Context Dependent Movie Recommendations Using a Hierarchical Bayesian Model

Daniel Pomerantz and Gregory Dudek

School of Computer Science
McGill University
Montreal, Quebec {dpomeran,dudek}@cim.mcgill.ca

Abstract. We use a hierarchical Bayesian approach to model user preferences in different contexts or settings. Unlike many previous recommenders, our approach is content-based. We assume that for each context, a user has a different set of preference weights which are linked by a common, "generic context" set of weights. The approach uses Expectation Maximization (EM) to estimate both the generic context weights and the context specific weights. This improves upon many current recommender systems that do not incorporate context into the recommendations they provide. In this paper, we show that by considering contextual information, we can improve our recommendations, demonstrating that it is useful to consider context in giving ratings. Because the approach does not rely on connecting users via collaborative filtering, users are able to interpret contexts in different ways and invent their own contexts.

Key words: recommender system, Expectation Maximization, hierarchical Bayesian model, context, content-based

1 Introduction

Recommender systems are becoming more and more widespread with many websites such as Amazon.com$^{\text{TM}}$able to provide personalized recommendations as to what products a customer will like. If the customer likes the product that has been recommended, then she is more likely to both buy the specific product and continue shopping there in the future.

One problem with many current recommender systems is they fail to take into account any *contextual* information. That is, they do not deal with important questions such as when, where, and with whom you will use the item. 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. Those systems that do incorporate contextual information do so in a way that does not allow users to define their own contexts.

Traditional recommender systems can only answer the question, "What item should I use?" In this paper, we focus on the movie domain, but the ideas can be generalized to other domains such as books or online blogs. We will demonstrate the usefulness of storing contextual information and adapt the hierarchical

Bayesian model described in [13] and [11] in order 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?" Throughout the rest of this paper, we will refer to these two problems as "Standard Context Recommendation" and "Different Context Evaluation."

One way to model a user in different contexts is to create separate movie accounts for each context. This is not ideal, however, as it very often happens that *some* of the user's tastes are still the same in different contexts. Without sharing information between contexts, users would be forced to rate lots of movies in *each* context 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. The hierarchical Bayesian model is ideal for our purposes as it allows us to share information between contexts, but at the same time, it allows the preferences in different contexts to be entirely different. In this way, we avoid forcing each user to rate an excessive number of movies. These techniques can then be extended to other domains in addition to movies such as books or online blogs.

2 Background Information and Related Work

To make a recommendation, we need some way to predict a rating r_p or usefulness of a given product or movie m for a given user u. We can then select the products that have the highest usefulness to the user. This paper will discuss ways to improve the predicted rating of a product, since once we calculate this, we can easily make a recommendation by sorting.

The two most common techniques to predict the usefulness of a movie are *collaborative-filtering algorithms* and *content-based models.* In each case, we normally represent each movie as a vector with each entry in the vector representing the amount of one particular feature (e.g.humor, Brad Pitt, bad acting, etc). Collaborative-filtering based techniques determine a set of similar users. Once the system has determined similar neighbors, it can make a recommendation based on assuming that similar users will like similar movies.

Content-based approaches model a user by determining the important features to each user based on previous ratings that the same user has made. Using the model, they will recommend items that are similar to other items that the user has rated highly. This paper will focus on the content-based technique. There are several ways to model a user. One model is based on a linear approach. Alspector [2] suggests that the recommender system should store a set of weight vectors corresponding with the user's preference of each feature. Another commonly used approach is to predict the same rating as the nearest item or the average of the k nearest items (nearest neighbor) [10]. Here we briefly summarize these approaches.

2.1 Representing Users

One approach to modeling users is a linear model which proposes that every user can be modelled as a vector of real numbers. This vector relates to the movie vector in that each element represents how much the user likes the presence of the corresponding feature. Once we learn these weights, denoted by w_u, we can make a prediction r_p as to whether a user u will like a movie m based on:

$$r_p(u, m) = \overrightarrow{w_u}\overrightarrow{m} \ . \tag{1}$$

We can use machine learning algorithms to learn the weights given a set of training data. One method is to compute a least squares linear regression [12]. This corresponds with assuming that every actual rating r_a differs from the predicted rating by adding Gaussian noise. For space considerations, we refer the reader to [12] for more details of this method as well as other approaches to learning the weights such as Support Vector Machines and Naive Bayes models.

