Learning Collaborative Information Filters

From: AAAI Technical Report WS-98-08. Compilation copyright © 1998, AAAI (www.aaai.org). All rights reserved.

Daniel Billsus and **Michael J. Pazzani**

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
{dbillsus, pazzani}@ics.uci.edu

Abstract

Predicting items a user would like on the basis of other users' ratings for these items has become a well-established strategy adopted by many recommendation services on the Internet. Although this can be seen as a classification problem, algorithms proposed thus far do not draw on results from the machine learning literature. We propose a representation for collaborative filtering tasks that allows the application of virtually any machine learning algorithm. We identify the shortcomings of current collaborative filtering techniques and propose the use of learning algorithms paired with feature extraction techniques that specifically address the limitations of previous approaches. Our best-performing algorithm is based on the singular value decomposition of an initial matrix of user ratings, exploiting latent structure that essentially eliminates the need for users to rate common items in order to become predictors for one another's preferences. We evaluate the proposed algorithm on a large database of user ratings for motion pictures and find that our approach significantly outperforms current collaborative filtering algorithms.

Introduction

Research on intelligent information agents in general, and recommender systems in particular, has recently attracted much attention. The reasons for this are twofold. First, the amount of information available to individuals is growing steadily. Second, the number of users accessing the Internet is also growing, which opens up new possibilities to organize and recommend information. The central idea here is to base personalized recommendations for users on information obtained from other, ideally like-minded, users. This is commonly known as collaborative filtering or social filtering.

A variety of collaborative filtering algorithms have previously been reported in the literature and their promising performance has been evaluated empirically (Shardanand and Maes, 1995; Resnick et al. 1994). However, the reported algorithms are based on rather simple predictive techniques. Although collaborative filtering can be seen as a classification task, the problem has not received much attention in the machine learning community. It seems likely that predictive performance can be increased through the development of special-purpose algorithms that draw on results from the machine learning literature.

Collaborative Filtering Algorithms

Shardanand and Maes, 1995, discuss and evaluate a variety of social filtering algorithms. These algorithms are based on a simple intuition: predictions for a user should be based on the similarity between the interest profile of that user and those of other users. Suppose we have a database of user ratings for items, where users indicate their interest in an item on a numeric scale. It is now possible to define similarity measures between two user profiles, *U* and *J*. A measure proposed by Shardanand and Maes is the *Pearson correlation coefficient,* r_{UJ}. Once the similarity between profiles has been quantified, it can be used to compute personalized recommendations for users. All users whose similarity is greater than a certain threshold *t* are identified and predictions for an item are computed as the weighted average of the ratings of those similar users for the item, where the weight is the computed similarity. Resnick et al., 1994, compute predictions according to the following formula, where U_x is a rating to be predicted for User *U* on item *x* and r_{UJ} is the correlation between users *U* and *J*.

$$U_x = \overline{U} + \frac{\sum_{J \in Raters\ of\ x} (J_x - \overline{J})\, r_{UJ}}{\sum_{J \in Raters\ of\ x} |r_{UJ}|}$$

While these correlation-based prediction schemes were shown to perform well, they suffer from several limitations. Here, we identify three specific problems: First, correlation between two user profiles can only be computed based on items that both users have rated, i.e. the summations and averages in the correlation formula are only computed over those items that both users have rated. If users can choose among thousands of items to rate, it is likely that overlap of rated items between two users will be small in many cases. Therefore, many of the computed correlation coefficients are based on just a few observations, and thus the computed correlation cannot be regarded as a reliable measure of similarity. For example, a correlation coefficient based on three observations has as much influence on the final prediction as a coefficient based on 30 observations. Second, the correlation approach induces one global model of similarities between users, rather than separate models for classes of ratings (e.g. positive rating vs. negative rating). Current approaches measure whether two user profiles are positively corre-

lated, not correlated at all or negatively correlated. However, ratings given by one user can still be good predictors for ratings of another user, even if the two user profiles are not correlated. Consider the case where user A's positive ratings are a perfect predictor for a negative rating from user B. However, user A's negative ratings do not imply a positive rating from user B, i.e. the correlation between the two profiles could be close to zero, and thus potentially useful information is lost. Third, and maybe most importantly, two users can only be similar if there is overlap among the rated items, i.e. if users did not rate any common items, their user profiles cannot be correlated. Due to the enormous number of items available to rate in many domains, this seems to be a serious stumbling block for many filtering services, especially during the startup phase. However, just knowing that users did not rate the same items does not necessarily mean that they are not like-minded. Consider the following example: Users A and B are highly correlated, as are users B and C. This relationship provides information about the similarity between users A and C as well. However, in case users A and C did not rate any common items, a correlation-based similarity measure could not detect any relation between the two users.

