprocessing is done using a correlation matrix (ACHB) versus when it is not done (HBA) to answer for a new item.

Table 5–6: A comparison of pre-processed data for an already rated item in a new *context* in which contexts have been merged using a correlation matrix with the original data. In each case the Hierarchical Bayesian algorithm was run. This is the case where the movie has already been rated by the user in a different context.

Algorithm (DCE)	**MAE**	**Median**	**MAE PU**	**Median PU**	**F-Score**
HBA	**1.8236**	1.3823	1.8071	1.5151	.4868
ACHB	1.8276	**1.3709**	**1.8011**	**1.4937**	**.5000**

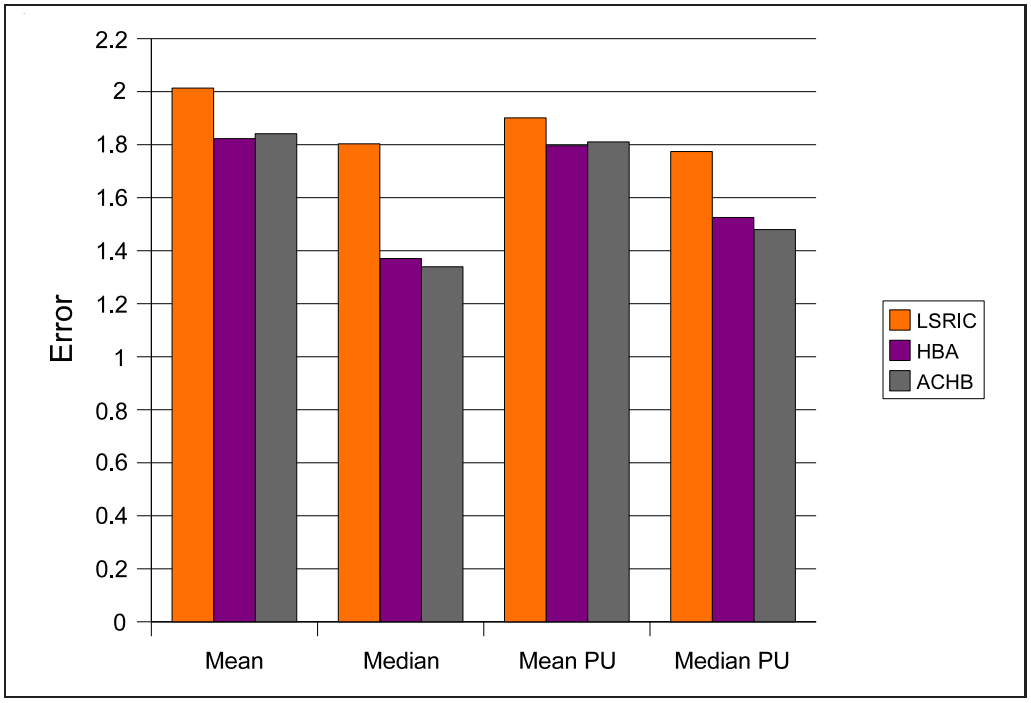


Figure 5–6: A comparison of the errors of the Hierarchical Bayesian algorithm when preprocessing is done using a correlation matrix (ACHB) versus when it is not done (HBA) to answer for an old item in a new context.

The results show a slight improvement in most measurements, but a decline in the median measurement in the first question. In answering for a previously rated item in a different context, the algorithms again are similar, with ACHB performing slightly better. The differences are not, however, statistically significant. While there are some users for which the correlation matrix produces an improvement over the Bayesian approach, the results are not strong enough to make any definite conclusions. Since there is slight improvement in most measurements, however, we are encouraged by the results and use the correlation matrix preprocessing in our recommender.

5.7 Feature Selection

As the dimensionality of our solution is higher because of context information, we seek to reduce the number of features. We choose features based on which features the user has selected. However, we can choose to use all of these or only some of them. By setting a limit of n features used, we can assure that the dimensionality is bounded by $3n$. We compared the results as n varies. We also included a PCA experiment with a corresponding number of features. These results are illustrated in Figure 5–7 and Figure 5–8.

