Information Retrieval, 5, 287–310, 2002

An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms

JON HERLOCKER herlock@cs.orst.edu
Department of Computer Science, Oregon State University, 102 Dearborn Hall, Corvallis, OR 97331-3202, USA

JOSEPH A. KONSTAN konstan@cs.umn.edu
JOHN RIEDL riedl@cs.umn.edu
GroupLens Research Project, Department of Computer Science and Engineering, University of Minnesota, USA

Received November 29, 2000; Revised September 28, 2001; Accepted January 23, 2002

Abstract. Collaborative filtering systems predict a user's interest in new items based on the recommendations of other people with similar interests. Instead of performing content indexing or content analysis, collaborative filtering systems rely entirely on interest ratings from members of a participating community. Since predictions are based on human ratings, collaborative filtering systems have the potential to provide filtering based on complex attributes, such as quality, taste, or aesthetics. Many implementations of collaborative filtering apply some variation of the neighborhood-based prediction algorithm. Many variations of similarity metrics, weighting approaches, combination measures, and rating normalization have appeared in each implementation. For these parameters and others, there is no consensus as to which choice of technique is most appropriate for what situations, nor how significant an effect on accuracy each parameter has. Consequently, every person implementing a collaborative filtering system must make hard design choices with little guidance. This article provides a set of recommendations to guide design of neighborhood-based prediction systems, based on the results of an empirical study. We apply an analysis framework that divides the neighborhood-based prediction approach into three components and then examines variants of the key parameters in each component. The three components identified are similarity computation, neighbor selection, and rating combination.

Keywords: collaborative filtering, information filtering, empirical studies, preference prediction

1. Introduction

Automated collaborative filtering is quickly becoming a popular technique for reducing information overload, often as a technique to complement content-based information filtering systems. Automated collaborative filtering has seen considerable success on the Internet, being used at sites like Amazon.com—the largest book store on the Internet and CDNow.com—the largest CD store on the Web.

Content-based and collaborative filtering use different types of data to arrive at a filtering decision. Content-filtering tools select the right information for the right people by comparing representations of content contained in the documents to representations of content that the user is interested in. Content-based information filtering has proven to be effective in locating textual documents relevant to a topic using techniques such as vector-space

queries, "intelligent" agents, and information visualization. Automated collaborative filtering systems work by collecting human judgments (known as ratings) for items in a given domain and matching together people who share the same information needs or the same tastes. Users of a collaborative filtering system share their analytical judgments and opinions regarding each item that they consume so that other users of the system can better decide which items to consume. In return, the collaborative filtering system provides useful personalized recommendations for interesting items.

Collaborative filtering provides three key additional advantages to information filtering that are not provided by content-based filtering: (i) support for filtering items whose content is not easily analyzed by automated processes; (ii) the ability to filter items based on quality and taste; and (iii) the ability to provide serendipitous recommendations.

First, in collaborative filtering, humans determine the relevance, quality, and interest of an item in the information stream. Consequently, filtering can be performed on items that are hard to analyze with computers, such as movies, ideas, feelings, people, and politicians.

Second, collaborative filtering systems can enhance information-filtering systems by measuring, in dimensions beyond that of simple content, how well an item meets a user's need or interests. Humans are capable of analyzing on dimensions such as quality or taste, which are very hard for computer processes. A content-based search of the Associated Press could retrieve all articles related to Minnesota Governor Jesse Ventura, but by combining content filtering with collaborative filtering, a search could return only those relevant articles that are well written!

Finally, a collaborative filtering system will sometimes make serendipitous recommendations—recommending items that are valuable to the user, but do not contain content that the user was expecting. We have found that serendipitous recommendations occur frequently in the movie domain, with the collaborative filtering system accurately recommending movies that a user would never have considered otherwise.