Collaborative Filtering as a Classification Problem

Collaborative filtering can be seen as a classification task. Based on a set of ratings from users for items, we are trying to induce a model for each user that allows us to classify unseen items into two or more classes, for example *like* and *dislike*. Alternatively, if our goal is to predict user ratings on a continuous scale, we have to solve a regression problem.

Our initial data exists in the form of a sparse matrix, where rows correspond to users, columns correspond to items and the matrix entries are ratings. Note that *sparse* in this context means that most elements of the matrix are empty, because every user typically rates only a very small subset of all possible items. The prediction task can now be seen as filling in the missing matrix values. Since we are interested in learning personalized models for each user, we associate one classifier (or regression model) with every user. This model can be used to predict the missing values for one row in our matrix.

Table 1: Exemplary User Ratings

	I_1	I_2	I_3	I_4	I_5
U_1	4		3		
U_2		1		2	
U_3	3	4	2		4
U_4	4	2	1		?

With respect to Table 1, consider that we would like to predict user 4's rating for item 5. We can train a learning algorithm with the information that we have about user 4's previous ratings. In this example user 4 has provided 3 ratings, which leads to 3 training examples: I_1, I_2, and I_3. These training examples can be directly represented as feature vectors, where users correspond to features (U_1, U_2, U_3) and the matrix entries correspond to feature values. User 4's ratings for I_1, I_2 and I_3 are the class labels of the training examples. However, in this representation we would have to address the problem of many missing feature values. If the learning algorithm to be used cannot handle missing feature values, we can apply a simple transformation. Every user can be represented by up to n Boolean features, where n is the number of points on the scale that is used for ratings. For example, if the full n-point scale of ratings is used to represent ratings from m users, the resulting Boolean features are of the form "User m's rating was i", where $0 < i \leq n$. We can now assign Boolean feature values to all of these new features. If this representation leads to an excessive number of features that only appear rarely throughout the data, the rating scale can be further discretized, e.g. into the two classes *like* and *dislike*. The resulting representation is simple and intuitive: a training example E corresponds to an item that the user has rated, the class label C is the user's discretized rating for that item, and items are represented as vectors of Boolean features F_i.

After converting a data set of user ratings for items into this format, we can draw on the machine learning literature and apply virtually any supervised learning algorithm that, through analysis of a labeled training sample $T = \{E_j, C_j\}$, can induce a function $f : E \rightarrow C$.

Reducing Dimensionality

Our goal is to construct or apply algorithms that address the previously identified limitations of correlation-based approaches. As mentioned earlier, the computation of correlation coefficients can be based on too little information, leading to inaccurate similarity estimates. When applying a learning algorithm, we would like to avoid this problem. In particular, we would like to discard information that we do not consider informative for our classification task. Likewise, we would like to be able to take possible interaction and dependencies among features into account, as we regard this as an essential prerequisite for users to become predictors for one another's preferences even without rating common items. Both of these issues can be addressed through the application of appropriate feature extraction techniques. Furthermore, the need for dimensionality reduction is of particular importance if we represent our data in the proposed learning framework. For large databases containing many users we will end up with thousands of features while our amount of training data is very limited. This is, of course, not a problem unique to collaborative filtering. Other domains with very similar requirements include the classification of natural language text or, in general, any information retrieval task. In these domains the similarity among text documents needs to be measured. Ideally, two text documents should be similar if they discuss the same subject or contain related information. How-

ever, it is often not sufficient to base similarity on the overlap of words. Two documents can very well discuss similar subjects, but use a somewhat different vocabulary. A low number of common words should not imply that the documents are not related. This is very similar to the problem we are facing in collaborative filtering: the fact that two users rated different items should not imply that they are not like-minded. Researchers in information retrieval have proposed different solutions to the text version of this problem. One of these approaches, Latent Semantic Indexing (LSI) (Deerwester et al., 1990) is based on dimensionality reduction of the initial data through singular value decomposition (SVD).

