Appears in *Proceedings of the SIGIR-2001 Workshop on Recommender Systems,*
New Orleans, LA, September 2001

Content-Boosted Collaborative Filtering

Prem Melville, 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. We also discuss methods to improve the performance of our hybrid system.

1. 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 [11]. 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 [9, 19]. 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 significantly better than both pure CF and pure content-based systems.

The remainder of the paper is organized as follows. Section 2 provides an illustrative example to motivate our approach. In Section 3, we describe our domain and the gathering of data. Section 4 describes in detail our implementation of the content-based predictor, the CF algorithm and the hybrid approach. We present our experimental results in Section 5 and explain why our system performs well in Section 6. Section 7 proposes methods to improve our CBCF predictions. In Section 8, we discuss prior attempts at integrating collaboration and content; and finally in Section 9, we conclude with some future extensions to our work.

2. MOTIVATING EXAMPLE

In this section, we describe a common scenario in recommender systems and show why both pure collaborative and content-based methods fail to provide good recommendations. We take the domain of movie recommendations as a representative case.

In most systems, users provide feedback on items that they liked or disliked, using which profiles are formed to learn about the specific interests of each user. For example, in

User A	User B
A Clockwork Orange	Lord of the Rings
Star Wars	Willow
	Blade Runner
	Twelve Monkeys

Table 1: Typical user profiles: movies liked

movie recommendations, typical user profiles could be as shown in Table 1. The table shows two users A and B and their profiles that consists of movies that each liked. Pure CF systems try to find *neighbors*(similar users) for a user by computing similarity measures based on the common set of movies that two users rated. If there is no overlap in the movies of two users, they will not be considered as neighbors. Thus in this example, A and B are not neighbors and potentially movies that B liked, may not be recommended to A even though their profiles suggest that both like science fiction movies.

Pure content-based systems on the other hand, form profiles for each user independently. Thus a typical system would learn that A likes science fiction movies, while B likes both fantasy and science fiction movies. But since each user is considered separately, movies that do not share any content with the ones already rated will not be considered for recommendation. In this case, fantasy movies that B liked may not be recommended to A, even though A and B seem to have a common taste — science fiction and it is quite likely that A will like fantasy movies as well.

Clearly both of the above approaches are inadequate. Let us consider a different approach. We could use a content-based system to predict A's preferences. A content-based predictor would rate *Blade Runner* and *Twelve Monkeys* highly based on A's predilection for science fiction. Now if we were to perform CF using A's content-based predictions, A and B would appear similar; and subsequently B's preferences would be recommended to A. Our CBCF predictor is based on this approach.

3. DOMAIN DESCRIPTION

We demonstrate the working of our hybrid approach in the domain of movie recommendation. We use the user-movie ratings provided by the EachMovie dataset and the movie details from the Internet Movie Database (IMDb) [1, 2]. 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.

3.1 EachMovie Dataset

The EachMovie dataset is provided by the Compaq Systems Research Center, which ran the EachMovie recommendation service for 18 months to experiment with a collaborative filtering algorithm. The information they gathered during that period consists of 72,916 users, 1,628 movies, and 2,811,983 numeric ratings. To have a quicker turnaround 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 IMDb. The reduced dataset has 299,997 ratings for 1,408 movies. The average votes per user is approximately 38 and the sparsity of the user ratings matrix is 2.6%.

The dataset provides optional unaudited demographic data such as age, gender, and the zip code supplied by each person. For each movie, information such as the name, genre, release date and IMDb URL are provided. Finally, the dataset provides the actual 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.

3.2 Data Collection

The content information for each movie was collected from the Internet Movie Database (IMDb). A simple 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 presently download content such as plot summary, plot keywords, cast, user comments, external reviews (newspaper or magazine articles), newsgroup reviews, and awards. This information, after suitable preprocessing such as elimination of stop words etc., is collected into a vector of bag of words, one bag for each feature describing the movie.

4. 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[1] ratings, predictions are made for a new item using CF on the full pseudo ratings matrix.

Sections 4.1 and 4.2 describe our implementation of the content-based predictor and the pure CF component. In Section 4.3 we describe our hybrid approach in detail.

4.1 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 [15] to learn a user profile from

[1]The active user is the user for which predictions are being made.