The potential for collaborative filtering to enhance information-filtering tools is great. However, to reach the full potential, it must be combined with existing content-based information filtering technology. Collaborative filtering by itself performs well predicting items that meet a user's interests or tastes, but is not well suited to locating information for a focused content need.

In this article, we present an algorithmic framework for performing collaborative filtering and examine empirically how existing algorithm variants perform under this framework. We present new, effective enhancements to existing prediction algorithms and finally conclude with a set of recommendations for selection of prediction algorithm variants.

This article is a further exploration of the concepts presented in a previous SIGIR conference paper (Herlocker et al. 1999). This article examines additional parameters and provides deeper discussion beyond that of the original conference paper. The datasets are also slightly different—details on the difference can be found in Section 4.1.

2. Problem space

The problem of automated collaborative filtering is to predict how well a user will like an item that he has not rated given that users ratings for other items and a set of historical ratings for a community of users.

Table 1. Collaborative filtering can be represented as the problem of predicting missing values in a user-item matrix.

	Star wars	Hoop dreams	Contact	Titanic
Joe	5	2	5	4
John	2	5		3
Al	2	2	4	2
Nathan	5	1	5	?

This is an example of a user-item rating matrix where each filled cell represents a user's rating for an item. The prediction engine is attempting to provide Nathan a prediction for the movie 'Titanic.'

A prediction engine collects ratings and uses collaborative filtering technology to provide predictions. An active user provides the prediction engine with a list of items, and the prediction engine returns a list of predicted ratings for those items. Most prediction engines also provide a recommendation mode, where the prediction engine returns the top predicted items for the active user from the database.

The problem space can be formulated as a matrix of users versus items, with each cell representing a user's rating on a specific item. Under this formulation, the problem is to predict the values for specific empty cells (i.e. predict a user's rating for an item). In collaborative filtering, this matrix is generally very sparse, since each user will only have rated a small percentage of the total number of items. Table 1 shows a simplified example of a user-rating matrix where predictions are being computed for movies.

The most prevalent algorithms used in collaborative filtering are what we call the neighborhood-based methods. In neighborhood-based methods, a subset of appropriate users are chosen based on their similarity to the active user, and a weighted aggregate of their ratings is used to generate predictions for the active user. Other algorithmic methods that have been used are Bayesian networks (Breese et al. 1998), singular value decomposition with neural net classification (Billsus and Pazzani 1998), inductive rule learning (Basu et al. 1998), a graph-theoretic approach (Aggarwal et al. 1999), a Bayesian mixed-effect model (Condliff et al. 1999), a combination of neighborhood-based algorithms with weighted majority weighting (Delgado and Ishii 1999), clustering in reduced dimensions using principal component analysis (Goldberg et al. 2001) and latent class models (Hofmann and Puzicha 1999).

As an example of a neighborhood based method, consider Table 1 again. We wish to predict how Nathan will like the movie "Titanic." Joe is Nathan's best neighbor, since the two of them have agreed closely on all movies that they have both seen. Therefore, Joe's opinion of the movie Titanic will influence Nathan's prediction the most. John and Al are not as good neighbors because both of them have disagreed with Nathan on certain movies; thus, they will influence Nathan's predictions less than Joe will.

In this article, we will explore the space of neighborhood-based collaborative filtering methods and describe some new better performing algorithms that we have developed. Neighborhood-based methods can be separated into three steps.

1. Weight all users with respect to similarity with the active user.
2. Select a subset of users to use as a set of predictors (possibly for a specific item)
3. Normalize ratings and compute a prediction from a weighted combination of selected neighbors' ratings.

Within specific systems, these steps may overlap or the order may be slightly different.

We will begin by discussing the most relevant prior work on collaborative filtering algorithms, examining which techniques were used to implement the three steps of neighborhood-based methods.

