Proceedings of the Eighteenth National Conference on Artificial Intelligence(AAAI-2002),
pp. 187-192, Edmonton, Canada, July 2002

Content-Boosted Collaborative Filtering for Improved Recommendations

Prem Melville and **Raymond J. Mooney** and **Ramadass Nagarajan**
Department of Computer Sciences
University of Texas
Austin, TX 78712
{melville,mooney,ramdas}@cs.utexas.edu

Abstract

Most recommender systems use Collaborative Filtering or Content-based methods to predict new items of interest for a user. While both methods have their own advantages, individually they fail to provide good recommendations in many situations. Incorporating components from both methods, a hybrid recommender system can overcome these shortcomings. In this paper, we present an elegant and effective framework for combining content and collaboration. Our approach uses a content-based predictor to enhance existing user data, and then provides personalized suggestions through collaborative filtering. We present experimental results that show how this approach, *Content-Boosted Collaborative Filtering*, performs better than a pure content-based predictor, pure collaborative filter, and a naive hybrid approach.

Introduction

Recommender systems help overcome information overload by providing personalized suggestions based on a history of a user's likes and dislikes. Many on-line stores provide recommending services e.g. Amazon, CDNOW, BarnesAndNoble, IMDb, etc. There are two prevalent approaches to building recommender systems — Collaborative Filtering (CF) and Content-based (CB) recommending. CF systems work by collecting user feedback in the form of ratings for items in a given domain and exploit similarities and differences among profiles of several users in determining how to recommend an item. On the other hand, content-based methods provide recommendations by comparing representations of content contained in an item to representations of content that interests the user.

Content-based methods can uniquely characterize each user, but CF still has some key advantages over them (Herlocker *et al.* 1999). Firstly, CF can perform in domains where there is not much content associated with items, or where the content is difficult for a computer to analyze — ideas, opinions etc. Secondly a CF system has the ability to provide serendipitous recommendations, i.e. it can recommend items that are relevant to the user, but do not contain content from the user's profile. Because of these reasons, CF systems have been used fairly successfully to build recommender systems in various domains (Goldberg *et al.* 1992; Resnick *et al.* 1994). However they suffer from two fundamental problems:

- *Sparsity*
  Stated simply, most users do not rate most items and hence the user-item rating matrix is typically very sparse. Therefore the probability of finding a set of users with $significantly$ similar ratings is usually low. This is often the case when systems have a very high item-to-user ratio. This problem is also very significant when the system is in the initial stage of use.
- *First-rater Problem*
  An item cannot be recommended unless a user has rated it before. This problem applies to new items and also obscure items and is particularly detrimental to users with eclectic tastes.

We overcome these drawbacks of CF systems by exploiting content information of the items already rated. Our basic approach uses content-based predictions to convert a sparse user ratings matrix into a full ratings matrix; and then uses CF to provide recommendations. In this paper, we present the framework for this new hybrid approach, Content-Boosted Collaborative Filtering (CBCF). We apply this framework in the domain of movie recommendation and show that our approach performs better than both pure CF and pure content-based systems.

Domain Description

We demonstrate the working of our hybrid approach in the domain of movie recommendation. We use the user-movie ratings from the EachMovie[1] dataset, provided by the Compaq Systems Research Center. The dataset contains rating data provided by each user for various movies. User ratings range from zero to five stars. Zero stars indicate extreme dislike for a movie and five stars indicate high praise. To have a quicker turn-around time for our experiments, we only used a subset of the EachMovie dataset. This dataset contains 7,893 randomly selected users and 1,461 movies for which content was available from the Internet Movie Database (IMDb)[2]. The reduced dataset has 299,997 ratings for 1,408 movies. The average number of votes per user is

[1] http://research.compaq.com/SRC/eachmovie
[2] http://www.imdb.com

approximately 38 and the sparsity of the user ratings matrix is 97.4%.

The content information for each movie was collected from IMDb using a simple crawler. The crawler follows the IMDB link provided for every movie in the EachMovie dataset and collects information from the various links off the main URL. We represent the content information of every movie as a set of slots (features). Each slot is represented simply as a bag of words. The slots we use for the EachMovie dataset are: movie title, director, cast, genre, plot summary, plot keywords, user comments, external reviews, newsgroup reviews, and awards.