Another content-based approach that has been used is the nearest neighbor approach or nearest k-neighbors approach. In this non-linear method, 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. One measure used to find similarity is the cosine similarity measure [7]. Another possibility is to use the inverse Euclidean distance to calculate similarity. Once this is calculated for all rated movies, we select the k-most similar previously rated movies and calculate a weighted average of the k movies to make a prediction.

2.2 Dimensionality Reduction

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*, meaning these features redundant. Other features appear in only a few movies, and can be dropped without much information loss. There are several ways to reduce the dimensionality of the space. Goldberg et al. [4] suggest using a gauge set of movies. The idea here is to find an ideal set of movies which all users should be asked about. Similarly, one could create a gauge set of features. Some other possibilities are to reduce the dimensionality based on approaches using information gain or mutual information or Independent Component Analysis (ICA). We use Principal Component Analysis (PCA), a technique that is geared towards storing only the most useful data. For a further comparison on dimensionality results, see [9].

2.3 Connecting Users

One of the downsides of looking at users separately is a user has to rate several movies before being able to be given a useful prediction on a new movie. This is referred to as the "new user" problem. Zhang and Koren propose a hierarchical Bayesian model to solve this by relating each user's preference weights to each other. The model assumes that, as in Section 2.1, for any given user, there

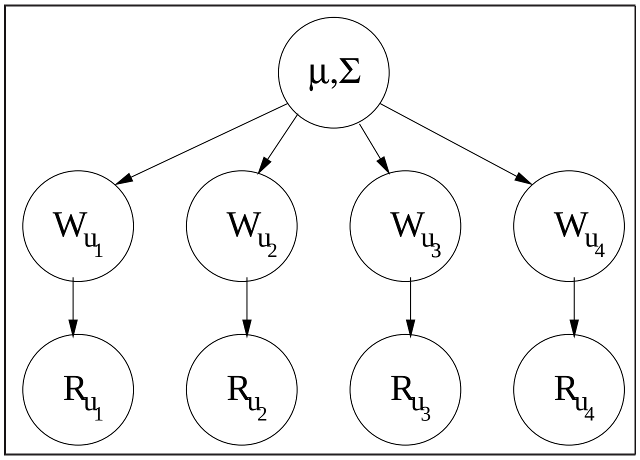

Fig. 1. Hierarchical model: First a mu and sigma are "chosen" for the entire "population." Then based on this, each user's weight vector is a Gaussian random vector. Finally, given the user's preference weights, the rating given to each movie can be determined, but it is not entirely deterministic due to noise. Note that R_i is observed.

is a linear relationship between the movie vector and the rating. The model relates each user's weights to each other. See Figure 1. It assumes that there is a mean vector μ and covariance matrix Σ^2 that exist for all users as the mean and covariance respectively of all user preferences. Each user's weight vector is then a Gaussian random vector with mean and covariance matrix μ and Σ^2 respectively. In other words, each user is merely a random sampling from the normal distribution. After determining the weight vector $\overrightarrow{w_u}$ for each user u, the rating for a movie with features $\overrightarrow{m}$ is a normal random variable with mean $\overrightarrow{w_u}\overrightarrow{m}$ and variance σ_u where σ_u is a noise factor that is calculated for each user. They use Expectation Maximization to estimate the unknown weights w_u. This solution is good for dealing with the new user problem because a new user is initially given a set of weights (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, any particular set of weights is allowed so that after rating enough movies, the user's weights under this algorithm will converge with the weights from a standard linear regression.

2.4 Context-Based Recommendations

Adomavicius and Sankaranarayanan [1] explore using context to find similar users and similar items. They start with standard collaborative filtering algorithms, which use a similarity measure between two items, and extend the measure to include additional context dimensions. Ono et. al. [6] design a Bayesian network. Contexts, users, and items all combine together to form "impressions" (e.g. funny, depressing, etc.) of the movie, which in turn leads to ratings. They estimate the probability of a rating given the user, context, and item.

The downside of these approaches is they require sharing information about contexts between users. 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 watching romantic movies on a date, there may be some users that prefer not. While it is theoretically possible to design an algorithm that determines which users treat contexts in one way, these are difficult parameters to estimate. Additionally, by designing a content-based model, we can easily allow users to add their own contexts.

3 Content-Based Context-Dependent Recommendations

Naively, one might think that modeling a user's preferences in different situations could be handled simply by considering each user as several different people. That is, for each user we maintain a different profile for every different possible context that they have rated movies in. However, the user's preferences in different contexts are possibly correlated even if they are not exactly the same. Since gathering enough data to accurately model every context separately is quite difficult, it is beneficial to use the ratings from one context to learn ratings in another. Otherwise every time a new context is added for a specific user, there would be no information about the ratings in that context. Thus you would suffer from a "new context problem."