Collaborative Filtering and the SVD

We start our analysis based on a rectangular matrix containing Boolean values that indicate user ratings for items. This matrix is typically very sparse, where *sparse* means that most elements are zero, because each item is only rated by a small subset of all users. Furthermore, many features appear infrequently or do not appear at all throughout this matrix. However, features will only affect the SVD if they appear at least twice. Therefore, we apply a first preprocessing step and remove all features that appear less than twice in our training data. The result of this preprocessing step is a matrix A containing zeros and ones, with at least two ones in every row. Using the SVD, the initial matrix A with r rows, c columns and rank m can be decomposed into the product of three matrices:

$$A = U \sum V^T$$

where the columns of U and V are orthonormal vectors that define the *left* and *right* singular vectors of A, and Σ is a diagonal matrix containing corresponding singular values. U is an $m \times c$ matrix and the singular vectors correspond to columns of the original matrix. V is an $r \times m$ matrix and the singular vectors correspond to rows of the original matrix. The singular values quantify the amount of variance in the original data captured by the singular vectors. This representation provides an ideal framework for dimensionality reduction, because one can now quantify the amount of information that is lost if singular values and their corresponding singular vector elements are discarded. The smallest singular values are set to zero, reducing the dimensionality of the new data representation. The underlying intuition is that the n largest singular values together with their corresponding singular vector elements capture the important "latent" structure of the initial matrix, whereas random fluctuations are eliminated. The usefulness of the SVD for our task can be further explained by its geometric interpretation. If we choose to retain the k largest singular values, we can interpret the singular vectors, scaled by the singular values, as coordinates of points representing the rows and columns of the original matrix in k dimensions. In our context, the goal of this transformation is to find a spatial configuration such that items and user ratings are represented by points in k-dimensional space, where every item is placed at the centroid of every user rating that it received and every user rating is placed at the centroid of all the items that it was assigned to. While the position of vectors in this k-dimensional space is determined through the assignment of ratings to items, items can still be close in this space even without containing any common ratings.

Using Singular Vectors as Training Examples

Our training data is a set of rated items, represented as Boolean feature vectors. We compute the SVD of the training data and discard the n smallest singular values, reducing the dimensionality to k. Currently, we set k to $0.9 \cdot m$, where m is the rank of the initial matrix. This value was chosen because it resulted in the best classification performance (evaluated using a tuning set). The singular vectors of matrix U scaled by the remaining singular values represent rated items in k dimensions. These vectors become our new training examples. Since we compute the SVD of the training data, resulting in real-valued feature vectors of size k, we need to specify how we transform examples to be classified into this format. We compute a k-dimensional vector for an item, so that with appropriate rescaling of the axes by the singular values, it is placed at the centroid of all the user ratings that it contains.

At this point we need to pick a suitable learning algorithm that takes real-valued feature vectors as its input and learns a function that either predicts class membership or computes a score a user would assign to an item. Ideally, we would like to use a learning paradigm that allows for maximum flexibility in evaluating this task as either a regression or classification problem. Therefore, we selected artificial neural networks as the method of choice for our purposes (Rumelhart and McLelland, 1986). We ran various experiments on a tuning set of the data available to us, to determine a network topology and learning paradigm that resulted in good performance. The winning approach was a feed-forward neural network with k input units, 2 hidden units and 1 output unit. The hidden units use sigmoid functions, while the output unit is linear. Weights are learned with backpropagation. Although the task at hand might suggest using a user's rating as the function value to predict, we found that a slightly different approach resulted in better performance. We determined the average rating for an item and trained the network on the difference between a user's rating and the average rating. In order to predict scores for items, the output of the network needs to be added to the mean of the item. We then used a threshold t (depending on the rating scale of the domain) to convert the predicted rating to a binary class label.

EXPERIMENTAL EVALUATION

We ran experiments using data from the *EachMovie* collaborative filtering service. The *EachMovie* service was part of a research project at the Systems Research Center of Digital Equipment Corporation. The database contains ratings from 72,916 users for 1,628 movies. User ratings were recorded on a numeric six-point scale (0.0, 0.2, 0.4,

0.6, 0.8, 1.0). Although data from 72,916 users is available, we restrict our analysis to the first 2,000 users in the database. These 2,000 users provided ratings for 1,410 different movies. We restricted the number of users considered, because we are interested in the performance of the algorithm under conditions where the ratio of users to items is low. This is a situation that every collaborative filtering service has to go through in its startup-phase, and in many domains we cannot expect to have significantly more users than items. We also believe that the deficiencies of correlation-based approaches will be more noticeable under these conditions, because it is less likely to find users with considerable overlap of rated items.

Performance Measures

We defined two classes, *hot* and *cold*, that were used to label movies. When transforming movies to training examples for a particular user, we labeled movies as *hot* if the rating for the movie was *0.8* or *1.0*, or *cold* otherwise. We classify items as *hot* if the predicted rating exceeds the threshold *0.7*. Not only does assigning class labels allow us to measure classification accuracy, we can also apply additional performance measures, such as *precision* and *recall*, commonly used for information retrieval tasks.