Figure 5–8 shows that in the Hierarchical Bayesian algorithm, as the number of features increases, so does the accuracy of the predictions. However, once the number of features becomes too large, the error is increased, suggesting that the algorithm is over-fitting the training data. The curve of the baseline linear algorithm shows less variation in the error as the number of features increases. Despite this, however, the error on the Hierarchical Bayesian model is smaller. This shows that the improvement

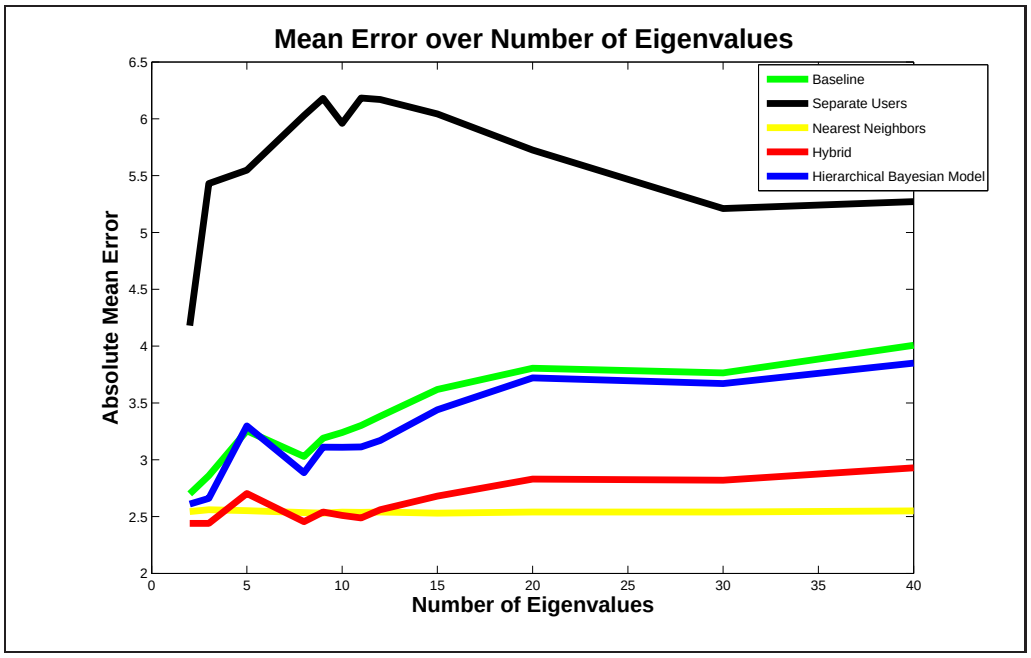


Figure 5–7: A graph showing the amount of error of the various algorithms as a function of how many features were used. The best results for each algorithm occur with more than ten features and less than fifteen.

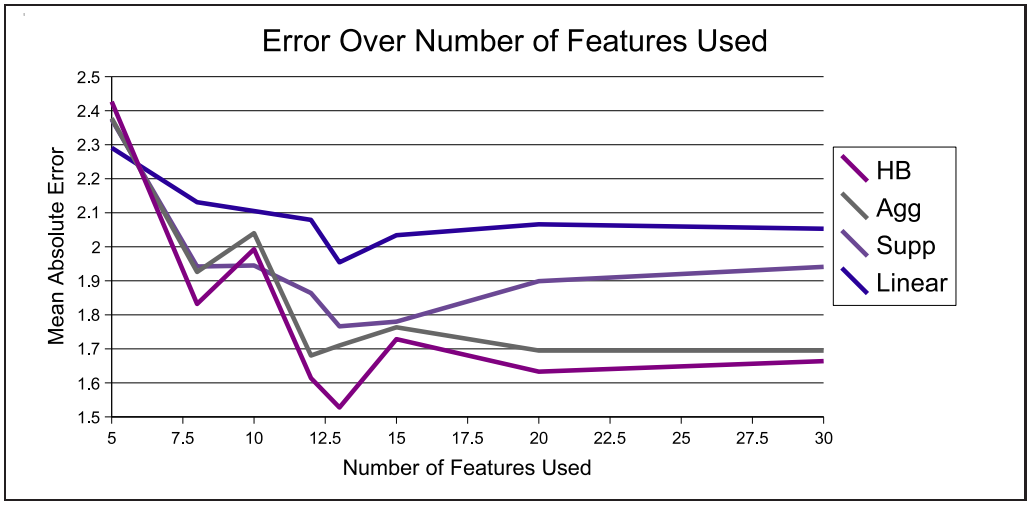


Figure 5–8: A graph showing the amount of error of the various algorithms as a function of how many eigenvalues were chosen. Note that for each algorithm except for separate users the best performance is around eleven eigenvalues. For the separate user algorithm, the error is so high that it is attributed to noise anyway.

gained by the Hierarchical Bayesian algorithm over the linear algorithm is reasonably robust, but if too many features are present, it will disappear.

The error using PCA is much higher than the error when we simply select features. This makes sense since PCA finds hybrid features which may or may not have any real world significance. By using a subset of the features the user has specifically told us, we are able to select features that are definitely (with some small amount of noise) important to the user.

5.8 Providing Recommendations

With the ability to predict the usefulness of items, we can provide context dependent recommendations. Since the best F-Score in recommendng unseen items in our experiments was using ACHB, we use this algorithm to give recommendations in a specific context. This is since this algorithm was best at selecting the top N

Table 5–7: Examples of predictions of which movies to watch in different contexts.

