Recommendation as Classification: Using Social and Content-Based Information in Recommendation

Chumki Basu*
Bell Communications Research
445 South Street
Morristown, NJ 07960-6438
cbasu@bellcore.com

Haym Hirsh
Department of Computer Science
Rutgers University
Piscataway, NJ 08855
hirsh@cs.rutgers.edu

William Cohen
AT&T Laboratories
180 Park Ave, Room A207
Florham Park, NJ 07932
wcohen@research.att.com

Abstract

Recommendation systems make suggestions about artifacts to a user. For instance, they may predict whether a user would be interested in seeing a particular movie. Social recomendation methods collect ratings of artifacts from many individuals, and use nearest-neighbor techniques to make recommendations to a user concerning new artifacts. However, these methods do not use the significant amount of other information that is often available about the nature of each artifact — such as cast lists or movie reviews, for example. This paper presents an inductive learning approach to recommendation that is able to use both ratings information and other forms of information about each artifact in predicting user preferences. We show that our method outperforms an existing social-filtering method in the domain of movie recommendations on a dataset of more than 45,000 movie ratings collected from a community of over 250 users.

Introduction

Recommendations are a part of everyday life. We usually rely on some external knowledge to make informed decisions about an artifact of interest or a course of action, for instance when we are going to see a movie or going to see a doctor. This knowledge can be derived from social processes. When we are buying a CD, we can rely on the judgment of a person who shares similar tastes in music. At other times, our judgments may be based on available information about the artifact itself and our known preferences. There are many factors which may influence a person in making these choices, and ideally one would like to model as many of these factors as possible in a recommendation system.

There are some general approaches to this problem. In one approach, the user of the system provides ratings of some artifacts or items and the system makes informed guesses about what other items the user may like. It bases these decisions on the ratings other users have provided. This is the framework for *social-filtering* methods (Hill, Stead, Rosenstein & Furnas 1995; Shardanand & Maes 1995). In a second approach, the system accepts information describing the nature of an item, and based on a sample of the user's preferences, learns to predict which items the user will like (Lang 1995; Pazzani, Muramatsu, & Billsus 1996). We will call this approach *content-based filtering,* as it does not rely on social information (in the form of other user's ratings). Both social and content-based filtering can be cast as learning problems: in both cases, the objective is to learn a function that can take a description of a user and an artifact and predict the user's preferences concerning the artifact.

Well-known recommendation systems like *Recommender* (Hill, Stead, Rosenstein & Furnas 1995) and *Firefly* (http: //www.firefly.net) (Shardanand & Maes 1995) are based on social-filtering principles. *Recommender*, the baseline system used in the work reported here, recommends as yet unseen movies to a user based on his prior ratings of movies and their similarity to the ratings of other users. Social-filtering systems perform well using only numeric assessments of worth, i.e., ratings. However, there is often readily available information concerning the content of each artifact. Social-filtering methods leave open the question of what role content can play in the recommendation process.

For many types of artifacts, there is already a substantial store of information that is becoming more and more readily accessible while at the same time growing at a healthy rate. Let's take, for instance, a sample of the information a person can obtain about a favorite movie on the Web alone: a complete breakdown of cast/crew, plot, movie production details, reviews, trailer, film and audio clips, (and ratings too) and the list goes on. When users decide on a movie to see, they are likely to be influenced by data provided by one or more of these sources. Social-filtering may be characterized as a generic approach, unbiased by the regularities exhibited by properties associated with the items of interest (Hill, Stead, Rosenstein & Furnas 1995). (Indeed, a significant motivation for some of the work on such systems is to explore the utility of recognizing communities of users based solely on similarities in their preferences.) However, the fact that

*Department of Computer Science, Rutgers University, Piscataway, NJ 08855

We would like to thank Susan Dumais for useful discussions during the early stages of this work.

content-based properties can be identified at low cost (with no additional user effort and that people are influenced by these regularities make a compelling reason to investigate how best to use them.

In what situations are ratings alone insufficient? Social-filtering makes sense when there are enough other users known to the system with overlapping characteristics. Typically, the requirement for overlap in most of these systems is that the users of the system rate the same items in order to be judged similar/dissimilar to each other. It is dependent upon the current state of the system — the number of users and the number and selection of movies that have been rated.