3. Related work

The GroupLens system first introduced an automated collaborative filtering system using a neighborhood-based algorithm (Resnick et al. 1994). GroupLens provided personalized predictions for Usenet news articles. The original GroupLens system used Pearson correlations to weight user similarity, used all available correlated neighbors, and computed a final prediction by performing a weighted average of deviations from the neighbor's mean (Eq. (1)). $P_{a,i}$ represents the prediction for the active user for item i. n is the number of neighbors, $r_{u,i}$ is user u's rating of item i, $\bar{r}_a$ is the active user's average rating, and $w_{a,u}$ is the similarity weight between the active user and neighbor u—as defined by the Pearson correlation coefficient, which is shown in Eq. (2).

$$P_{a,i} = \bar{r}_a + \frac{\sum_{u=1}^{n}[(r_{u,i} - \bar{r}_u) * w_{a,u}]}{\sum_{u=1}^{n} w_{a,u}} \tag{1}$$

$$w_{a,u} = \frac{\sum_{i=1}^{m}[(r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)]}{\sqrt{\sum_{i=1}^{m}(r_{a,i} - \bar{r}_a)^2 \sum_{i=1}^{m}(r_{u,i} - \bar{r}_u)^2}} \tag{2}$$

The Ringo music recommender (Shardanand and Maes 1995) and the Bellcore Video Recommender (Hill et al. 1995) expanded upon the original GroupLens algorithm. Ringo claimed better performance by computing similarity weights using a constrained Pearson correlation coefficient, shown in Eq. (3). The value 4 was chosen because it was the midpoint of Ringo's seven-point rating scale. Ringo limited membership in a neighborhood by only selecting those neighbors whose correlation was greater than a fixed threshold, with higher thresholds resulting in greater accuracy, but reducing the number of items for which Ringo was able to generate predictions for. To generate predictions, Ringo computed a weighted average of ratings from all users in the neighborhood.

$$w_{a,u} = \frac{\sum_{i=1}^{m}[(r_{a,i} - 4)(r_{u,i} - 4)]}{\sqrt{\sum_{i=1}^{m}(r_{a,i} - 4)^2 \sum_{i=1}^{m}(r_{u,i} - 4)^2}} \tag{3}$$

The Bellcore Video Recommender used Pearson correlation to weight a random sample of neighbors, selected the best neighbors, and performed a full multiple regression on them to create a prediction.

Breese et al. (1998) performed an empirical analysis of several variants of neighborhood-based collaborative filtering algorithms. For similarity weighting, Pearson correlation and cosine vector similarity were compared, with correlation being found to perform better, although work done after this study suggests that they may be equivalent (Pennock et al. 2000b).

A promising nearest-neighbor-like approach called "Personality Diagnosis" has been proposed by Pennock et al. (2000b). Personality diagnosis combines the approaches of Bayesian modeling and neighborhood-based methods. Personality diagnosis has the nice property that it produces a rating likelihood distribution instead of a real valued prediction. This distribution could be used to identify the confidence of a prediction. Additionally Pennock et al. claim that personality diagnosis is more accurate than correlation-based nearest neighbor on movie data very similar to the data we used in our experiment (Pennock et al. 2000b). Personality diagnosis results are not presented here, because we became aware of the algorithm after performing our experiments described.

This article compares different variants of neighborhood-based prediction algorithms. Other work has compared neighborhood-based prediction algorithms to different prediction frameworks such as Bayesian networks techniques (Breese et al. 1998) and classical data-mining algorithms (Sarwar et al. 2000a). In both cases, the neighborhood-based prediction algorithm was shown to be more accurate for multi-valued (non-binary) rating data.

4. Experimental design

4.1. Data

In order to compare the results of different neighborhood based prediction algorithms we ran a prediction engine using historical ratings data collected for purposes of anonymous review from the MovieLens movie recommendation site. The historical data consisted of 100,000 ratings from 943 users, with every user having at least 20 ratings. The data set covers ratings entered into MovieLens during a three-month period of time, after removing all users with less than 20 ratings. Over the entire MovieLens database (currently more than 7 million ratings), the average number of rated movies per user is 71. The ratings were on a five-point scale with 1 and 2 representing negative ratings, 4 and 5 representing positive ratings, and 3 indicating ambivalence.[1] Unlike many alternative algorithms, e.g. Breese et al. (1998) and Sarwar et al. (2000b) the neighborhood-based prediction algorithms can operate on sparse user-item rating matrices, and do not require the use of default voting (such as is described in Breese et al. (1998)).