The Hierarchical Bayes model is well suited for our situation because it can give a prediction for a context without requiring as many movies to be rated. In order to give a context-dependent recommendation, we adapt the model proposed by Zhang and Koren in [13]. Rather than each branch of the tree corresponding to a *user*, we design one tree for every user and let each branch correspond to a specific *context*. By incorporating an average weight and variance into our model, we help avoid the problem of over-fitting or ill-posed problems that otherwise would occur frequently in context ratings. Often while there are more movies rated than the number of dimensions of the feature space, there are not more movies rated in a specific context than the number of dimensions.

3.1 Estimating the Weights

We need to estimate the weights for each context. If we know the generative μ and Σ^2, then we can estimate the weights W_c of each branch using a linear regression with a prior. If we know the weights W_c of each branch, then we can estimate μ and Σ^2 using maximum likelihood. 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 guess of μ and Σ^2. We repeat this until the variables all stabilize. For spatial reasons, we do not present the derivation of the formulas here but merely present the resulting algorithm. See [13] and [11] for further information.

1. Make an initial guess for μ and Σ^2

2. E step: For each context c, estimate $P(w_c|R, \mu, \Sigma^2)$ 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 step 2, in order to estimate $P(w_u|R, \mu, \Sigma^2)$, for each context 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 $P(w_c|R, \mu, \Sigma^2)$ are:

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

$$\Sigma_c^2 = \left((\Sigma^2)^{-1} + \frac{S_{xx,c}}{\sigma_\epsilon^2} \right)^{-1} . \tag{3}$$

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 (i.e. Take the matrix composed of all the different feature vectors of movies that the user rated and multiply it by its transpose.) and $S_{xy,c}$ is the matrix created by taking the vector of movies rated and multiplying it by the actual ratings given.

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

$$\mu = \frac{1}{C} \sum_c w_c , \qquad \Sigma^2 = \frac{1}{C} \Sigma_c^2 + (w_c - \mu)(w_c - \mu)^T . \tag{4}$$

where C is the number of contexts for the user.

Looking at equation 2, we see that as the number of movies rated in a given context goes to infinite, the weights converge to the standard linear model because the overall mean and sigma become very small compared to the other terms.

3.2 Estimating the noise per user: σ_ϵ

In [13], it is assumed that σ_ϵ is given. In [11], they propose solving for σ_ϵ during the EM process as well during the M step by measuring the error on the training data. However, since we are assuming that the number of ratings in a given context is often less than the number of dimensions of the feature space, this spread is often quite low and is not a useful measurement. Since σ_ϵ represents the amount of noise added to the linear model, we estimate σ_ϵ heuristically by setting it equal to the variance of the error on the training data using *non-context* linear weights, which is the noise in the non-context linear model. This is done using the least-squares linear regression outlined in [12]. We then leave it constant throughout the EM algorithm.

3.3 Reducing the Dimensionality

By allowing users to have different weight vectors for each context we increase the dimensionality of the solution. If we normally had d weights to solve for, we now have to solve for cd weights where c is the number of contexts for the user. If a movie is represented as a vector of size n where n is the number of features, then we will now have a dimensionality of cn, which is in general much larger than the size of the training set. Thus it is often necessary to reduce the dimensionality of the space.

We chose to solve this problem using Principal Component Analysis (PCA) for two reasons. The first is that it is a relatively efficient model. We can pre-compute the eigenvalues and eigenvectors over the entire movie database thus allowing us to quickly give a rating at run time. The other benefit of the algorithm is we can judge the amount of precision lost by the reduction and adjust the number of eigenvalues used accordingly.

4 Experiments

We gathered data using the online website Recommendz. This site is accessible at `http://www.recommendz.com` and has been used in previous papers [3]. The site has over 3000 users and has been running for several years. Unfortunately, most of the ratings previously given are in context-independent settings and are not useful to these experiments. However, we did gather context-dependent data from fifteen users who rated on average about thirty movies. This is a small number of movies compared with other algorithms, but is useful for demonstrating the effectiveness of the algorithm on a small sample.

When a user rates a movie on the Recommendz system, they are required to give a numerical rating (on a scale of 1-10) along with a feature that they thought was important in the movie and the amount of it (on a scale of 1-10). These are used to estimate the movie feature vector. There are approximately 1500 movie features in the database. To reduce the dimensionality, we ran PCA on the movie features.