Generic	Guys Night Out	Romantic Evening
The Net	Ace Ventura	Simple Twist of Fate
The Transporter	Die Hard	Mona Lisa Smile
Star Trek	Highlander	The Terminal
Fantastic Four	Mortal Kombat	The Net

Table 5–8: Examples of recommendations of which context to watch a movie in.

Movie Name	Best Context	Worst Context
The Matrix Reloaded	Guys Night Out	With Children
Billy Madison	With Children	Romantic Evening
Golden Eye	Guys Night Out	With Children
Miracle on 34th St.	Romantic Evening	Halloween Movie
Pulp Fiction	Feeling Thoughtful	Guys Night Out

items in answering for an unwatched item. An example of these recommendations are given in Table 5–7.

Another variation on the movie recommendation problem is the *context recommendation problem*. This problem refers to the case where the user has already chosen the item she wants to watch and wishes to have the best *context* recommended to her. This occurs, for example, after a user receives a DVD as a gift. Several examples of context suggestions are given in Table 5–8. For example, the first entry in the table is the movie *The Matrix Reloaded*, which is best to see in the context of *Guys Night Out* and worst to see in the context *With Children*. In Table 5–9, we have listed all of the contexts in the Recommendz system. Since users can add their own contexts, this list can easily be extended.

Finally, we can use the correlation matrix to calculate the contexts that are most often correlated and the contexts that are least often or even negatively correlated.

Table 5–9: A list of all the contexts in the Recommendz system

Context Name
Date/Romantic Evening
Halloween
Christmas
With Friends
With Children
At a Theatre
As a Rental
Guys Night Out
Feeling Thoughtful/Mellow
Relaxing
Need to Make an Informed Decision
Needing a Laugh
By Myself
Girls Night Out
Relaxation
With Family

The most and least correlated contexts are presented in Table 5–10. These correlations are computed over *all* users by taking a weighted average of the correlation matrix of each user. The weights are the number of movies rated in both contexts for each user. Note that this approach does not necessarily guarantee properties such as transitivity of correlations. For example, if user u_1 has rated similarly many movies in contexts c_1 and c_2, then c_1 and c_2 will be highly correlated in $u_1's$ correlation matrix and hence the overall matrix. Similarly if user u_2 has rated similarly many movies in contexts c_2 and c_3, then c_2 and c_3 will be highly correlated in the overall correlation matrix. However, if neither of these users has rated many movies in both c_1 and c_3, then there will be no correlation in the overall matrix.

Some of the correlations make intuitive sense while others do not. We should not, however, read too much into the correlations as each user can have different tastes. Moreover, the amount of overlap is quite small so these are not definitive. Nevertheless, we will point out a few correlations. "Christmas" movies and "With Children" certainly makes sense. An interesting correlation is "Guys night out" and "Girls night out." Intuitively, one might think men and women have different tastes when with friends, but perhaps not. In fact, this supports our hypothesis that different users will have different interpretations of contexts when we created the "Guys night out" context, we assumed it was to be used by males watching a movie with other males and "Girls night out" was created to be used by females watching a movie with other females. However, some users have rated movies in both, contradicting our intended meaning of the contexts. Because the Hierarchical Bayesian algorithm is content based, rather than collaborative filtering based, this does not cause a problem.

5.9 Further Analysis

We have demonstrated the usefulness of context in providing recommendations. The Hierarchical Bayesian algorithm performs better than all baseline algorithms in every form of measurement we examined. Additionally we have shown that using a correlation matrix to aggregate contexts before running the Hierarchical Bayesian algorithm can improve accuracy. The improvements are statistically significant. However, there are some remaining issues.

Table 5–10: The most and least correlated movies and the number of movies (N) that correlation is based on. Note correlations based on fewer than 3 common movies are ignored as they are too susceptible to noise.

Context 1	**Context 2**	**Correlation**	**N**
Christmas	With Children	.9824	3
Theater	Rental	.9780	4
By Myself	Christmas	.9737	5
Guys night out	Girls night out	.9701	9
Girls night out	Needing a laugh	.9563	3
Friends	Rental	.9522	6
Christmas	Relaxation	.9488	3
Friends	Guys night out	.9487	5
By Myself	Girls night out	.9487	5
By Myself	Feeling Thoughtful	.8963	9
...	...	...	...
With Children	Feeling Thoughtful	.1078	10
Christmas	Needing a Laugh	.0937	3
Date	Guys Night Out	.0102	14
Friends	Relaxation	.0229	5
By Myself	Needing a Laugh	-.0072	16
Christmas	Theater	-.0610	4
Girls Night Out	With Children	-.9400	3

One important factor in contrasting our results to others is the difficulty of comparing results taken on different data sets. In our set, the average user rated approximately 30 movies. Other studies have shown smaller errors using larger training sets (75-100 ratings per user) [39], but it was infeasible to get this large a data set. Additionally, the main goal of this work was to demonstrate the effectiveness on a relatively small data set.