The dataset used in this article is a subset of the one used in the original SIGIR conference paper, consisting of 100,000 ratings instead of 122,000 ratings (covering a slightly shorter timespan) (Herlocker et al. 1999). The holdout sets in this analysis are different from the ones in the SIGIR paper; originally 10% of each user's ratings were withheld, while in this analysis, we withhold exactly 10 ratings from each user. In spite of the difference in the holdout set, the findings were consistent. Thus, we view the results presented here as a superset of the results presented in the conference paper.

4.2. Experimental method

From each user in the test set, ratings for 10 items were withheld, and predictions were computed for those 10 items using each variant of the tested neighborhood based prediction algorithms. Since users are more likely to rate good items than bad items, the results may reflect predictive accuracy for generally good items more than for generally bad items.[2]

For each item predicted, the highest-ranking neighbors that have rated the item in question are used to compute a prediction (they form the user's neighborhood for that item). Note that this means that a user may have a different neighborhood for each item. All users in the database are examined as potential neighbors for a user—no sampling is performed.

The quality of a given prediction algorithm can be measured by comparing the predicted values for the withheld ratings to the actual ratings.

4.3. Metrics

For a full discussion of metrics in collaborative filtering, please refer to Herlocker (2000). For this experiment, we consider the following metrics.

4.3.1. Coverage. Coverage is a measure of the percentage of items for which a recommendation system can provide predictions. Common system features that can reduce coverage are small neighborhood sizes and sampling of users to find neighbors. We compute coverage as the percentage of prediction requests for which the algorithm was able to return a prediction. Unless otherwise noted, all experimental results demonstrated in this article had maximal coverage. Maximal coverage may be slightly less than perfect (99.8 in our experiments) because there may be no ratings in the data for certain items, or because very few people rated an item, and those that did had zero correlations or no overlap with the active user. In certain applications it may be possible to trade off coverage for accuracy (for example, by only presenting confident predictions to the user), however our assumption in this research has been that close to maximal coverage is important.

4.3.2. Accuracy. To measure the accuracy of predictions, we chose mean-absolute error as a prediction error evaluation metric. There have been many different predictive accuracy metrics applied to collaborative filtering results, including mean-absolute error (Breese et al. 1998, Herlocker et al. 1999, Pennock et al. 2000b, Resnick et al. 1994, Shardanand and Maes 1995), correlation (Hill et al. 1995, Sarwar et al. 1998), ROC sensitivity (Good et al. 1999, Herlocker et al. 1999), mean-squared error (Shardanand and Maes 1995), precision/recall (Sarwar et al. 2000a), and a rank-utility metric (Breese et al. 1998). We have done empirical experiments that have shown that mean absolute error correlates strongly with many other proposed metrics for collaborative filtering (Herlocker 2000), yet is easier to measure and has well understood significance measures. Furthermore, mean absolute error is still the most frequently used metric among collaborative filtering researchers.

Table 2. A list of prediction algorithm components tested and the variants of each component that were tested.

Component	Variants tested
Similarity weight	Pearson correlation
	Spearman correlation
	Entropy
	Mean-squared-difference
Significance weighting	No significance weighting
	$n/50$ weighting
Variance weighting	None
	$(\text{Variance} - \text{Variance}_{\min})/(\text{Variance}_{\max} - \text{Variance}_{\min})$
Selecting neighborhoods	Weight threshold—Full range
	Max neighborhood size—Full range
Rating normalization	No normalization
	Deviation from mean
	Z-score
Weighting neighbor contributions	Using weighting
	Without weighting

4.4. Parameters evaluated

The components that were evaluated are listed in Table 2 along with the variations of each component that were tested. For each component, the performance using each of the variations was measured. All components except the one being measured were held constant to ensure that the results reflected the differences in the component being tested.