System Description

The general overview of our system is shown in Figure 1. The web crawler uses the URLs provided in the EachMovie dataset to download movie content from IMDb. After appropriate preprocessing, the downloaded content is stored in the Movie Content Database. The EachMovie dataset also provides the user-ratings matrix, which is a matrix of users versus items, where each cell is the rating given by a user to an item. We will refer to each row of this matrix as a *user-ratings vector*. The user-ratings matrix is very sparse, since most items have not been rated by most users. The content-based predictor is trained on each user-ratings vector and a pseudo user-ratings vector is created. A pseudo user-ratings vector contains the user's actual ratings and content-based predictions for the unrated items. All pseudo user-ratings vectors put together form the pseudo ratings matrix, which is a full matrix. Now given an active user's[3] ratings, predictions are made for a new item using CF on the full pseudo ratings matrix.

The following sections describe our implementation of the content-based predictor and the pure CF component; followed by the details of our hybrid approach.

Pure Content-based Predictor

To provide content-based predictions we treat the prediction task as a text-categorization problem. We view movie content information as text documents, and user ratings 0-5 as one of six class labels. We implemented a bag-of-words naive Bayesian text classifier (Mitchell 1997) extended to handle a vector of bags of words; where each bag-of-words corresponds to a movie-feature (e.g. title, cast, etc.). We use the classifier to learn a user profile from a set of rated movies i.e. labeled documents. The learned profile is then used to predict the label (rating) of unrated movies. A similar approach to recommending has been used effectively in the book-recommending system LIBRA (Mooney & Roy 2000).

Pure Collaborative Filtering

We implemented a pure collaborative filtering component that uses a *neighborhood-based algorithm* (Herlocker *et al.* 1999). In neighborhood-based algorithms, a subset of users are chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for the active user. The algorithm we use can be summarized in the following steps:

1. Weight all users with respect to similarity with the active user.
   - Similarity between users is measured as the Pearson correlation between their ratings vectors.
2. Select n users that have the highest similarity with the active user.
   - These users form the *neighborhood*.
3. Compute a prediction from a weighted combination of the selected neighbors' ratings.

In step 1, similarity between two users is computed using the Pearson correlation coefficient, defined below:

$$P_{a,u} = \frac{\sum_{i=1}^{m}(r_{a,i}-\overline{r}_a)\times(r_{u,i}-\overline{r}_u)}{\sqrt{\sum_{i=1}^{m}(r_{a,i}-\overline{r}_a)^2\times\sum_{i=1}^{m}(r_{u,i}-\overline{r}_u)^2}} \tag{1}$$

where $r_{a,i}$ is the rating given to item i by user a; $\overline{r}_a$ is the mean rating given by user a; and m is the total number of items.

In step 3, predictions are computed as the weighted average of deviations from the neighbor's mean:

$$p_{a,i} = \overline{r}_a + \frac{\sum_{u=1}^{n}(r_{u,i}-\overline{r}_u)\times P_{a,u}}{\sum_{u=1}^{n}P_{a,u}} \tag{2}$$

where $p_{a,i}$ is the prediction for the active user a for item i; $P_{a,u}$ is the similarity between users a and u; and n is the number of users in the neighborhood. For our experiments we used a neighborhood size of 30, based on the recommendation of (Herlocker *et al.* 1999).