In Figures 5–9 and 5–10, we have plotted a visualization of the sparsity of the ratings matrix. To visualize the matrix we have plotted a point for every user-item pairing given in a context independent setting. In Figure 5–10, we have turned a 3D matrix into a 2D matrix by putting each context's matrix adjacent to each other. We can see from this how sparse this matrix is, showing that it is virtually impossible and very inefficient to not link contexts.

It is difficult to collect context data because standard data sets such as the Netflix^{TM}set do not keep track of context information and are thus not applicable to our work. Moreover, we wanted to demonstrate the effectiveness of the Hierarchical Bayesian algorithm on a small number of data points. Given enough ratings, it would even be reasonable to treat each context as a different user. However, our algorithm does not require a user to rate dozens of movies in each context. The algorithm gives a good approximation for each separate context even when only four or five movies have been rated in that context, thus dealing with the "new context" problem. With a smaller mean and median error than the baseline linear regression models have, the Hierarchical Bayesian model accomplishes what we want. It shares information between contexts without requiring all information be shared.

When all contexts are left out of the training set for a specific movie, the Bayesian algorithm has a smaller median error without aggregating contexts first. However, when we leave the ratings of the same item in different contexts in the training set, the Bayesian algorithm is improved with the pre-processing step.

When we leave only the one rating out, but leave all the others from different contexts in, as expected, the ratings are, in general, closer to their actual ratings. This confirms that ratings in one context are correlated with ratings in another context. However, the ratings of the baseline algorithm do not improve by nearly as much as the Bayesian algorithm's ratings. This is because the linear algorithm does not allow much variety from one context to another. Since it ignores the context dimension, it will have a point in the training set for every context that the item is rated in, meaning when we calculate a best fit line, we will factor in a possibly incorrect point.

Our aim was to show that while the preferences from one context are correlated, they are not determinative. Since the results of the algorithms are all better when we leave in ratings from other contexts, they clearly are correlated. However, the fact that HBA is able to handle the extra rating better than linear shows that they are not determinative. Otherwise the "Guess Mean" or linear algorithm would perform as well as HBA or perhaps even better due to a simpler model.

To answer the "Different Context Evaluation" problem, we find that the Bayesian model has the lowest median error. The results show that the Bayesian model is able to use the non-context information to improve overall ratings than the other algorithms as the median here is significantly lower than the others. While the results

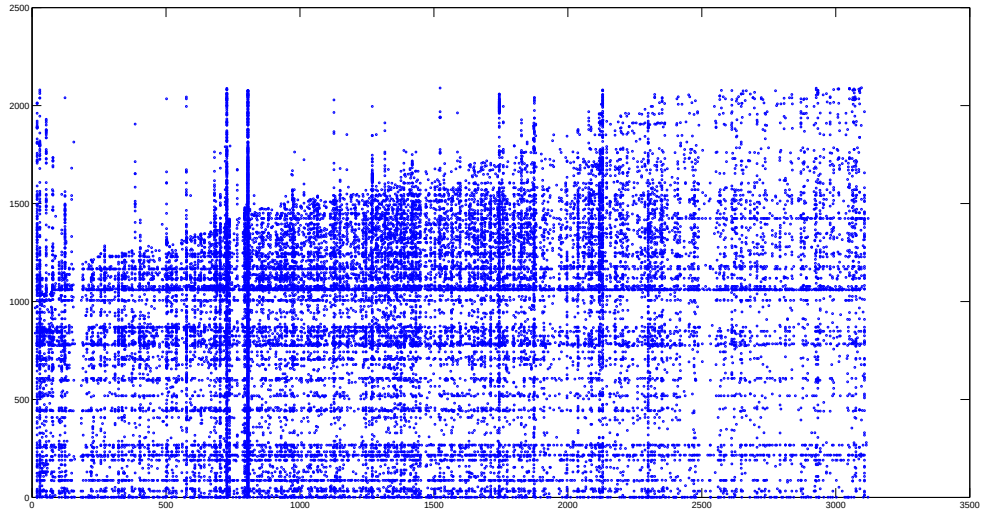


Figure 5–9: A visualization of the sparsity of the context independent matrix on *Recommendz*. Points are plotted whenever an item-user pairing is given.

in the F-score are unclear, we consider the mean and median error a more appropriate measure to answering the second question. While in the first question, it did not matter how you ranked poor movies as they would not be presented to the user anyway, in this case it does matter. The user will ask about a specific movie in a specific context, and we would like to give as accurate an answer as possible. It does not necessarily make sense to make a recommendation for a specific context when we allow movies already seen in different contexts to be recommended. For example, suppose a user likes one or two movies very highly in all contexts. This is a very reasonable assumption since many users would enjoy their favorite movie in almost every settings. In this case, the item similarity algorithm is almost guaranteed to give the correct answer since the closest item to a movie is always itself. Thus we