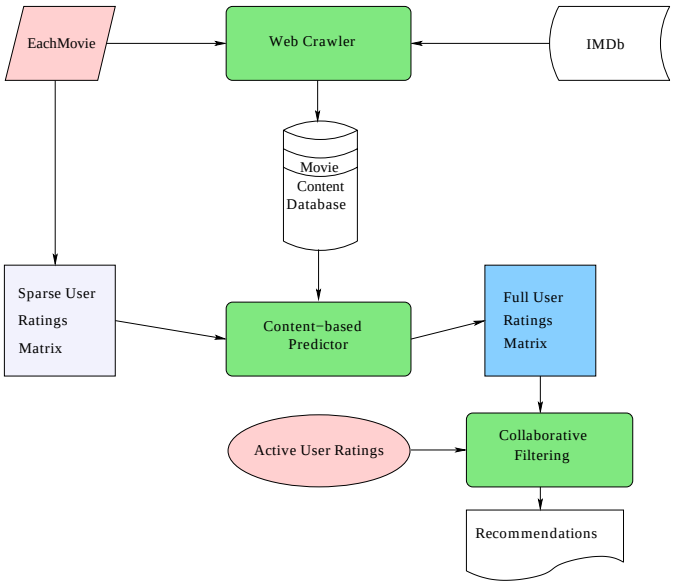

Figure 1: System Overview

a set of rated movies i.e. labeled documents. A similar approach to recommending has been used effectively in the book-recommending system *LIBRA* [16, 17].

We use a multinomial text model [14], in which a document is modeled as an ordered sequence of word events drawn from the same vocabulary, V. The naive Bayes assumption states that the probability of each word event is dependent on the document class but independent of the word's context and position. For each class c_j, and word (token), $w_k \in V$, the probabilities, $P(c_j)$ and $P(w_k|c_j)$ must be estimated from the training data. Then the posterior probability of each class given a document D, is computed using Bayes rule:

$$P(c_j|D) = \frac{P(c_j)}{P(D)} \prod_{i=1}^{|D|} P(a_i|c_j)$$

where a_i is the ith word in the document, and $|D|$ is the number of words in the document. The prior $P(D)$ can be ignored, since it is a constant for any given document.

In our case, since movies are represented as a vector of "documents", d_m, one for each slot (where s_m denotes the mth slot), the probability of each word given the category and the slot, $P(w_k|c_j, s_m)$, must be estimated and the posterior category probabilities for a film, F, computed using:

$$P(c_j|F) = \frac{P(c_j)}{P(F)} \prod_{m=1}^{S} \prod_{i=1}^{|d_m|} P(a_{mi}|c_j, s_m)$$

where S is the number of slots and a_{mi} is the ith word in the mth slot. The class with the highest posterior probability determines the predicted rating.

The model parameters are estimated using the algorithm in Algorithm 1. Note that Laplace smoothing [12] is used to avoid zero probability estimates. The evaluation of the content-based recommender can be found in the appendix.

Algorithm 1 Training the Content-Based Predictor

$Train\_Naive\_Bayes(Examples, C)$

Each example in $Examples$ is a vector of bag-of-words and a category corresponding to a 0-5 rating. Each bag of bag-of-words corresponds to a slot e.g. title, cast, reviews, etc. C is the set of all possible categories. This function estimates the probability terms $P(a_{mi}|c_j, s_m)$, describing the probability that a randomly drawn word from a slot s_m in an example in class c_j will be the word a_{mi}.

1. Calculate class priors, $P(c_j)$
   - $docs_j \leftarrow$ subset of documents from $Examples$ for which the class label is j
   - $P(c_j) \leftarrow \frac{|docs_j| + \frac{1}{|Examples|}}{|Examples| + \frac{|C|}{|Examples|}}$
2. Calculate conditional probabilities, $P(a_{mi}|c_j, s_m)$

   For each slot s_m,
   - $Vocabulary_m \leftarrow$ set of all distinct tokens occurring in slot s_m in all examples
   - For each possible class c_j
     - $Text_{mj} \leftarrow$ a single document created by concatenating all bags-of-words appearing in slot s_m and in class c_j
     - $n \leftarrow$ total number of distinct word positions in $Text_{mj}$
     - For each token, a_{mi} in $Vocabulary_m$
       - $n_k \leftarrow$ number of times token a_{mi} occurs in $Text_{mj}$
       - $P(a_{mi}|c_j, s_m) \leftarrow \frac{n_k + \frac{1}{|Examples|}}{n + \frac{|Vocabulary_m|}{|Examples|}}$

4.2 Pure Collaborative Filtering

We implemented a pure collaborative filtering component that uses a *neighborhood-based algorithm* [11]. 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:

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

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

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

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

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. n is the number of users in the neighborhood. For our experiments we used a neighborhood size of 30, based on the recommendation of [11].