It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping)

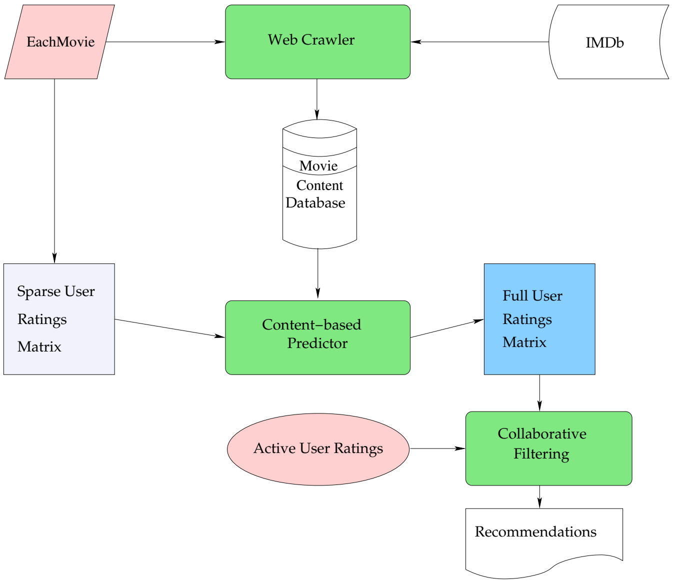

Figure 1: System Overview

[3] The active user is the user for whom predictions are being made.

items. These neighbors based on a small number of overlapping items tend to be bad predictors. To devalue the correlations based on few co-rated items, we multiply the correlation by a *Significance Weighting* factor (Herlocker *et al.* 1999). If two users have less than 50 co-rated items we multiply their correlation by a factor $sg_{a,u} = n/50$, where n is the number of co-rated items. If the number of overlapping items is greater than 50, then we leave the correlation unchanged i.e. $sg_{a,u} = 1$.

Content-Boosted Collaborative Filtering

In content-boosted collaborative filtering, we first create a *pseudo user-ratings vector* for every user u in the database. The pseudo user-ratings vector, v_u, consists of the item ratings provided by the user u, where available, and those predicted by the content-based predictor otherwise.

$$v_{u,i} = \begin{cases} r_{u,i} : \text{ if user } u \text{ rated item } i \\ c_{u,i} : \text{otherwise} \end{cases}$$

In the above equation $r_{u,i}$ denotes the actual rating provided by user u for item i, while $c_{u,i}$ is the rating predicted by the pure content-based system.

The pseudo user-ratings vectors of all users put together give the dense pseudo ratings matrix V. We now perform collaborative filtering using this dense matrix. The similarity between the active user a and another user u is computed using the Pearson correlation coefficient described in Equation 1. Instead of the original user votes, we substitute the votes provided by the pseudo user-ratings vectors v_a and v_u.

Harmonic Mean Weighting The accuracy of a pseudo user-ratings vector computed for a user depends on the number of movies he/she has rated. If the user rated many items, the content-based predictions are good and hence his pseudo user-ratings vector is fairly accurate. On the other hand, if the user rated only a few items, the pseudo user-ratings vector will not be as accurate. We found that inaccuracies in pseudo user-ratings vector often yielded misleadingly high correlations between the active user and other users. Hence to incorporate confidence (or the lack thereof) in our correlations, we weight them using the *Harmonic Mean weighting* factor (*HM weighting*).

$$\begin{aligned} hm_{i,j} &= \frac{2m_i m_j}{m_i + m_j} \\ m_i &= \begin{cases} \frac{n_i}{50} : \text{if } n_i < 50 \\ 1 : \text{otherwise} \end{cases} \end{aligned}$$

In the above equation, n_i refers to the number of items that user i has rated. The harmonic mean tends to bias the weight towards the lower of the two values — m_i and m_j. Thus correlations between pseudo user-ratings with at least 50 user-rated items each, will receive the highest weight, regardless of the actual number of movies each user rated. On the other hand, even if one of the pseudo user-rating vectors is based on less than 50 user-rated items, the correlation will be devalued appropriately.

The choice of the threshold 50 is based on the performance of the content-based predictor, which was evaluated using 10-fold cross-validation (Mitchell 1997). To test performance on varying amounts of training data, a learning curve was generated by testing the system after training on increasing subsets of the overall training data. We generated learning curves for 132 users who had rated more than 200 items. The points on the 132 curves were averaged to give the final learning curve. From the learning curve we noted that as the predictor is given more and more training examples the prediction performance improves, but at around 50 it begins to level off. Beyond this is the point of diminishing returns; as no matter how large the training set is, prediction accuracy improves only marginally.