As an example of the limitations of using ratings alone, consider the case of an artifact for which no ratings are available, such as when a new movie comes out. Since there will be a period of time when a recommendation system will have little ratings data for this movie, the recommendation system will initially not be able to recommend this movie reliably. However, a system which makes use of content might be able to make predictions for this movie even in the absence of ratings.

In this paper, we present a new, inductive learning approach to recommendation. We show how pure social-filtering can be accomplished using this approach, how the naive introduction of content-based information does not help — and indeed harms — the recommendation process, and finally, how the use of hybrid features that combine elements of social and content-based information makes it possible to achieve more accurate recommendations. We use the problem of movie recommendation as our exploratory domain for this work since it provides a domain with a large amount of data (over 45,000 movie evaluations across more than 250 people), as well as a baseline social-filtering method to which we can compare our results (Hill, Stead, Rosenstein & Furnas 1995).

The Movie Recommendation Problem

As noted above, in the social filtering approach, a recommendation system is given as input a set of ratings of specific artifacts for a particular user. In recommending movies, for instance, this input would be a set of movies that the user had seen, with some numerical rating associated with each of these movies. The output of the recommendation system is another set of artifacts, not yet rated by the user, which the recommendation system predicts the user will rate highly.

Social-filtering systems would solve this problem by focusing solely on the movie ratings for each user, and by computing from these ratings a function that can give a rating to a user for a movie that others have rated but the user has not. These systems have traditionally output ratings for movies, rather than a binary label. They compute ratings for unseen objects by finding similarities between peoples' preferences about the rated items. Similarity assessments are made amongst individual users of a system and are computed using a variety of statistical techniques. For example, *Recommender* computes for a user a smaller group of reference users known as recommenders. These recommenders are other members of the community most similar to the user. Using regression techniques, these recommenders' ratings are then used to predict ratings for new movies. In this social recommendation approach recommended movies are usually presented to the user as a rank-ordered list.

Content-based recommendation systems, on the other hand, would reflect solely the non-ratings information. For each user they would take a description of each liked and disliked movie, and learn a procedure that would take the description of a new movie and predict whether it will be liked or disliked by the user. For each user a separate recommendation procedure would be used.

Our Approach

The goal of our work is to develop an approach to recommendation that can exploit both ratings and content information. We depart from the traditional social-filtering approach to recommendation by framing the problem as one of classification, rather than artifact rating. On the other hand, we differ from content-based filtering methods in that social information, in the form of other users' ratings, will be used in the inductive learning process.

In particular, we will formalize the movie recommendation problem as a learning problem—specifically, the problem of learning a function that takes as its input a user and a movie and produces as output a label indicating whether the movie would be liked (and therefore recommended) or disliked:

$$f(\langle user, movie \rangle) \rightarrow \{liked, disliked\}$$

As a problem in classification, we also are interested in predicting whether a movie is liked or disliked, not an exact rating. Our output is also not an ordered list of movies, but a set of movies which we predict will be liked by the user. Most importantly, we are now able to generalize our inputs to the problem to other information describing both users and movies.

The information we have available for this process is a collection of user/movie ratings (on a scale of 1-10), and certain additional information concerning each movie.[1] To present the results as sets of movies predicted to be liked or disliked by a user we compute a ratings threshold for each user such that 1/4 of all the user's ratings exceed and the remaining 3/4 do not, and we return as recommended any movie whose predicted rating is above the training-data-based threshold on movies.

[1]It would be desirable to make the recommendation process a function of user attributes such as age or gender, but since that information is not available in the data we are using in this paper, we are forced to neglect it here.

Below we will outline a number of alternative ways that a user/movie rating might be represented for the learning system. We will first describe how we represent social recommendation information, which we call "collaborative" features, then how we represent "content" features, and finally describe the hybrid features that form the basis for our most successful recommendation system.