It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping) 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 [11]. 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. $sg_{a,u} = 1$.

4.3 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 gives 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.

4.3.1 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*, for short).

$$hm_{i,j} = \frac{2 m_i m_j}{m_i + m_j}$$

$$m_i = \begin{cases} \frac{n_i}{50} : \text{if } n_i < 50 \\ 1 : \text{otherwise} \end{cases}$$

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 learning curve[2] of the content predictor. As can be seen in Figure 2, initially 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.

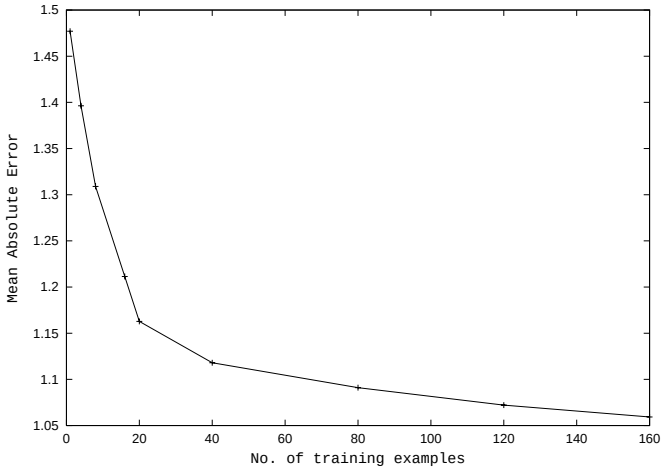

Figure 2: Learning Curve for the Content-based Predictor

To the HM weight, we add the significance weighting described in Section 4.2, and thus obtain the *hybrid correlation weight* $hw_{a,u}$.

$$hw_{a,u} = hm_{a,u} + sg_{a,u} \quad (3)$$

[2]The appendix provides a detailed explanation of the generation of the learning curve.

4.3.2 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[3] 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:

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

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.

4.3.3 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 and $\overline{v}_u$ 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; n is the size of neighborhood. The denominator is a normalization factor that ensures all weights sum to one.

5. 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.

5.1 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 in Section 3.1). Ten percent of the users were randomly selected to be the test users — all test user had rated at least forty movies. 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.

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

5.2 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*. 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.

Herlocker et al. used the same metrics to compare their algorithms [11]. The statistical significance of any differences in performance between two predictors was evaluated using two-tailed paired t-tests [15].

5.3 Results

Algorithm	MAE	ROC-4
Pure content-based 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 2: Summary of Results

The results of our experiments are summarized in Table 2 and Figure 3. 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.