5. Weighting possible neighbors

The first step in neighborhood-based prediction algorithms is to weight all users with respect to similarity with the active user. When you are given recommendations for movies or books, you are more likely to trust those that come from people who have historically proven themselves as providers of accurate recommendations. Likewise, when automatically generating a prediction, we want to weight neighbors based on how likely they are to provide an accurate prediction. Thus, we compute similarity weights to measure closeness between users. However, we have only incomplete data specifying the history of agreement between users; in certain cases, we have only a small sample of ratings on common items for pairs of users. We have to be wary of false indicators of predictive relationships between users. To address these issues we adjust the similarity weight with significance weighting and variance weighting.

5.1. Similarity weighting

Several different similarity weighting measures have been used. The most common weighting measure used is the Pearson correlation coefficient. Pearson correlation

(Eq. (2)) measures the degree to which a linear relationship exists between two variables.

The Spearman rank correlation coefficient is similar to Pearson, but computes a measure of correlation between ranks instead of rating values. To compute Spearman's correlation, we first covert the user's list of ratings to a list of ranks, where the user's highest rating gets rank 1. Tied ratings get the average of the ranks for their spot. Then the computation is the same as the Pearson correlation, but with ranks substituted for ratings (Eq. (4)). $k_{a,i}$ represents the rank of the active user's rating of item i. $k_{u,i}$ represents the rank of neighbor u's rating for item i.

$$w_{a,u} = \frac{\sum_{i=1}^{m}[(k_{a,i} - \bar{k}_a)(k_{u,i} - \bar{k}_u)]}{\sqrt{\sum_{i=1}^{m}(k_{a,i} - \bar{k}_a)^2 \sum_{i=1}^{m}(k_{u,i} - \bar{k}_u)^2}} \tag{4}$$

Mean-squared difference (Eq. (5)) is another alternative that was used in the Ringo music recommender (Shardanand and Maes 1995). Mean-squared difference gives more emphasis to large differences between user ratings than small differences.

$$d = \frac{\sum_{i=1}^{m}(r_{a,i} - r_{u,i})^2}{m} \tag{5}$$

In our experiments, we have found that Spearman correlation performs similarly to Pearson correlation. As an example, figure 1 shows results for both Spearman and Pearson correlation across a variety of different values of max_nbors,[3] with all other parameters held constant. Note that the results are very close between Spearman and Pearson. The same data is shown in Table 3, along with indications of statistical significance. With the exception of one data point in Table 4, there was no statistical difference between the algorithm using Pearson and the algorithm using Spearman. Occasionally, such as with the max-nbors = 60 data point, Pearson correlation will perform significantly better than Spearman.

The performance of the mean-squared difference as a similarity metric is also shown in figure 1. The graph shows that mean-squared difference results in lower prediction accuracy than either Spearman or Pearson correlation. Table 4 shows the same data on a point-by-point comparison with algorithms using Pearson correlation.

Given the data we have observed, Spearman correlation does not appear to be valuable. Algorithms using Spearman correlation perform worse or the same as comparable algorithms using Pearson correlation. Furthermore, computation of Spearman correlation is much more compute-intensive, due to the additional pass through the ratings necessary to compute the ranks.