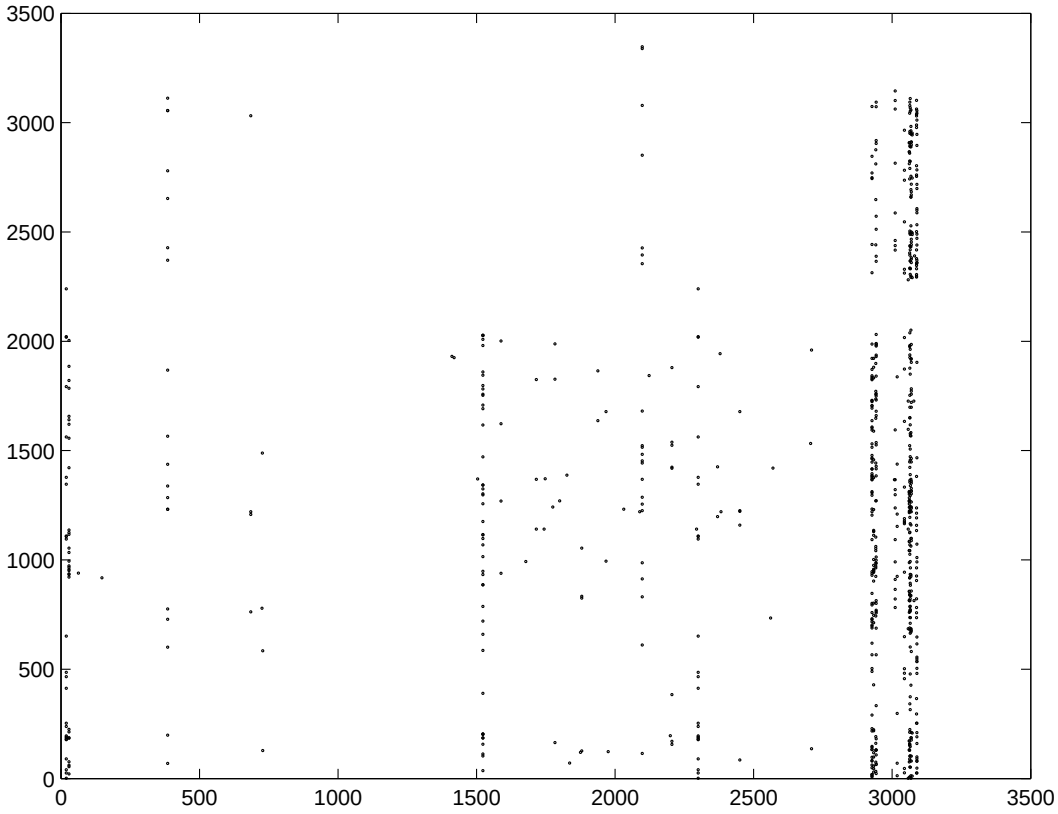


Figure 5–10: A visualization of the sparsity of the context independent matrix on *Recommendz*. Points are plotted whenever an item-user-context pairing is given. As the context dimension is discrete, we have morphed the 3D matrix into a 2D matrix by simply putting each different context matrix adjacent.

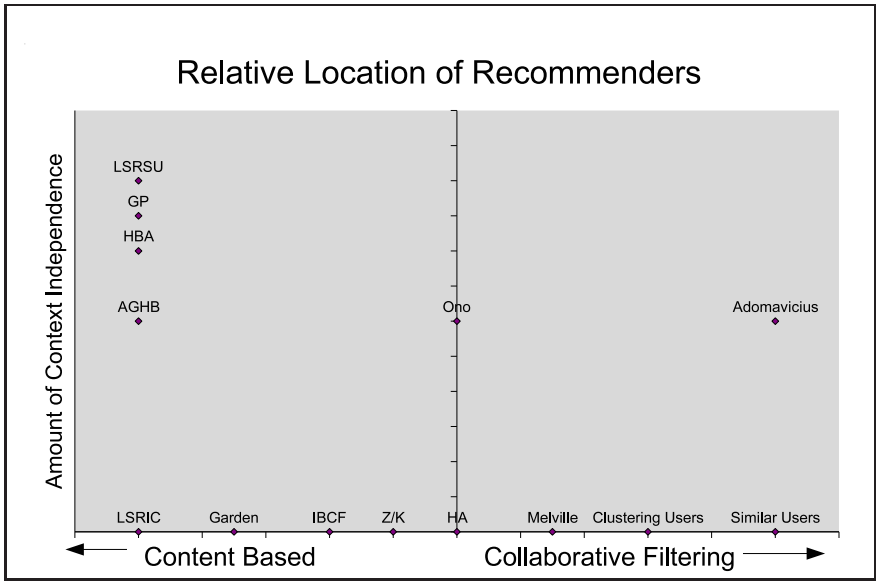


Figure 5–11: A plot showing where the algorithms we compared lie along the same two dimensions as before: CB vs CF and context-independent vs. context-dependent.

will always recommend movies that the user has already seen several times, just we will suggest that the user watch them in a different context.