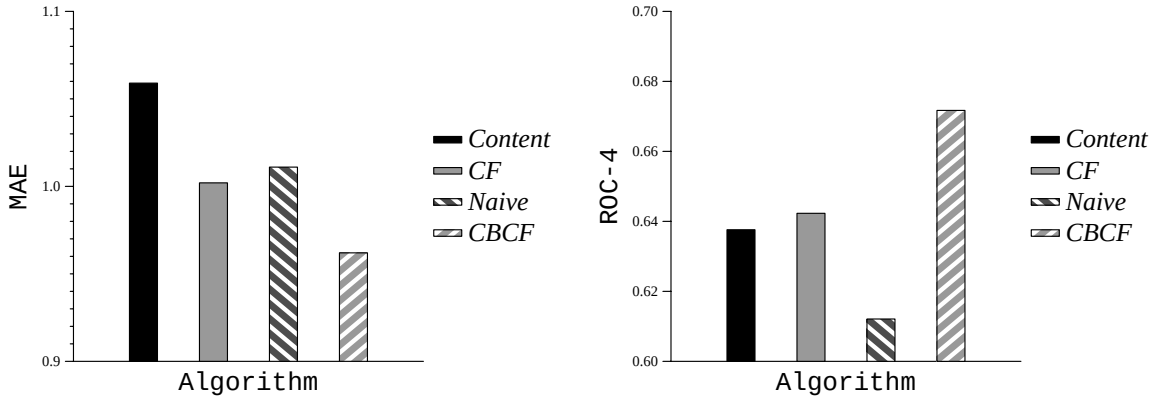

Figure 3: Comparison of algorithms

6. DISCUSSION

In this section we explain how content-boosted collaborative filtering overcomes some of the shortcomings of pure CF; and we also discuss some of our performance results.

6.1 Overcoming Sparsity and the First-Rater Problem

Since we use a pseudo ratings matrix, which is a full matrix, we eliminate the root of the sparsity and first-rater problems. 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.

The original user-ratings matrix may contain items that have not been rated by any user — there are 53 such movies in our dataset. In a pure CF approach these items would be ignored. However in CBCF, these items would receive a content-based prediction from all users. Hence these items can now be recommended to the active user, thus overcoming the first-rater problem.

6.2 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.

6.3 Making Better Predictions

As discussed above, CBCF improves the selection of neighboring users. In traditional CF, we would compute a prediction for each item as a weighted sum of only the *actual* ratings of these neighbors. However, in our approach, if the actual rating from a neighboring user does not exist, we use his content-based predicted rating. This approach is motivated by the hypothesis that if a user is highly correlated to the active user then his content-based predictions are also very relevant to the active user. We believe that the use of the content-based ratings of neighbors to compute predictions is another important factor contributing to CBCF's superior performance.

6.4 Self Weighting

Content predictions based on a large number of training examples tend to be fairly accurate, as is apparent from Figure 2. Hence, giving a greater preference to such predictions should improve the overall accuracy of our hybrid prediction. Interestingly, this was not reflected in our results. This may because of the choice of the max parameter in Equation 4, which was fixed to be 2 in our experiments. A higher value for max would increase the weight of content-based predictions, and might yield better results.

6.5 Naive Hybrid

The naive hybrid approach that we used to compare our system with was inspired by [6]. We found that this approach was a poor strawman to compare with. As can be seen by the results the naive hybrid performs worse than CF on the MAE metric. It also performs poorly on the ROC-4 metric, when compared to the other approaches. In Section 8, we present some other approaches we can use as benchmarks to compare our approach against.

6.6 Efficient Implementation

Outwardly CBCF may appear to be infeasible for an online recommending system, since generating the pseudo ratings matrix requires computing the content-based predictions for all users and all items. However the computational costs of running a CBCF system can be significantly reduced by only making incremental updates to the pseudo ratings matrix. To do this, we need to maintain the most recent pseudo ratings matrix and the models learned by the content-based

predictor for each user. If a user rates new items (or changes existing ratings) then we only need to change that user's column in the pseudo ratings matrix i.e. we retrain the content-based predictor on his new ratings vector and produce predictions for his unrated items. The computational complexity of training and producing predictions with the naive Bayesian classifier is linear in the size of the documents; and therefore a single vector can be updated fairly efficiently. Furthermore, to speed up an online system, we can perform all updates offline in batches at regular intervals.

The pseudo ratings matrix will also need to be updated if a new item is added to the database (e.g. a new movie is released). In this case, a new row with predictions for this item must be added to the ratings matrix. This does not require any retraining, since we maintain the current user models built by the content-based predictor. All we need to do is generate predictions for the new item, for each user. The computational complexity of this operation is linear in the size of the new item (document) times the number of users. Therefore this update is also taken care of efficiently.

7. 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. We discuss some methods we could use to improve the individual components.

7.1 Improving the Content-based Predictor

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. For example, the posterior probabilities for the classes 1 and 3 might be 0.4 and 0.6 respectively, this would imply that a good prediction should be close to 2. But the classifier will predict a 3 i.e the class with the higher posterior probability.

This problem can be overcome by using a learning algorithm that can directly produce numerical predictions. For example, logistic regression and locally weighted regression [7] 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.