The lack of improvement using Spearman correlation was surprising. We had thought that the inherent ranking characteristics in the rating process would cause the rank-based correlation to perform better. We believe that the large number of tied rankings (there are only five distinct ratings—five distinct ranks—for each user) results in a degradation of the accuracy of Spearman correlations. Increasing the size of the ranking scale could increase the difference in accuracy between Spearman and Pearson, however, with larger ratings scales, there is the issue of rate-rerate reliability.[4]

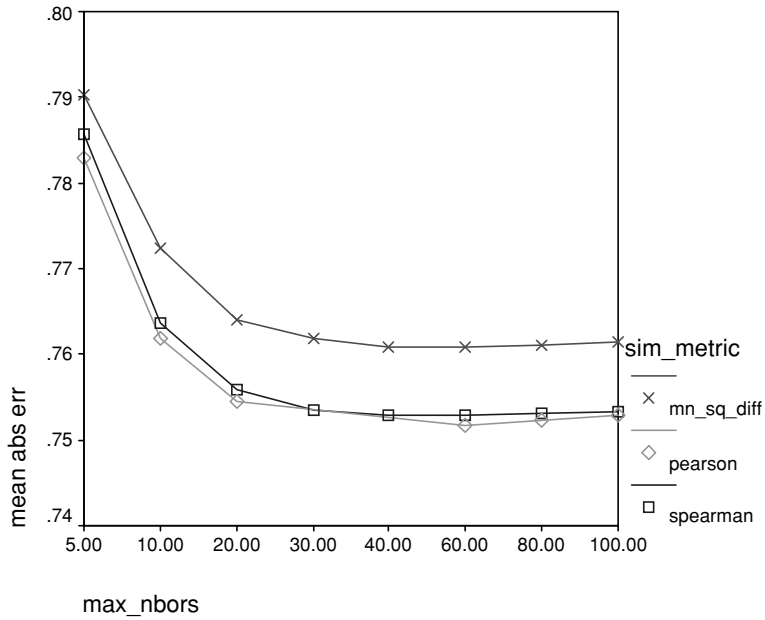

Figure 1. This graph compares the performance of algorithms using three different correlation measures as similarity metrics. Spearman is a rank-based metric, Pearson is the standard product-moment correlation, and mn_sq_diff is the mean squared difference of the ratings. The performance of the two correlation algorithms is effectively the same. The data used for this chart is weighted_average_deviation_from mean, negative correlations = 0, devalue = 50, and no threshold.

Other similarity measures include the vector similarity "cosine" measure, the entropy-based uncertainty measure, and the mean-squared difference algorithm. The vector similarity measure has been shown to be successful in information retrieval, however Breese has found that vector similarity does not perform as well as Pearson correlation in collaborative filtering (Breese et al. 1998). The measure of association based on entropy uses conditional probability techniques to measure the reduction in entropy of the active user's ratings that results from knowing another user's ratings (Press et al. 1986). In our tests, entropy has not performed as well as Pearson correlation.

5.2. *Significance weighting*

One issue that has not been addressed in previously published studies is the amount of trust to be placed in a correlation with a neighbor. In our experience with collaborative filtering systems, we have found that it was common for the active user to have highly correlated neighbors that were based on a very small number of co-rated items. These neighbors that were based on tiny samples (often three to five co-rated items) frequently proved to be terrible predictors for the active user. The more data points that we have to compare the opinions of two users, the more we can trust that the computed correlation is representative of

Table 3. This table shows the mean absolute error for algorithms using Pearson and Spearman correlation.

Max-nbors	Pearson MAE	Spearman MAE	Significance	*P*-value
5	0.7829	0.7855	No	>0.05
10	0.7618	0.7636	No	>0.05
20	0.7545	0.7558	No	>0.05
60	0.7518	0.7529	Yes	0.05
80	0.7523	0.7531	No	>0.05
100	0.7528	0.7533	No	>0.05

This is the same data as shown in figure 1. The *P*-value indicates the strength of significance as measured by a large-sample paired hypothesis test. The items to hold out have been randomly selected.

Table 4. This table shows the mean absolute error for algorithms using Pearson correlation and mean squared difference as similarity measures. This is the same data as shown in figure 1.