It is important to evaluate *precision* and *recall* in conjunction, because it is easy to optimize either one separately. In order to do this, (Lewis and Gale, 1994) proposed the *F-measure*, a weighted combination of precision and recall that produces scores ranging from 0 to 1. Here we assign equal importance to precision and recall, i.e. $F = (2 \cdot precision \cdot recall) / (precision + recall)$. Since we see the F-measure as a useful construct to compare classifiers, but think that it is not an intuitive measure to indicate a user's perception of the usefulness of an actual system, we use an additional measure: precision at the top n ranked items (here, we report scores for $n = 3$ and $n = 10$).

Experimental Methodology

Since we are interested in the performance of the algorithms with respect to the number of ratings provided by users, we report learning curves where we vary the number of rated items from 10 to 50. For each user we ran a total of 30 paired trials for each algorithm. For an individual trial of an experiment, we randomly selected 50 rated items to use as a training set, and 30 as a test set. We then started training with 10 examples out of the set of 50 and increased the training set incrementally in steps of 10, measuring the algorithms' performance on the test set for each training set size. Final results for one user are then averaged over all trials. We repeated this for 20 users and the final curves reported here are averaged over those 20 users.

We determined parameters for our algorithms using a tuning set of 20 randomly selected users. The results reported here are averaged over 20 different users. We selected users randomly, but with the following constraints. First, only users whose prior probability of liking a movie is below 0.75 are considered. Otherwise, scores that indicate high precision of our algorithms might be biased by the fact that there are some users in the database who either like everything or just gave ratings for movies they liked. Second, only users that rated at least 80 movies were selected, so that we could use the same number of training and test examples for all users.

Summary of Results

Figure 1 summarizes the performance of three different algorithms. The algorithm labeled *Correlation* uses the prediction formula as described in (Resnick et al. 1994). The algorithm labeled *SVD/ANN* is our dimensionality reduction approach coupled with a neural network. Since this algorithm is a combination of a feature extraction technique (SVD) and a learning algorithm (ANN), the observed performance does not allow us to infer anything about the relative importance of each technique individually. Therefore, we report the performance of a third algorithm, labeled *InfoGain/ANN*, in order to quantify the importance of our proposed feature extraction technique. *InfoGain/ANN* uses the same neural network setup as *SVD/ANN*, but applies a different feature selection algorithm. Here, we compute the expected information gain (Quinlan, 1986) of all the initial features and then select the n most informative features, where n is equivalent to the number of features used by *SVD/ANN* for each training set size. Since expected information gain cannot detect interaction and dependencies among features, the difference between *SVD/ANN* and *InfoGain/ANN* allows us to quantify the utility of the SVD for this task. The results show that both *SVD/ANN*, as well as *InfoGain/ANN*, performed better than the correlation approach. In addition, *SVD/ANN* is more accurate and substantially more precise than *InfoGain/ANN*. At 50 training examples *Correlation* reaches a classification accuracy of 64.4%, vs. 67.9% for *SVD/ANN*. While predictive accuracy below 70% might initially seem disappointing, we need to keep in mind that our goal is not the perfect classification of all movies. We would like to have a system that identifies many interesting items and does this with high precision. This ability is measured by the F-measure and we can see that *SVD/ANN* has a significant advantage over the correlation approach (at 50 examples 54.2% for *Correlation* vs. 68.8% for *SVD/ANN*). Finally, if we restrict our analysis to the top 3 or top 10 suggestions of each algorithm, we can see that *SVD/ANN* is much more precise than the other two algorithms. At 50 training examples *Correlation* reaches a precision of 72.6% at the top 3 suggestions, *InfoGain/ANN*'s precision is 78.3% and *SVD/ANN* reaches 83.9%.