7.2 Improving the CF Component

The CF component in our system can be improved by using a *Clustered Pearson Predictor* (CPP) [8], instead of the Simple Pearson Predictor (SPP) that we currently employ. The CPP algorithm creates k clusters of users based on the k-means clustering algorithm. A profile is created for each cluster, which contains the average of the ratings given for each item by all the users in the cluster. Now, predictions are computed using SPP where only the k profiles generated earlier are considered as potential neighbors. Fisher et al. claim that this approach is more accurate than SPP [8]. CPP also has the advantage of being more scalable than SPP.

8. 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 ranked lists of recommendations, and then merge their results to produce a final list [6]. There can be several schemes to merging the ranked lists, such as interleaving content and collaborative recommendations or averaging the rank or rating predicted by the two methods. This is essentially what our naive hybrid approach does.

Soboroff et al. propose a novel approach to combining content and collaboration using latent semantic indexing (LSI) [20]. In their approach, first a *term-document matrix* is created, where each cell is a weight related to the frequency of occurrence of a term in a document. The term-document matrix is multiplied by the normalized ratings matrix to give a *content-profile matrix*. The singular value decomposition (SVD) of this matrix is computed. Using LSI, a rank-k approximation of the content-profile matrix is computed. Term vectors of the user's relevant documents are averaged to produce a centroid representing the user's profile. Now, new documents are ranked against each user's profile in the LSI space.

In Pazzani's approach [18], user profiles are represented by a set of weighted words derived from positive training examples using the Winnow algorithm. This collection of user profiles can be thought of as the *content-profile matrix*. Predictions are made by applying CF directly to the *content-profile matrix* (as opposed to the user-ratings matrix).

An alternate approach to providing content-based collaborative recommendations is used in Fab [3]. Fab 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 user's personal filters. A user then provides relevance feedback for that document, which is used to modify both the personal filter and the originating topic filter.

Basu et al. integrate content and collaboration in a framework in which they treat recommending as a classification task [4]. They use *Ripper*, an inductive logic program, to learn a function that takes a user and movie and predicts a label indicating 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. [10] use collaborative filtering along with a number of personalized information filtering agents. Predictions for a user were 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 [13] 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.

In related work, Billsus and Pazzani [5] use singular value decomposition to directly tackle the *sparsity* problem. They use the SVD of the original user-ratings matrix to project user-ratings and rated items into a lower dimensional space. By doing this they eliminate the need for users to have co-rated items in order to be predictors for each other.

9. 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 significantly better than a pure content-based predictor, collaborative filtering, or 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 nature of the approach, 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%). We are currently attempting to boost the performance of our system by using the methods described in Section 7. In future, we also plan to test if our approach performs better than the other approaches that combine content and collaboration outlined in Section 8.

Acknowledgments

We would like to thank the Compaq Computer Corporation for generously providing the EachMovie dataset used in this paper. We are grateful to 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 grant IRI-9704943.

10. REFERENCES

[1] *EachMovie dataset.* http://research.compaq.com/SRC/eachmovie.

[2] *Internet Movie Database.* http://www.imdb.com.

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

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

[5] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*, pages 46–54, Madison, WI, 1998. Morgan Kaufmann.

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

[7] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley, New York, November 2000.

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

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

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

[11] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. 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*, pages 230–237, 1999.

[12] R. Kohavi, B. Becker, and D. Sommerfield. Improving simple Bayes. In *Proceedings of the European Conference on Machine Learning*, 1997.

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

[14] A. K. McCallum and K. Nigam. A comparison of event models for naive Bayes text classification. In *Papers from the AAAI 1998 Workshop on Text Categorization*, pages 41–48, Madison, WI, 1998.

[15] T. Mitchell. *Machine Learning.* McGraw-Hill, New York, NY, 1997.

[16] R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the SIGIR-99 Workshop on Recommender Systems: Algorithms and Evaluation*, Bekeley, CA, 1999.

[17] R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *"Proceedings of the Fifth ACM Conference on Digital Libraries"*, pages 195–204, San Antonio, TX, June 2000.

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

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

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

APPENDIX

The performance of the content-based predictor was evaluated using 10-fold cross-validation, in which each data set is randomly split into 10 equal-size segments and results are averaged over 10 trials. For each trial, one segment is set aside for testing, while the remaining data is available for training. 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 learning curves were averaged to give the learning curve in Figure 2.