Collaborative Features

As an initial representation we use a set of features that take into account, separately, user characteristics and movie characteristics. For instance, perhaps a group of users were identified as liking a specific movie:

> Mary, Bob, and Jill liked *Titanic.*

We defined an attribute called *users who liked movie X* to group users like these into a single feature, the value of which is a set. (*E.g.*, { Mary, Bob, Jill} would be the value of the feature *users who liked movie X* for the movie *Titanic*). Since our ground ratings data contain numerical ratings, we say a user likes a movie if it is rated in the top-quartile of all movies rated by that user.[2]

We also found it important to note that a particular user was interested in a set of movies, namely the ones which appeared in his top-quartile:

> Tim liked the movies, *Twister*, *Eraser*, and *Face/Off.*

This led us to develop an attribute, *movies liked by user*, which encoded a user's favorite movies as another set-valued feature. We called these attributes *collaborative features* because they made use of the data known to social-filtering systems: users, movies, and ratings.

The result of this is that every user/movie rating gets converted into a tuple of two set-valued features. The first attribute is a set containing the movies the given user liked, and can be thought of as a single attribute describing the user. The second attribute is a set containing the users who like the given movie, and can be thought of as a single attribute describing the movie. Each such tuple is labeled by whether it was liked or disliked by the user, according to whether it was in the top-quartile for the user.

The use of set-valued features led naturally to use of *Ripper*, an inductive learning system that is able to learn from data with set-valued attributes (Cohen 1995; 1996). *Ripper* learns a set of rules, each rule containing a conjunction of several tests. In the case of a set-valued feature f, a test may be of the form "$e_i \in f$" where e_i is some constant that is an element of f in some example. As an example, *Ripper* might learn a rule containing the test *Jaws* $\in$ *movies-liked-by-user.*

[2]The value of 1/4 was chosen rather arbitrarily, and our results are similar when this value was changed to 20% or 30%.

Content Features

Content features are more naturally available in a form suitable for learning, since much of the information concerning a movie are available from (semi-) structured online repositories of information. An example of such a resource which we found very useful for movie recommendation is the Internet Movie Database (IMDb) (http: //www.imdb.com). The IMDb contains an extensive collection of movies and factual information relating to movies. All of our content features were extracted from this resource. In particular, the features we used in our experiments using "naive" content features were: Actors, Actresses, Directors, Writers, Producers, Production Designers, Production Companies, Editors, Cinematographers, Composers, Costume Designers, Genres, Genre Keywords, User-submitted Keywords, Words in Title, Aka (also-known-as) Titles, Taglines, MPAA rating, MPAA reason for rating, Language, Country, Locations, Color, Soundmix, Running Times, and Special Effects Companies.

Hybrid Features

Our final set of features reflect the common human-engineering effort that involves inventing good features to enable successful learning. Here this resulted in *hybrid features*, arising from our attempts to merge data that was not purely content-based nor collaborative. We looked for content that was frequently associated with the movies in our data and that is often used when choosing a movie. One such content feature turned out to be a movie's *genre.* However, to make effective use of the *genre* feature, it turned out to be necessary to relax an apparently natural assumption: that a $\langle user, movie \rangle$ pair would be encoded as a set of collaborative features, plus a set of content features describing the movie. Instead, it turned out to be more effective to define new collaborative features that are influenced by content. We call these features *hybrid* features.

We isolated three of the most frequently occurring genres in our data — comedy, drama, and action. We then introduced features that isolated groups of users who liked movies of the same genre, such as *users who liked dramas.* Similar features were defined for comedy and action movies. These hybrid features combine knowledge about users who liked a set of movies with knowledge of a particular content feature associated with the movies in a set. Definitions concerning what it means for a user to like a movie remain the same (top-quartile) as in the earlier parts of this paper.

Experiments and Results

We conducted a number of experiments using different sets of features. Below we will report on some of the significant results.

Training and Test Data

Our data set consists of more than 45,000 movie ratings collected from approximately 260 users. This data originated from a data set that was used to evaluate *Recommender*. However, over the course of our work we discovered that the training and test distributions in this data were distributed very differently. We therefore generated a new partition of data into a training set which contained 90% of the data and a testing set which contained the remaining 10%, for which the two distributions would be be more similar. Unfortunately, for some of the users *Recommender* failed to run correctly, and those few users were dropped from this study. Note that this was the only reason for dropping users. No users were dropped due to the performance of our own methods.