We compared our Hierarchical Bayesian algorithm to four different algorithms under the context ratings, each of which were tested on all fifteen users. The first algorithm ignores context dependency and predicts user ratings using the weight vector computed by a least squares regression on non-context dependent ratings as described in Section 2.1. This is what would happen if you do not consider contexts at all and share all the information between contexts.

The second algorithm we used involved separating the data completely from one context to another and then performing a linear regression. This would be the same as a user creating a different account for each different context. We expect that this approach will not work well in practice because given the small number of ratings given in each context, the model does not have enough data to learn the parameters. The third algorithm is the k-nearest neighbor algorithm described in Section 2.1. Finally, we ran a “hybrid” algorithm which simply averages the k-nearest neighbor algorithm and the new EM algorithm.

For each of these algorithms, we ran the data twice. In the first run, while performing cross-validation, we left out *all* the ratings from the movie we were leaving out, and not just the one from that context. That is, if a user rated the same movie several times but in different contexts, we left out each rating. This tests whether we can solve the "Standard Context Recommendation" problem. In the second run, we left out only that specific rating, potentially keeping ratings of the same movie in a different context. This tests the algorithm's ability to solve the "Different Context Evaluation" problem.

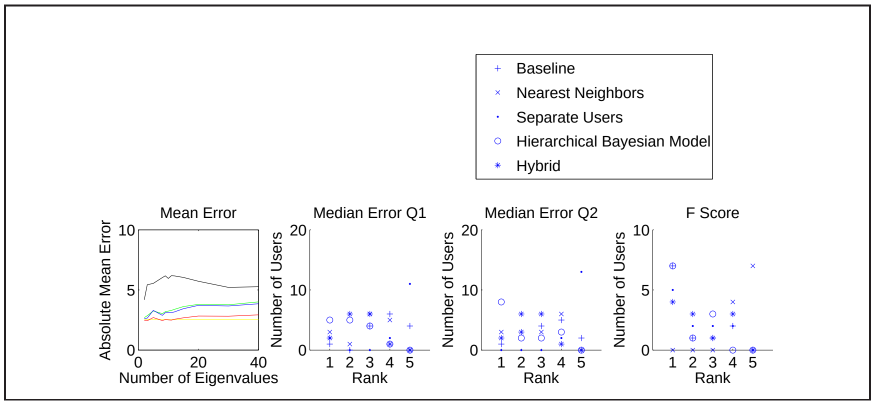

Fig. 2. From left to right: A graph showing the amount of error of the various algorithms as a function of how many eigenvalues were used. Note that each algorithm except for "separate users" (the top line) performs best with approximately eleven eigenvalues, which has very high error regardless. Next, the relative performance of the various algorithms under mean, median, and F-Score. The Hierarchical Bayesian and hybrid models perform best on the most users

5 Experimental Results

We performed leave one out cross-validation on all data in the set. We first determined the ideal number of eigenvalues to use. The mean absolute error is presented in Figure 2. Based on the cross-validation results, we determined the optimal number of eigenvalues to use was eleven because all of the graphs other than the "separate users" graphs have smallest error with approximately eleven eigenvalues. When we use too few eigenvalues, too much data is lost. When we use too many eigenvalues, over-fitting occurs because the training set is small.

We evaluated the mean absolute error (MAE), the median absolute error, and the F-Score of each algorithm. While MAE is an intuitive error measure, F-Score, which depends on both the precision and recall of the algorithm, is considered a more accurate measure of error in recommender systems because the most important criterion for a recommender system is that it recommends the top movies [5]. In evaluating the F-Score, we divided our rankings into percentiles. We compared how successful each algorithm was at putting movies in various quantiles. For the numerical performance of the various algorithms, see Table 1.

Table 1. The results of the recommender when using cross validation. The first set of results are in solving the "Standard Context Recommendation" (SCR) problem. The second set of results are in the "Different Context Evaluation" problem (DCE). For F-Score M, F-Score Q, and F-Score O, we calculate the success of the algorithms in selecting the top 50%, 25%, and 12.5% respectively. Bold items mean optimal relative performance. Note that both the Bayesian model and Hybrid model are original contributions of this paper.