Max-nbors	Pearson MAE	mn_sq_diff MAE	Significance	*P*-value
5	0.7829	0.7898	Yes	0.05
10	0.7618	0.7718	Yes	0.001
20	0.7545	0.7634	Yes	<0.001
60	0.7518	0.7602	Yes	<0.001
80	0.7523	0.7605	Yes	<0.001
100	0.7528	0.7610	Yes	<0.001

the true correlation between the two users. We hypothesized that the accuracy of prediction algorithms would be improved if we were to add a correlation significance-weighting factor that would devalue similarity weights that were based on a small number of co-rated items. For our experiments, we applied a linear drop-off to correlations that were based on less than a certain threshold of number of items. For example, if our significance threshold was 50, and if two users had fewer than 50 commonly rated items, we multiplied their correlation by a significance weight of $n/50$, where n is the number of co-rated items. If there were more than 50 co-rated items, then no adjustment was applied. In this manner, correlations with small numbers of co-rated items are devalued, but correlations with 50 or more commonly co-rated items are not dependent on the number of co-rated items.

Figure 2 compares the accuracy with and without the devaluing term, at different significance cutoff levels. The line with a significance cutoff of 1 represents the use of non-weighted raw correlations. Increasing the cutoff to 10 actually causes an increase in error, which was quite surprising. Our interpretation of this result is that 10 is not a high enough significance threshold to catch the worst offending false correlations, and therefore simply enters noise into the system. However, significance thresholds of 25 or more do improve the accuracy of the system. Such high thresholds effectively remove the non-performing neighbors from your neighborhood. Significance thresholds larger than 50 don't seem to have much significant additional effect on the accuracy of the system.

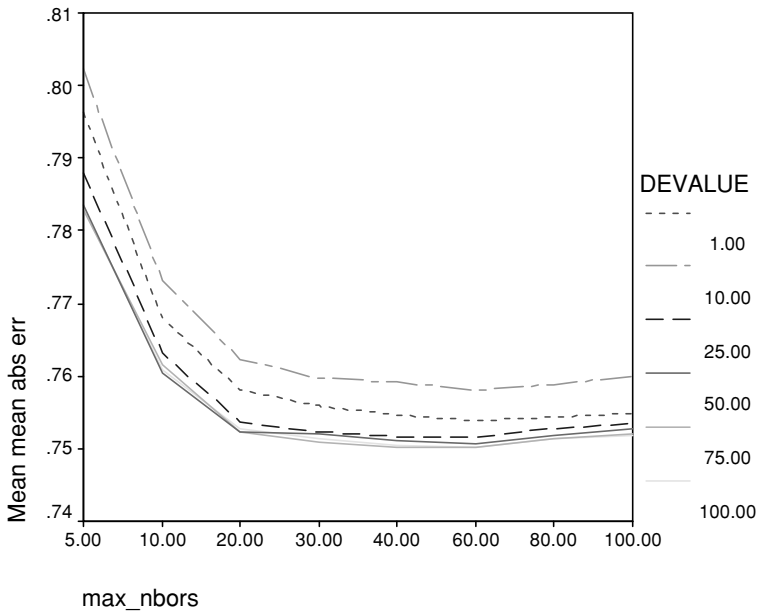

Figure 2. Shows the effects of significance weighting on prediction accuracy. *Devalue* indicates the number of shared ratings a correlation must be based on to not get devalued by a significance weighting. All correlations less than the specified constant were devalued linearly. There are 48 data points represented in this chart, with Pearson similarity measure, weighted neighbor contributions, and deviation from mean combination normalization. No devaluing (DEVALUE = 1) is significantly worse than devaluing with a threshold 25 or greater (DEVALUE $\geq$ 25). Significance is computed through a large sample paired hypothesis test.

Note that applying devaluing based on significance thresholds has several other advantages. As you increase the significance threshold, users need to have rated more items in common with you in order to enter your neighborhood. The fact that those users have seen more of the same movies as you is an indication in of itself of potential commonality of tastes, independent of the ratings. In addition, if you are choosing neighborhoods independent of the items for which ratings are being predicted, high significance thresholds will make it more likely that the neighbors selected have rated a large number of items, thus making them more useful in the generation of predictions.