We generated a testing set by taking a *stratified random sample* of the data, in the following way:

- For every user, separate and group his movie/rating pairs into intervals defined by the ratings. Movies are rated on a scale from 1 to 10.
- For each interval, take a random sample of 10% of the data and combine the results.

Among the advantages of using stratified random sampling (Moore 1985), the primary one for us is that we have clearly defined intervals where all the units in an interval share a common property, the rating. Therefore, the holdout set we computed is more representative of the distribution of ratings for the entire data set than it would have been if we had used simple random sampling.

Evaluation Criteria

As mentioned earlier, we differ from other approaches in the output that we desire. This stems from how we compare ratings of different movies to deal with similarity. Rather than getting the exact rating right, we are interested in predicting whether a movie would be amongst the user's favorites. This has the nice effect of dealing with the fact that the intervals on the ratings scale are *not equidistant*. For instance, given a scale of 1 to 10 where 1 indicates low preference and 10, high preference, the "qualitative" difference between a rating of 1 and a rating of 2 is less when compared to the difference between 6 and 7, for any user whose ratings are mostly 7 and above. Our evaluating a movie as being liked if it is in the top-quartile reflects our belief that knowing the actual rating of a movie is not as important as knowing where the rating was relative to other ratings for a given user.

Both (Hill, Stead, Rosenstein & Furnas 1995) and (Karunanithi & Alspector 1996) evaluate the recommendations returned by their respective systems using *correlation* of ratings. For instance, they compared how well their results correlated with actual user ratings and the ratings of movie critics. Strong positive correlations are indicative of good recommendations. However, since we are not predicting exact ratings, we cannot use this method of evaluation.

We instead use two metrics commonly used in information retrieval — *precision* and *recall*. Precision gives us an estimate of how many of the movies predicted to be in the top-quartile for a user really belong to that group. Recall estimates how many of all the movies in the user's top-quartile were predicted correctly. A system that returns all movies as liked can achieve high recall. On the other hand, if we are more generous and consider all movies except those in the lowest quartile as liked, then we would expect precision estimates to increase. Therefore, we cannot consider any one measure in isolation.

However, we feel that when recommending movies, the user is more interested in examining a small set of recommended movies rather than a long list of candidates. Unlike document retrieval, where the user can narrow a list of retrieved items by actually reading some of the documents, here, the user is really interested in seeing just one movie. Therefore, our objective for movie recommendation is to maximize precision without letting recall drop below a specified limit. Precision represents the fact that a movie selected from the returned set will be liked, and the recall cutoff reflects the fact that there should be a non-trivial number of movies returned (for example, in case a video store is out of some of the recommended titles).

Baseline Results

In our initial experiment, we use *Recommender*'s social-filtering methods to compute predictions for $\langle user, movie \rangle$ pairs on the holdout data set. To do this, for every user, we separate his data from the holdout set. The rest of the data is made available to *Recommender*'s analysis routines, which means that every other user's test data serves as part of the training data for a given hold-out-user's test data. Then, for every movie in the user's holdout data, we apply *Recommender*'s evaluation routines to compute a rating. These routines look for a set of recommenders correlated with the user and compute a rating for a movie using a prediction equation with the recommenders as variables.

For every rating computed by *Recommender*, we need to determine whether it is in the top-quartile. To do this, we precompute *thresholds* for every user corresponding to the ratings which separate the top from the lower quartiles. To convert a rating, we use this rule:

- If a *predicted rating* >= *user's threshold*, set the rating to "+".
- Otherwise, set the rating to "−".

These thresholds are set individually for each user, using only the training data ratings for the training data threshold, but the full set of data for a user is used to set the testing data threshold.

Our precision estimates are *microaveraged* (Lewis 1991). Microaveraging meant that our prediction decisions were made from a single group and an overall precision estimate was computed. This is preferable to *macroaveraging*, in which one computes results on a per individual basis and averages them at the end, giving equal weight to each user. Unfortunately, in some cases (due to the small amount of data for some users) no movies were recommended, leaving precision ill-defined in these cases. Microaveraging does not suffer this problem (unless no movies are returned for any users). As shown in Table 1, the *Recommender* achieved microaveraged values of 78% for precision and 33% for recall.