To the HM weight, we add the significance weighting factor described earlier, and thus obtain the *hybrid correlation weight* $hw_{a,u}$.

$$hw_{a,u} = hm_{a,u} + sg_{a,u} \tag{3}$$

Self Weighting Recall that in CF, a prediction for the active user is computed as a weighted sum of the mean-centered votes of the best-n neighbors of that user. In our approach, we also add the pseudo active user[4] to the neighborhood. However, we may want to give the pseudo active user more importance than the other neighbors. In other words, we would like to increase the confidence we place in the pure-content predictions for the active user. We do this by incorporating a *Self Weighting* factor in the final prediction:

$$sw_a = \begin{cases} \frac{n_a}{50} \times max : \text{if } n_a < 50 \\ max \quad\quad : \text{otherwise} \end{cases} \tag{4}$$

where n_a is the number of items rated by the active user. Again, the choice of the threshold 50 is motivated by the learning curve mentioned earlier. The parameter max is an indication of the over-all confidence we have in the content-based predictor. In our experiments, we used a value of 2 for max.

Producing Predictions Combining the above two weighting schemes, the final CBCF prediction for the active user a and item i is produced as follows:

$$p_{a,i} = \overline{v}_a + \frac{sw_a(c_{a,i} - \overline{v}_a) + \sum_{\substack{u=1 \\ u \neq a}}^{n} hw_{a,u} P_{a,u} (v_{u,i} - \overline{v}_u)}{sw_a + \sum_{\substack{u=1 \\ u \neq a}}^{n} hw_{a,u} P_{a,u}}$$

In the above equation $c_{a,i}$ corresponds to the pure-content predictions for the active user and item i; $v_{u,i}$ is the pseudo user-rating for a user u and item i; $\overline{v}_u$ is the mean over all items for that user; sw_a, $hw_{a,u}$ and $P_{a,u}$ are as shown in Equations 4, 3 and 1 respectively; and n is the size of neighborhood. The denominator is a normalization factor that ensures all weights sum to one.

Experimental Evaluation

In this section we describe the experimental methodology and metrics we use to compare different prediction algorithms; and present the results of our experiments.

[4]Pseudo active user refers to the pseudo user-ratings vector based on the active user's ratings.

Methodology

We compare CBCF to a pure content-based predictor, a CF predictor, and a naive hybrid approach. The naive hybrid approach takes the average of the ratings generated by the pure content-based predictor and the pure CF predictor. For the purposes of comparison, we used a subset of the ratings data from the $EachMovie$ data set (described earlier). Ten percent of the users were randomly selected to be the test users. From each user in the test set, ratings for 25% of items were withheld. Predictions were computed for the withheld items using each of the different predictors.

The quality of the various prediction algorithms were measured by comparing the predicted values for the withheld ratings to the actual ratings.

Metrics

The metrics for evaluating the accuracy of a prediction algorithm can be divided into two main categories: *statistical accuracy metrics* and *decision-support metrics* (Herlocker *et al.* 1999). Statistical accuracy metrics evaluate the accuracy of a predictor by comparing predicted values with user-provided values. To measure statistical accuracy we use the mean absolute error (MAE) metric — defined as the average absolute difference between predicted ratings and actual ratings. In our experiments we computed the MAE on the test set for each user, and then averaged over the set of test users.

Decision-support accuracy measures how well predictions help users select *high-quality* items. We use Receiver Operating Characteristic (ROC) sensitivity to measure decision-support accuracy. A predictor can be treated as a filter, where predicting a high rating for an item is equivalent to accepting the item, and predicting a low rating is equivalent to rejecting the item. The ROC sensitivity is given by the area under the ROC curve — a curve that plots *sensitivity* versus 1-*specificity* for a predictor. Sensitivity is defined as the probability that a good item is accepted by the filter; and specificity is defined as the probability that a bad item is rejected by the filter. We consider an item *good* if the user gave it a rating of 4 or above, otherwise we consider the item *bad*. We refer to this ROC sensitivity with threshold 4 as ROC-4. ROC sensitivity ranges from 0 to 1, where 1 is ideal and 0.5 is random.