In Figure 5–11, we have added the algorithms we compared to the same graph as Figure 2–3. Here we see that HBA is a good compromise between linking contexts too much and too little. By aggregating contexts, we make each context more dependent on one another, which is good in some cases but not in others.

The results of our experiments show that by ignoring context completely, a recommender system is ignoring an important dimension of the data. However, this dimension is not orthogonal from the other dimensions as ratings in one context are statistically correlated with ratings in a different context. Because of these two important notions, the Hierarchical Bayesian model performs better than a linear baseline model. It is successfully able to share information user preferences between

contexts whilst maintaining some degree of independence between contexts. In addition, aggregating contexts using a correlation matrix between contexts is promising as the F-Score is highest although the results are not conclusive.

CHAPTER 6
Conclusions

6.1 Summary

We designed and implemented a Hierarchical Bayesian algorithm in order to provide context-dependent recommendations. We first verified that context actually matters and that it can not be dealt with in trivial manners. We used a Hierarchical Bayesian model and Expectation Maximization to answer two questions: "What movie should I watch in context X?" and "Given that I gave a rating to movie M in context X, what would I think of movie M in context Y?" We then tested the Hierarchical Bayesian algorithm in the movie domain and the algorithm performs much better than baseline algorithms that ignore context (linear, item based collaborative filtering) or handle it in trivial ways (separate users). We experimented with creating a correlation matrix to relate contexts and improve predictions. When we aggregate related contexts, there is some improvement in F-Score, but the improvement is not statistically significant.

The first set of experiments we ran verify how important it is to use an algorithm such as the Hierarchical Bayesian algorithm. After determining that ratings vary from context to context, we showed a statistical correlation between ratings in different contexts. This means that the best model must consider context but at the same time share information between contexts. A Bayesian network thus fits the model very well.

We assumed that the user's preferences in each context can be modeled by a set of linear weights. However, it is not necessary to enforce a linear requirement. The only requirement is that each context has to use the same basic model in order to link them. That is, if one context is linear, then all of the contexts have to be linear. One issue with the linear model is that there are some features which are best in either high or low amounts, but not in medium amounts. This could be modeled by a parabolic graph. Under the linear model, the ratings are given by:

$$r(u, i, c) = \mathbf{i} \cdot \mathbf{W_{u_c}} + b \tag{6.1}$$

Under a parabolic model they would be given by

$$r(u, i, c) = \mathbf{i}^2 \cdot \mathbf{Q_{u_c}} + \mathbf{i} \cdot \mathbf{W_{u_c}} + b \tag{6.2}$$

This model could be incorporated into a Hierarchical Bayesian model using similar methods as applied for the linear model. Now for each context the set of variables to learn are the quadratic weights $\mathbf{Q_{u_c}}$ and the linear weights $\mathbf{W_{u_c}}$. These are linked as they come from a common Gaussian or perhaps two or more different Gaussians. The Expectation and Maximization formulas will change slightly because the maximums will occur at different places, but the essence of the method will remain unchanged.

Since the original model is content-based, we do not rely on other's definitions of a context. This allows users to create their own contexts and provide feedback to the user after only a handful of ratings. Other methods such as [1] require collaborative filtering techniques and thus assume that contexts are either similar or different over all users. However, the Hierarchical Bayesian algorithm allows one user to find two

contexts similar and another user to find them different. It also allows for users to create several unique contexts, such as with a specific friend.

6.2 Future Work

In this final section, we will discuss some of the further directions this research can be taken. These include ways to model more complicated user preferences by storing each item as a set of real numbers instead of discrete values ("low," "medium," and "high") and also weighting features by importance. Additionally, we can extend the correlation matrix work. We will now describe these extensions in more detail.

The most basic extension is to extend our model beyond the linear model. There are several limitations of the linear model in each context, the most obvious of which is accounting for parabolic preferences. While our model does learn different weights for "low," "medium," and "high" frequencies of features, this requires extra dimensions in our solution. More ideal would be to store the presence of a feature as a real number and then calculate a parabolic regression. This would allow the dimensionality of the solution space to be reduced but there is a trade off as we would no longer be able to store the feature vectors as probability distributions.