Algorithm (SCR)	**Mean Error**	**Median**	**F-Score M**	**F-Score Q**	**F-Score O**
Linear Regression	2.9378	2.4045	**.676**	.449	.263
Separate Users	6.5767	4.9286	.545	.406	.259
Bayesian Model	2.8411	2.000	.668	**.469**	**.296**
Item Similarity	2.3444	2.1772	.383	.117	.018
Hybrid (Item + Bayes)	**2.2508**	**1.7958**	.591	.389	.182
Algorithm (DCE)	**Mean Error**	**Median**	**F-Score M**	**F-Score Q**	**F-Score O**
Linear Regression	2.0565	1.655	.696	.554	.418
Separate Users	6.5767	4.9286	.545	.406	.259
Bayesian Model	1.9109	**1.1569**	.681	**.557**	.351
Item Similarity	2.0847	1.8709	.643	.479	**.447**
Hybrid (Item + Bayes)	**1.8188**	1.4633	**.708**	.389	.408

When all contexts are left out of the training set for a specific movie, the hybrid algorithm, combining item similarity and the Bayesian model, has the smallest mean and median error. The Bayesian model performs better under this measure than the linear regression and separate users approach, but worse than the item similarity on its own. In measuring the F-Score, the linear regression model is best at determining which elements belong in the top half with the Bayesian model a close second. When considering the algorithms success at putting movies into the top *quarter*, the results are flipped, with the Bayesian model slightly outperforming the linear regression model. In the top *eighth*, the Bayesian model maintains the highest score again, this time with a larger gap.

When we leave only the one rating out, but leave all the others from different contexts in, as expected, the predicted ratings are closer to the actual ratings. This shows that ratings in one context are indeed correlated with ratings in another context. We aim to show that while they are correlated, they are not determinative. In this case the hybrid algorithm has the best results in mean error, and the Bayesian model has smallest error for median error. The hybrid algorithm has the strongest F-score for the median, but the Bayesian algorithm is strongest for the quarter F-score. Interestingly, the item similarity, which has a very low score in giving context ratings in the first run, has the highest score in the F-score when considering a successful match of the top 8th.

We wanted to consider the percentage of users the Bayesian algorithm worked best on. To do this, we broke our results down by user to rank the various algorithms. The data is shown in Figure 2. Along the x-axis is the relative rank of the algorithm (i.e. 1st, 2nd, 3rd, 4th, and 5th). On the y-axis is the number of users for which each algorithm has that rank. The hybrid and hierarchical Bayesian algorithm each have the largest number of users for which they rank first or second. An example of some of the movie recommendations given by the algorithm in different contexts is given in Table 2.

Table 2. Examples of predictions given by the hierarchical Bayesian network in different contexts. Some of the movies are the same, but the list varies.

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

6 Discussion

An important factor in contrasting our results with others is that in our data set, the average user rated approximately 30 movies. Other studies have shown smaller errors using larger training sets (75-100 ratings per user) [13], but that was not the main goal of this work. Unfortunately, it is difficult to collect data as standard data sets such as the Netflix$^{\text{TM}}$set do not keep track of context information and are thus not applicable to our work. Additionally, we wanted to demonstrate the effectiveness of the Hierarchical Bayesian algorithm on a small sample size 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 baseline linear algorithms, the Hierarchical Bayesian model accomplishes what we want: it shares information between contexts without requiring that all information be shared. Even on users for which the Bayesian model does not work best (i.e. those who the nearest neighbor model works well for), we can still improve the nearest neighbor recommendation by averaging it with the Bayesian prediction.

In answering the "Standard Context Recommendation" question, the mean and median error are lowest on the hybrid algorithm. Since the baseline linear algorithm works as well as the Bayesian model at selecting the top 50 % of the movies, but not as well at selecting the top 12 % of the movies, we conclude that the baseline algorithm is good at giving a coarse guess of "good or bad" for a movie in a context, but the Hierarchical Bayesian model is best at distinguishing movies in a finer manner. This makes sense, since it is very often the case that in a different context, a user would still have the same general feeling for a movie (e.g. good vs. bad), but would have variations within this category (e.g. excellent vs. good). In this situation, the F-Score is more appropriate to use as a measure of error than mean or median because this question resembles more closely the traditional question of "What movie should I watch?"

The Bayesian model has the lowest median error in answering the "Different Context Evaluation" problem, showing that it is best able to use ratings from different contexts without automatically assigning the same value. While the results in the F-score are unclear, we consider the mean and median error a more appropriate measure to answering the second question. In the first question, it did not matter how poor movies are ranked as they are not presented to the user anyway. In this case, however, it does matter. The user will ask about a specific movie in a specific context, and we want to give as accurate an answer as possible. Additionally, looking at only the top movies can be very misleading. If a user rates a few movies very highly in all contexts, a very reasonable assumption since many users enjoy their favorite movie in almost every setting, then the item similarity algorithm is almost guaranteed to give the correct answer since the closest item to a movie is always itself. Since we only look at the top 12.5% of movies, many movies fit into this category, causing the item similarity algorithm to have an exceptionally strong score. The other algorithms do not have this benefit because they are parametric. In summary, the very top movies are often the same in various contexts, but after that there is diversity.