The statistical significance of any differences in performance between two predictors was evaluated using two-tailed paired *t*-tests (Mitchell 1997).

Results

Algorithm	MAE	ROC-4
Pure content-based (CB) predictor	1.059	0.6376
Pure CF	1.002	0.6423
Naive Hybrid	1.011	0.6121
Content-boosted CF	**0.962**	**0.6717**

Table 1: Summary of Results

The results of our experiments are summarized in Table 1. As can be seen, our CBCF approach performs better than the other algorithms on both metrics. On the MAE metric, CBCF performs 9.2% better than pure CB, 4% better than pure CF and 4.9% better than the naive hybrid. All the differences in MAE are statistically significant ($p < 0.001$).

On the ROC-4, metric CBCF performs 5.4% better than pure CB, 4.6% better than pure CF and 9.7% better than the naive hybrid. This implies that our system, compared to others, does a better of job of recommending high-quality items, while reducing the probability of recommending bad items to the user.

Interestingly, *Self Weighting* did not make significant improvements to our predictions. We believe that *Self Weighting* would play a more important role if the pure CB predictor significantly outperformed CF.

Discussion

In this section we explain how content-boosted collaborative filtering overcomes some of the shortcomings of pure CF; and we also discuss some ways of improving CBCF.

Overcoming the First-Rater Problem

In pure CF a prediction cannot be made for an item, for the active user, unless it was previously rated by other users. However, we can make such a prediction using a content-based predictor for the user. Using CBCF we can further improve the CB predictions by utilizing the content-based predictions of *other* users as well. If the neighbors of the active user are highly correlated to it, then their CB predictions should also be very relevant to the user. This is particularly true if neighbors have rated many more items than the active user; because their CB predictions are likely to be more accurate than the active user's. To verify this hypothesis we ran the following experiments. A set of 500 users were selected at random from the $EachMovie$ data set. We randomly selected an item to be deleted from the user-ratings matrix. We then produced pure content-based and CBCF predictions for the users that had rated the selected item. We repeated this process for 55 items and averaged the MAE over all users and items. We found that the MAEs for the pure CB predictor and CBCF were 1.060 and 1.023 respectively; and that the difference is statistically significant ($p < 0.001$). So we can conclude that using collaborative information is beneficial even if no other user has rated the item in question. In this way, CBCF solves the first-rater problem, and produces even better predictions than the content-based predictor.

Tackling Sparsity

In CBCF, since we use a pseudo ratings matrix, which is a full matrix, we eliminate the root of the sparsity problem. Pseudo user-ratings vectors contain ratings for all items; and hence all users will be considered as potential neighbors. This increases the chances of finding similar users. Thus the sparsity of the user-ratings matrix affects CBCF to a smaller degree than CF. To verify this hypothesis we ran the following experiments. A set of 1000 users were selected at random from the $EachMovie$ data set. We treated 500 of these users as test users, and produced predictions on 25% of the items they rated. We artificially increased the sparsity

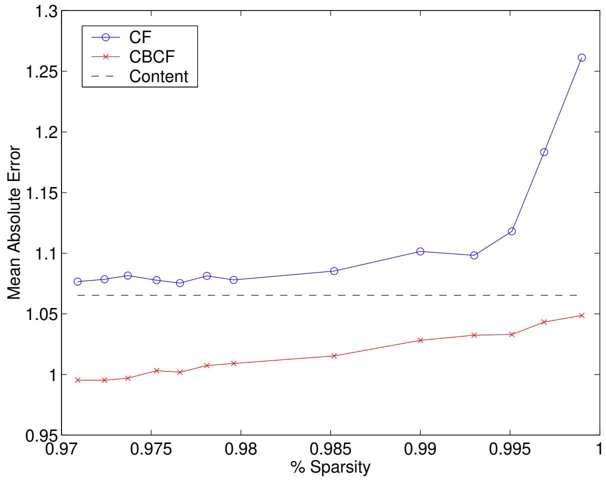

Figure 2: Effect of Sparsity on Prediction Accuracy

of the user-ratings matrix, by randomly dropping elements from the matrix. We compared the MAE of the different predictors at varying levels of sparsity. Figure 2 confirms our hypothesis that CBCF is more stable than CF with respect to sparsity. In fact, when sparsity exceeds 99%, the performance of CF drops precipitously, while CBCF is relatively unaffected. In the limit, CBCF will converge to pure content-based predictions.