Each feature may need to be weighted by its importance. One idea is to use Boosting for regression such as in [4], [13], [33]. Boosting is an established machine learning technique for for combining simple "rules of thumb" using a voting scheme. In this case, the "rules of thumb" could be the predictions based on only one feature. AdaBoostRT [33] is an algorithm that computes the weights of the voting scheme. This has the potential to be better as some features may be more important than others. There may be some features that when present trump all other features.

For example, when watching a movie with young children, it may be a parent will dislike a movie with violence no matter what else is in the movie. This problem could be addressed by using a model that allows for step functions. We did not investigate these techniques further as we consider this an orthogonal problem to that of providing context dependent recommendations. This thesis focuses on techniques for relating contexts to each other and not techniques for representing each individual context.

Another important idea is that of a correlation matrix to relate contexts before using the Hierarchical Bayesian algorithm This approach worked well on synthetic data but had only limited success in our experimentation on real data. This may have been due to the small data sample as the correlations were based on only a few movies. There were several users for which aggregating contexts improved recommendations essentially by smoothing the data. One might also examine the effect of choosing different techniques for computing the correlation matrix perhaps using collaborative filtering techniques to find similar contexts. One approach is to cluster the contexts based on their similarity of ratings such as in [11]. Another idea is to make the threshold for aggregating contexts a function of the number of movies rated in the context being merged. If a context has many ratings, it is less necessary to merge it as we can learn the weights from the data directly.

We also would like to improve the correlation matrix by examining item similarity as well. Currently, we only consider contexts to be similar if the same item has been rated similarly by the same user in multiple contexts. However, we could

expand our technique to look at similar movies. That is, if we calculate item similarity using standard item collaborative filtering techniques, we can correlate contexts when *similar* items are rated similarly. This would allow us to increase the size of the data set that the correlation matrix is computed based on.

The main goal of the correlation matrix is to make an educated guess of which contexts are the same so as to reduce the ratio of the number of contexts to the number of training points. That is, we use a heuristic to merge contexts or create data. The default training set is susceptible to noise because there are only a handful of movies rated in each context. We can reduce this by smoothing it. However, if a context already has a large amount of training data, then the smoothing is not necessary and we only will increase the noise due to the heuristic, particularly when the heuristic is based on too small a sample. One other approach is to experiment with the error as a function of the size of the contexts.

Another related approach is to supplement the list of ratings for a given context based on the correlation matrix. We can take ratings from another context, put them in the desired context, and then weight each result based on the correlation matrix. We then need to extend the EM formulas to include weighted examples. The experiments we ran with this approach were not successful as the error increased in all measurements, albeit by a small amount, and thus we do not present the results in this thesis. It is possible that improving the correlation matrix will immediately lead to improvement in this algorithm as the weights will be more accurate. There are several other techniques to try. First, the weighting can be changed so as to be a function of the certainty of both the specific context and the correlation. If

the correlation between two contexts is very high but based on a small sample, then perhaps the correlation is not actually high and we should not weight the movie as highly. On the other hand, if there is a context with very few movies rated in it, we could use the information from other contexts to a larger extent. A similar heuristic could be used to decide when to aggregate contexts. When a context has only been rated a few times, then it is more important to aggregated.

By being content-based, our model allows users to create their own contexts. However, like all other content-based recommenders, it suffers from the new user problem. The context based model proposed by Adomavicius in [1] addresses the new user problem but as it is collaborative filtering based does not allow users to create their own contexts. We could try combining these two methods using a hybrid recommender scheme, perhaps as simply as by a linear combination of the two. It might also be possible to create a content based hybrid similar to the method experimented on in the thesis except combining the learnings with weights other than .5.

All of the experiments on real user data that we carried out were in the movie domain. There is no reason, however, that these methods can not be applied to other domains as long as an item in the domain can be described as a set of features. We have demonstrated that it is useful to maintain contextual information in the movie domain. Experimentation should be performed in other domains such as books, blogs, and restaurants to determine whether ratings are a function of context in these domains as well.

We have shown that context is important to providing good movie recommendations and created an algorithm to do so. This algorithm can be used for any content-based model. We have shown that a linear algorithm can be improved by combining it with a Hierarchical Bayesian algorithm. This was verified on both synthetic data and real data. We have also demonstrated on synthetic data that using a correlation matrix is useful to aggregate contexts, which reduces the dimensionality of the solution. This is an important step to improving recommender systems.

References

[1] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, January 2005.

[2] Gediminas Adomavicius and Er Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[3] Robert Bell and Yehuda Koren. Improved neighborhood-based collaborative filtering. In *Proc. KDDCup*, pages 7–14, 2007.

[4] Alberto Bertoni, Paola Campadelli, and M. Parodi. A boosting algorithm for regression. In *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*, pages 343–348, London, UK, 1997. Springer-Verlag.

[5] Daniel Billsus and Michael J. Pazzani. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.