In answering both questions, the mean and median error were smallest in the Hierarchical Bayesian model or the hybrid algorithm. The hybrid model performs better than the item similarity algorithm, showing that even if we do not assume a linear model, the Hierarchical Bayesian model can be useful. The Hierarchical Bayesian model performs much better than other linear models. While the raw error is relatively high, the size of the training set is quite small and the results show that the Hierarchical Bayesian model (or in some cases a hybrid form of it) is better than baseline algorithms in making context-dependent recommendations. The Hierarchical Bayesian model is strongest at recommending the top movies when they are previously unrated. The Bayesian model is best at predicting a score of a movie when given a previous rating in a different context.

7 Conclusions

We designed a Hierarchical Bayesian model [13] to learn different weights in different contexts. This algorithm answers 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 compared our algorithm to several other techniques, some of which share *all* information between contexts and some of which share *none* of the information between contexts. We found that our algorithm or a hybrid algorithm performed at least as well in most forms of measurement. Because the approach is content-based, the algorithm does not assume that the preferences of every user change in the same way depending on the context. This allows users to have personal interpretations of contexts or even to add their own new contexts. This work demonstrates that it is useful to store contextual information. Naive approaches do not incorporate contexts as effectively as the Hierarchical Bayesian model. A potential area to explore is creating a hybrid of the content-based approach discussed here with a context-dependent collaborative filtering approach.

References

1. Adomavicius, G., Sankaranarayanan, R., Sen, S., and Tuzhilin, A. 2005. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.* 23, 1 (Jan. 2005), 103-145.
2. Alspector, J., Kolcz, A. and Karunanithi, N.: Comparing Feature-based and Clique-based User Models for Movie Selection. In *Proc. of the 3rd ACM Conf. on Digital Libraries,* Pittsburgh, PA, pp. 11-18 (1998).
3. Garden, M. and Dudek, G. 2005. Semantic Feedback for Hybrid Recommendations in Recommendz. In *Proc. of the 2005 IEEE International Conf. on E-Technology, E-Commerce and E-Service (Eee'05).* Hong Kong, China (2005).
4. Goldberg, K., Roeder, T., Gupta, D., and Perkins, C.: Eigentaste: A Constant Time Collaborative Filtering Algorithm. In *Information Retrieval Journal* 4 (2001)(2), pp. 131-151 (2001).
5. Herlocker, J., Konstan, J., Terveen, L., and Riedl, J: Evaluating Collaborative Filtering Recommender Systems. In *ACM Transactions on Information Systems* v. 22, pp. 5-53 (2004).
6. Ono, C., Kurokawa, M., Motomura, Y., and Asoh, H.: A Context-Aware Movie Preference Model Using a Bayesian Network for Recommendation and Promotion. In *User Modelling*, pp 247-257 (2007).
7. Salton, G. :Automatic Text Processing, Addison-Wesley, (1989).
8. Sarwar, B.M., Karypis, G., Konstan, J.A., and Riedl, J :Item-base Collaborative Filtering Recommendation Algorithms. In *Proc. of the 10th International World Wide Web Conf.* (WWW10)(2001).
9. Vinay, V., Cox, I., Wood, K., and Milic-Frayling, N. : A Comparison of Dimensionality Reduction Techniques for Text Retrieval. In ICMLA, pp 293-298 (2005).
10. Yang, Y.: An Evaluation of Statistical Approaches to Text Categorization. Information Retrieval 1(1) 67-88 (1999)
11. Yu, K., Tresp, V., and Schwaighofer, A: Learning Gaussian Processes from Multiple Tasks. In *ICML '05:Proc. of the 22nd international Conf. on Machine Learning,* pp. 1012-1019, New York, NY, USA, (2005).
12. Zhang, T., Iyengar, V.S., and Kaelbling, P : Recommender Systems Using Linear Classifiers. In *Journal of Machine Learning Research.* v. 2. pp. 313-334 (2002).
13. Zhang, Y. and Koren, J. : Efficient Bayesian Hiearchical User Modeling for Recommendation Systems. In *Proc. of the 30th Annual International ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR '07)*, New York, NY, USA, (2007).