Finding Better Neighbors

A crucial step in CF is the selection of a neighborhood. The neighbors of the active user entirely determine his predictions. It is therefore critical to select neighbors who are most similar to the active user. In pure CF, the neighborhood comprises of the users that have the best n correlations with the active user. The similarity between users is only determined by the ratings given to co-rated items; so items that have not been rated by both users are ignored. However, in CBCF, the similarity is based on the ratings contained in the pseudo user-ratings vectors; so users do not need to have a high overlap of co-rated items to be considered similar. Our claim is that this feature of CBCF, makes it possible to select a better, more representative neighborhood. For example, consider two users with identical tastes who have not rated any items in common. Pure collaborative filtering would not consider them similar. However, pseudo user-ratings vectors created using content-based predictions for the two users would be highly correlated, and therefore they would be considered neighbors. We believe that this superior selection of neighbors is one of the reasons that CBCF outperforms pure CF.

Improving CBCF

Due to the nature of our hybrid approach, we believe that improving the performance of the individual components would almost certainly improve the performance of the whole system. In other words, if we improved our pure content-based predictor or the CF algorithm, we would be able to improve our system's predictions. A better content-based predictor would mean that the pseudo ratings matrix generated would more accurately approximate the *actual* full user-ratings matrix. This in turn, would improve the chances of finding more representative neighbors. And since the final predictions in our system are based on a CF algorithm, a better CF algorithm can only improve our system's performance.

In our current implementation of the content-based predictor, we use a naive Bayesian text-classifier to learn a six-way classification task. This approach is probably not ideal, since it disregards the fact that classes represent ratings on a linear scale. This problem can be overcome by using a learning algorithm that can directly produce numerical predictions. For example, logistic regression and locally weighted regression (Duda, Hart, & Stork 2000) could be used to directly predict ratings from item content. We should be able to improve our content-based predictions using one of these approaches. In addition, the CF component in our system may be improved by using a *Clustered Pearson Predictor* (CPP) (Fisher *et al.* 2000). The CPP algorithm creates clusters of users based on k-means clustering. Collaborative predictions are made by only using the cluster centroids as potential neighbors. Fisher et al. claim that this approach is more accurate than the pure CF algorithm, with the added advantage of being more scalable.

Related Work

There have been a few other attempts to combine content information with collaborative filtering. One simple approach is to allow both content-based and collaborative filtering methods to produce separate recommendations, and then to directly combine their predictions (Cotter & Smyth 2000; Claypool, Gokhale, & Miranda 1999). In another approach (Soboroff & Nicholas 1999), the *term-document matrix* is multiplied with the user-ratings matrix to produce a *content-profile matrix*. Using Latent Semantic Indexing, a rank-k approximation of the content-profile matrix is computed. Term vectors of the user's relevant documents are averaged to produce a user's profile. Now, new documents are ranked against each user's profile in the LSI space. In Pazzani's approach (1999), each user-profile is represented by a vector of weighted words derived from positive training examples using the Winnow algorithm. Predictions are made by applying CF directly to the matrix of user-profiles (as opposed to the user-ratings matrix). An alternate approach, Fab (Balabanovic & Shoham 1997), uses relevance feedback to simultaneously mold a personal filter along with a communal "topic" filter. Documents are initially ranked by the topic filter and then sent to a user's personal filter. The user's relevance feedback is used to modify both the personal filter and the originating topic filter. In another approach, Basu et al. (1998) treat recommending as a classification task. They use *Ripper*, a rule induction system, to learn a function that takes a user and movie and predicts whether the movie will be liked or disliked. They combine collaborative and content information, by creating features such as *comedies liked by user* and *users who liked movies of genre X*. Good et al. (1999) use collaborative filtering along with a number of personalized information filtering agents. Predictions for a user are made by applying CF on the set of other users and the active user's personalized agents. Our method differs from this by also using CF on the personalized agents of the other users. In recent work, Lee (2001) treats the recommending task as the learning of a user's preference function that exploits item content as well as the ratings of similar users. They perform a study of several mixture models for this task. Popescul et al. (2001) extended Hofmann's aspect model to incorporate three-way co-occurrence data among users, items, and item content. They propose a method of dealing with sparsity that is similar to ours. They estimate the probability of a user accessing a document, that he has not seen before, by the average cosine similarity of the document to all the documents the user has seen. Our task differs from their's since we provide numerical ratings instead of just rankings. Also their approach is tied to the EM framework; whereas our approach is more modular and general, and as such it is independent of the choice of collaborative and content-based components.