[6] Christian Bomhardt. Newsrec, a svm-driven personal recommendation system for news websites. In *WI '04: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 545–548, Washington, DC, USA, 2004. IEEE Computer Society.

[7] Keith Bradley and Barry Smyth. Improving recommendation diversity. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, pages 85–94, 2001.

[8] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, July 1998.

[9] G. Y. Chen and P. Bhattacharya. Function dot product kernels for support vector machine. In *ICPR '06: Proceedings of the 18th International Conference*

on Pattern Recognition, pages 614–617, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Mark Claypool, Anuja Gokhale, Tim Mir, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *In Proceedings of ACM SIGIR Workshop on Recommender Systems*, 1999.

[11] Mark Connor and Jon Herlocker. Clustering items for collaborative filtering. In *Proceedings of the ACM SIGIR Workshop on Recommender Systems*, 1999.

[12] Gregory Dudek and Matt Garden. On user recommendations based on multiple cues. In *WI/IAT 2003 Workshop on Applications, Products, and Services of Web-based Support Systems*, pages 139–143, 2003.

[13] Nigel Duffy and David Helmbold. Boosting methods for regression. *Mach. Learn.*, 47(2-3):153–200, 2002.

[14] Matthew Garden and Gregory Dudek. Semantic feedback for hybrid recommendations in recommendz. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 754–759, Washington, DC, USA, 2005. IEEE Computer Society.

[15] Matthew Garden and Gregory Dudek. Mixed collaborative and content-based filtering with user-contributed semantic features. In *Proceedings of the 21st AAAI National Conference on Artificial Intelligence (AAAI'06)*, Boston, Massachusetts, July 2006.

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

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

[18] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[19] Zan Huang, Hsinchun Chen, and Daniel Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):116–142, 2004.

[20] Shane T. Jensen, Blake Mcshane, and Abraham J. Wyner. Hierarchical bayesian modeling of hitting performance in baseball, Retrieved Feb 2009 from http://arxiv.org/abs/0902.1360.

[21] Rong Jin and Luo Si. A study of methods for normalizing user ratings in collaborative filtering. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 568–569, New York, NY, USA, 2004. ACM.

[22] Rong Jin, Luo Si, ChengXiang Zhai, and Jamie Callan. Collaborative filtering with decoupled models for preferences and ratings. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 309–316, New York, NY, USA, 2003. ACM.

[23] Charles Kemp, Amy Perfors, and Joshua B. Tenenbaum. Learning overhypotheses with hierarchical bayesian models. *Developmental Science*, 10(3):307–321, May 2007.

[24] Ronny Luss and Alexandre d'Aspremont. Support vector machine classification with indefinite kernels. In *Advances in Neural Information Processing Systems*, pages 953–960, 2007.

[25] Prem Melville, Raymond J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering. In *In Proceedings of the 2001 SIGIR Workshop on Recommender Systems*, 2001.

[26] Chihiro Ono, Mori Kurokawa, Yoichi Motomura, and Hideki Asoh. A context-aware movie preference model using a bayesian network for recommendation and promotion. *User Modeling 2007*, pages 247–257, 2007.

[27] John C. Paolillo and Shashikant Penumarthy. The social structure of tagging internet video on del.icio.us. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, page 85, 2007.

[28] Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13(5-6):393–408, 1999.

[29] Al M. Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. Mcnee, Joseph A. Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134, New York, NY, USA, 2002. ACM Press.

[30] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstorm, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[31] Gerald Salton, editor. *Automatic text processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

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

[33] Dimitri P. Solomatine and Durga L. Shrestha. Adaboost.rt: a boosting algorithm for regression problems. In *Proceedings IEEE International Joint Conference on Neural Networks*, volume 2, pages 1163–1168 vol.2, July 2004.

[34] Kirsten Swearingen and Rashmi Sinha. Interaction design for recommender systems. In *Designing Interactive Systems 2002. ACM*. Press, 2002.

[35] Lyle Ungar and Dean Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*. AAAI Press, Menlo Park California, 1998.

[36] Vishwa Vinay, Ingemar J. Cox, Ken Wood, and Natasa Milic-Frayling. A comparison of dimensionality reduction techniques for text retrieval. In *ICMLA '05: Proceedings of the Fourth International Conference on Machine Learning and Applications*, pages 293–298, Washington, DC, USA, 2005. IEEE Computer Society.

[37] Kai Yu, Volker Tresp, and Anton Schwaighofer. Learning gaussian processes from multiple tasks. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 1012–1019, New York, NY, USA, 2005. ACM.

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

[39] Yi Zhang and Jonathan Koren. Efficient bayesian hierarchical user modeling for recommendation system. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 47–54, New York, NY, USA, 2007. ACM.

[40] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 22–32, New York, NY, USA, 2005. ACM.