Inductive Learning Results

In the first of our inductive learning recommendation experiments using *Ripper*, we use the same training and holdout sets described above. However, now every data point is represented by a collaborative feature vector. The collaborative features we used were:

- Users who liked the movie
- Users who disliked the movie
- Movies liked by the user

The ratings are converted to the appropriate binary classification as described earlier. The entire training set and holdout set are made available to *Ripper* in two separate files. We then ran *Ripper* on this data and generated a classification for each example in the holdout set. *Ripper* produces a set of rules that it learns for this data which it uses to make predictions about the class of an example.

When running *Ripper*, we have the choice of setting a number of parameters. The parameters we found most useful in adjusting from the default settings allow ***negative tests in set-valued attributes*** and varying the *loss ratio*. The first parameter allows the tests in rules to check for non-containment of attribute values within a set-valued feature. (*E.g.*, tests like *Jaws* $\notin$ *movies-liked-by-user* are allowed.) The *loss ratio* is the ratio of the perceived cost of a false positive to the cost of a false negative; increasing this parameter encourages *Ripper* to improve precision, generally at the expense of recall. In most of the experiments, we varied the loss ratio until we achieved a high value of precision with a a reasonable recall. At a loss ratio of 1.9, we achieved a microaveraged precision of 77% and a recall of 27% (see Table 1). This level of precision is comparable to *Recommender*, but at a lower level of recall.

In the second set of experiments, we replaced the collaborative feature vector with a new set of features. In our studies, we extracted 26 different features from the IMDb. The features we chose ranged from common attributes such as *actors* and *actresses* to lesser known features such as *taglines*. We also chose a few features which were assigned to movies by users, such as *keyword* descriptors.

We began by adding the 26 content features to the collaborative features. With these new features, we were not able to improve precision and recall at the same time (see Table 1). Recalling that high precision was more important to us than high recall, we find these results generally inferior to that of *Recommender*. Furthermore, examining the rules that *Ripper* generated, we found that content features were seldom used.

Two points should be noted from this experiment. First, the collaborative data appear to be better predictors of user preferences than our initial encoding of content; as a result, *Ripper* learned rules which ignored all but a few of the content features. Secondly, given the high dimensionality of our feature space, it appears to be difficult to make reasonable associations amongst the examples in our problem.

In our next attempt, we created features that combined collaborative with content information relating to the genre of a movie. These hybrid features were:

- Comedies liked by user
- Dramas liked by user
- Action movies liked by user

Although the movies in our data set are not limited to these three genres, we took a conservative approach to adding new features and began with the most popular genres as determined by the data.

To introduce the next set of collaborative features, we face a new issue. For example, we want a feature to represent the set of users who liked comedies. Although we have defined what it means to like a movie, we have not defined what it means to like movies of a particular genre. How many of the movies in the user's top-quartile need to be of a particular genre in order for the user to like movies of that genre?

Surveying the data, we found that the proportion of movies of any particular genre appearing in a user's top-quartile usually fall into some broad clusters. As a first cut, we divided the proportions of movies of different genres into four groups and created features to reflect the degree to which the user liked a particular genre. For each of the popular genres, *comedy*, *drama*, and *action*, we defined the following features:

- Users who liked many movies of genre X
- Users who liked some movies of genre X
- Users who liked few movies of genre X
- Users who disliked movies of genre X

We also add features including, for example, the genre of a particular movie. Running *Ripper* on this data with a loss ratio of 1.5, we achieved a microaveraged precision of 83% with a recall of 34%. These results are summarized in Table 1.