Conclusions and Future Work

Incorporating content information into collaborative filtering can significantly improve predictions of a recommender system. In this paper, we have provided an effective way of achieving this. We have shown how Content-boosted Collaborative Filtering performs better than a pure content-based predictor, collaborative filtering, and a naive hybrid of the two.

CBCF elegantly exploits content within a collaborative framework. It overcomes the disadvantages of both collaborative filtering and content-based methods, by bolstering CF with content and vice versa. Further, due to the modular nature of our framework, any improvements in collaborative filtering or content-based recommending can be easily exploited to build a more powerful system.

Although CBCF performs consistently better than pure CF, the difference in performance is not very large (4%). The performance of our system can be boosted by using the methods described earlier. Experiments comparing the different approaches of combining content and collaboration, outlined in the previous section, are also needed.

Acknowledgments

We would like to thank Vishal Mishra for his web crawler and many useful discussions. We also thank Joydeep Ghosh and Inderjit Dhillon for their valuable advice during the course of this work. This research was supported by the National Science Foundation under grants IRI-9704943 and IIS-0117308.

References

Balabanovic, M., and Shoham, Y. 1997. Fab: Content-based, collaborative recommendation. *Communications of the Association of Computing Machinery* 40(3):66–72.

Basu, C.; Hirsh, H.; and Cohen, W. 1998. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 714–720.

Claypool, M.; Gokhale, A.; and Miranda, T. 1999. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of the SIGIR-99 Workshop on Recommender Systems: Algorithms and Evaluation*.

Cotter, P., and Smyth, B. 2000. PTV: Intelligent personalized TV guides. In *Twelfth Conference on Innovative Applications of Artificial Intelligence*, 957–964.

Duda, R. O.; Hart, P. E.; and Stork, D. G. 2000. *Pattern Classification*. New York: Wiley.

Fisher, D.; Hildrum, K.; Hong, J.; Newman, M.; Thomas, M.; and Vuduc, R. 2000. Swami: A framework for collaborative filtering algorithm development and evaluation. In *SIGIR 2000*. Short paper.

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

Good, N.; Schafer, J. B.; Konstan, J. A.; Borchers, A.; Sarwar, B.; Herlocker, J.; and Riedl, J. 1999. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, 439–446.

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

Lee, W. S. 2001. Collaborative learning for recommender systems. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, 314–321.

Mitchell, T. 1997. *Machine Learning*. New York, NY: McGraw-Hill.

Mooney, R. J., and Roy, L. 2000. Content-based book recommending using learning for text categorization. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, 195–204.

Pazzani, M. J. 1999. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review* 13(5-6):393–408.

Popescul, A.; Ungar, L.; Pennock, D. M.; and Lawrence, S. 2001. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the Seventeenth Conference on Uncertainity in Artificial Intelligence*.

Resnick, P.; Iacovou, N.; Sushak, M.; Bergstrom, P.; and Reidl, J. 1994. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 Computer Supported Cooperative Work Conference*. New York: ACM.

Soboroff, I., and Nicholas, C. 1999. Combining content and collaboration in text filtering. In Joachims, T., ed., *Proceedings of the IJCAI'99 Workshop on Machine Learning in Information Filtering*, 86–91.