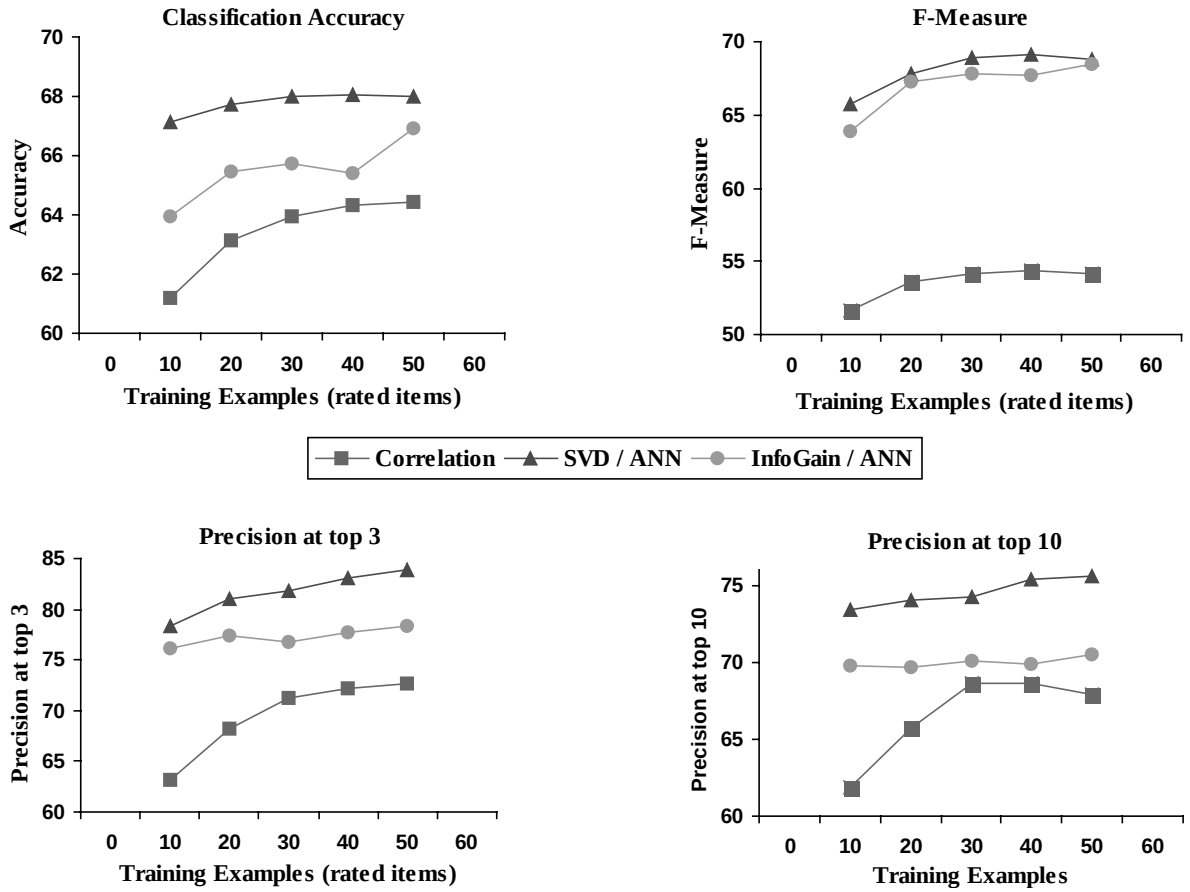

Figure 1: Learning Curves

SUMMARY AND CONCLUSIONS

In this paper we have identified the shortcomings of correlation-based collaborative filtering techniques and shown how these problems can be addressed through the application of classification algorithms. The contributions of this paper are twofold. First, we have presented a representation for collaborative filtering tasks that allows the use of virtually any machine learning algorithm. We hope that this will pave the way for further analysis of the suitability of learning algorithms for this task. Second, we have shown that exploiting latent structure in matrices of user ratings can lead to improved predictive performance. In a set of experiments with a database of ratings for movies, we used the singular value decomposition to project user ratings and rated items into a lower dimensional space. This allows users to become predictors for one another's preferences even without any overlap of rated items.

References

Deerwester, D., Dumais, S. T., Landauer, T. K., and Furnas, G.W., and Harshman, R.A. (1990). "Indexing by latent semantic analysis." *Journal of the Society for Information Science, 41(6)*, 391-407.

Lewis, D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. In Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, 3-12, London, Springer-Verlag.

Pazzani M., and Billsus, D. (1997). Learning and Revising User Profiles: The identification of interesting web sites. *Machine Learning 27*, 313-331.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Resnick, P., Neophytos, I., Mitesh, S. Bergstrom, P. and Riedl, J. (1994) GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of CSCW94: Conference on Computer Supported Cooperative Work, 175-186, Chapel Hill, Addison-Wesley.

Rumelhart, D. E. and McClelland, J. L. (eds) (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: The MIT Press.

Shardanand, U. and Maes, P. Social Information Filtering: Algorithms for Automating 'Word of Mouth', In Proceedings of CHI95, 210-217, Denver, CO, ACM Press.