Using the standard test for a difference in proportions (Mendenhall, Scheaffer, & Wackerly 1981, pages 311-315) it can be determined that *Ripper* with hybrid features attains a statistically significant improvement

Method	*Precision*	*Recall*
Recommender	78%	33%
Ripper (no content)	77%	27%
Ripper (simple content)	73%	33%
Ripper (hybrid features)	83%	34%

Table 1: Results of the different recommendation approaches.

over the baseline *Recommender* system with respect to precision ($z = 2.25$, $p > 0.97$), while maintaining a statistically indistinguishable level of recall.[3] *Ripper* with hybrid features also attains a statistically significant improvement over *Ripper* without content features with respect to both precision ($z = 2.61, p > 0.99$) and recall ($z = 2.61, p > 0.998$).

Observations

Our results indicate that an inductive approach to learning how to recommend can perform reasonably well when compared to social-filtering methods, evaluated on the same data. We have also shown that by formulating recommendation as a problem in classification, we are able to combine meaningfully information from multiple sources, from ratings to content.

At equal levels of recall, our evaluation criteria would favor results with higher precision. Our results using hybrid features show that even with high precision, we also have a slight edge over recall as well.

We can comment on our features in terms of their effects on recall and precision. When we try to improve recall we are trying to be more inclusive — to add more items in our pot at the expense of unwanted items. On the flip side, when we improve precision, we are being more selective about those items we add. Features like *users who like comedies* help to increase recall. They are a generalization of simple collaborative features like *users who liked movie X*. Features like *comedies liked by user* have the reverse effect. They are a specialization of the collaborative feature, *movies liked by user*, and thereby focus our attention on a subset of a larger space of examples and increase precision.

Related Work

We have already described previous work on recommendation in our discussion of the *Recommender* system. There has also been work which explored the use of content features in selecting movies, in the context of another system designed on social-filtering principles. This previous study compared *clique-based* and *feature-based* models for movie selection (Karunanithi & Alspector 1996). A clique is a set of users whose movie ratings are similar, comparable to the set of recommenders in (Hill, Stead, Rosenstein & Furnas 1995). Those members of the clique who have rated a movie that the user has not seen predict a rating for that movie. Clique formation is dependent upon two parameters. The first is a correlation threshold which is the minimum correlation necessary to become a member of another user's clique. The second is a size threshold which defines a lower limit on the number of movies that a user must see and rate to become a of that clique. In their implemention, the authors set the size parameter to a constant value of 10 and set the correlation threshold such that the number of users in the clique is held at a constant 40. After a clique is formed, a movie rating is estimated by calculating the arithmetic mean of the ratings of the members of the clique. This mean serves as the predicted rating for the user. The authors also outline a general algorithm for a feature-based approach to recommendation:

1. Given a collection of rated movies, extract features for those movies.
2. Build a model for the user where the features serve as input and the ratings as output.
3. For every new movie not seen by the user, estimate the rating based on the features of the movie.

They used a neural-network model which associated these features (inputs to the model) with movie ratings (outputs of the model).

In this study, the authors isolated six features describing movies (not necessarily gathered from the IMDb): MPAA ratings, Category (genre), Maltin (critic's) rating, Academy Award, Length of movie, and Origin (related to the country of origin). They justified their choice of features on the grounds that they wanted to start with as small a set of features as possible, and that they found these features easiest to encode for their model.

The category and MPAA ratings were first fed into hidden units. Unlike the other features, which were nominal valued, these two features had a 1-of-N unary encoding. In other words, the feature is encoded as a N-bit vector where each bit represents one of the feature's possible values. (Only one bit corresponding to the feature's value in the example is set.) Although this representation is suited for their model and allows the feature to take multiple values, it is limited to the extent that all the values need to be enumerated at the outset.

In our case, the majority of features, content as well as collaborative, turned out to be set-valued. Set-valued features are more flexible than the 1-of-N unary features. These values can grow over time and need not be predetermined. Computationally, set-valued features are also much more efficient to work with than the corresponding 1-of-N encoding, particularly in cases for which N is large.

In the feature-based study, the authors found that by using features, in most cases, they outperformed a

[3] More precisely, one can be highly confident that there is no practically important loss in recall relative to the baseline; with confidence 95%, the recall rate for *Ripper* with hybrid features is at least 32.8%.

human critic but almost consistently did worse than the clique method. Our initial results with content features supported these findings. However, we also demonstrated that content information can lead to improved recommendations, if encoded in an appropriate manner.

Fab (Balabanovic & Shoham 1997) is a system which tackles both issues of content-based filtering and social-filtering. In the *Fab* system, content information is maintained by two types of agents: *user agents* associated with individuals and *collection agents* associated with sets of documents. Each collection agent represents a different topic of interest. Each of these agent-types maintains its own profiles, consisting of terms extracted from documents, and uses these profiles to filter new documents. These profiles are reinforced over time with user feedback, in the form of ratings, for new documents. In so doing, the goal is to evolve the agents to better serve the interests of the user and the larger community of users (who receive documents from the collection agents). There are some key differences in our approach. Ours is not an agent-based framework. We do not have access to topics of interest information, which in Fab, were collected from the users. We also do not use ratings as relevance feedback for updating profile information. Since we are not dealing with documents, we do not employ IR techniques for feature extraction.

Another well known social-filtering system is *Firefly*. *Firefly*, which has since expanded beyond the domain of music recommendation, is a descendant of *Ringo* (Shardanand & Maes 1995), a music recommendation system. *Ringo* presents the user with a list of artists and albums to rate. This system maintains this information on behalf of every user, in the form of a user profile. The profile is a record of the user's likes and dislikes and is updated over time as the user submits new ratings. The profile is used to compare an individual user with others who share similar tastes. During similarity assessment, the system selects profiles of other users with the highest correlation with an individual user. In the *Ringo* system, two of the metrics used to determine similarity are *mean-squared difference* and the *Pearson-R measure*. In the first case, *Ringo* makes predictions by thresholding with respect to how dissimilar two profiles are based on their mean-squared difference. Then, it computes a weighted average of the ratings provided by the most similar users. In the second case, *Ringo* makes predictions by using Pearson-R coefficients as weights in a weighted-average of other users' ratings.

Final Remarks

In this paper, we have presented an inductive approach to recommendation. This approach has been evaluated via experiments on a large, realistic set of ratings. One advantage of the inductive approach, relative to other social-filtering methods, is that it is far more flexible; in particular it is possible encode collaborative and content information as part of the problem representation, without making any algorithmic modifications. Exploiting this flexibility, we have evaluated a number of representations for recommendation, including two types of representations that make use of content features. One of these representations, based on hybrid features, significantly improves performance over the purely collaborative approach. We have thus begun to realize the impact of multiple information sources, including sources that exploit a limited amount of content. We believe that this work provides a basis for further work in this area, particularly in harnessing other types of information content.

References

Balabanovic, M.; and Shoham Y. 1997. Content-Based, Collaborative Recommendation. *Communications of the ACM* Vol. 40, No. 3. March, 1997.

Cohen, W. 1995. Fast Effective Rule Induction. In *Proceedings of the Twelfth Conference on Machine Learning.* Lake Taho, California.

Cohen, W. 1996. Learning Trees and Rules with Set-valued Features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence.*

Hill, W.;Stead, L.;Rosenstein, M.; and Furnas, G. 1995. Recommending and Evaluating Choices in a Virtual Community of Use. In *Proceedings of the CHI-95 Conference.* Denver, CO.

Karunanithi, N.; and Alspector, J. 1996. Feature-Based and Clique-Based User Models for Movie Selection. In *Proceedings of the Fifth International Conference on User Modeling.* Kailua-Kona, HI.

Lang, K. 1995. NewsWeeder: Learning to filter netnews. In *Machine Learning: Proceedings of the Twelfth International Conference.* Lake Taho, California: Morgan Kaufmann.

Lewis, D. 1991. Evaluating Text Categorization. In *Proceedings of the Speech and Natural Language Workshop.* Asilomar, CA.

Mendenhall, W.; Scheaffer, R.; and Wackerly, D., eds. 1981. *Mathematical Statistics with Applications.* Duxbury Press, second edition.

Moore, D. 1985. *Statistics: concepts and controversies.* W. H. Freeman.

Pazzani, M.; Muramatsu, J.; and Billsus, D. 1996. Syskill & Webert: identifying interesting web sites. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence.*

Shardanand, U.; and Maes, P. 1995. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings of the CHI-95 Conference.* Denver, CO.