An additional interesting observation is that devaluing correlations has a noticeably weaker effect on prediction accuracy if a non-weighted average of neighbor ratings is used to generate a prediction. Figure 3 shows the effects of devaluing on such algorithms. Compare this to figure 2, which contains data from algorithms where weighted averages of ratings were used.

5.3. Variance weighting

All the similarity measures described above treat each item evenly in a user-to-user correlation. However, knowing a user's rating on certain items is more valuable than others in

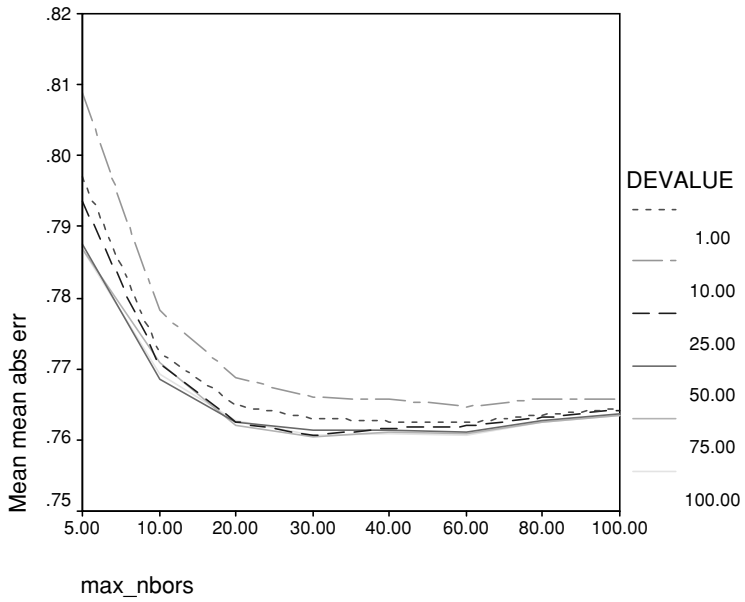

Figure 3. This graph is similar to figure 2, except the 48 points shown here do not weight neighbor contribution to the prediction. Since the similarity measure is only once (to find the neighborhoods), and not twice (it is not used for weighting contributions), the effect of devaluing the similarity measure is smaller.

discerning a user's interest. For example, we have found that the majority of MovieLens users have rated the movie "Titanic" highly. Therefore knowing that two users rated Titanic high tells us very little about the shared interests of those two users. Opinions on other movies have been known to distinguish users' tastes. The movie "Sleepless in Seattle" has shown itself to separate those users who like action movies from those who like romance movies. Knowing that two people agree on "Sleepless in Seattle" tells us a lot more about their shared interests than Titanic would have. We hypothesized that giving the distinguishing movies more influence in determining a correlation would improve the accuracy of the prediction algorithm. To achieve this, we modified the mean-squared difference algorithm to incorporate an item-variance weight factor. We added a variance weight for each item, then adding a normalizing factor in the denominator. This is shown in Eq. (6). By incorporating a variance weight term, we will increase the influence of items with high variance in ratings and decrease the influence of items with low variance.

$$d = \frac{\sum_{i=1}^{m} v_i (r_{a,i} - r_{u,i})^2}{\sum_{i=1}^{m} v_i} \tag{6}$$

We computed an item variance weight as $v_i = \text{var}_i - \text{var}_{\min}/\text{var}_{\max}$ where $\text{var}_i = \sum_{u=1}^{n} (r_{u,i} - \bar{r}_i)^2/n - 1$, and $\text{var}_{\min}$ and $\text{var}_{\max}$ respectively are the minimum and maximum variances over all items. Contrary to our initial hypothesis, applying variance-weighting

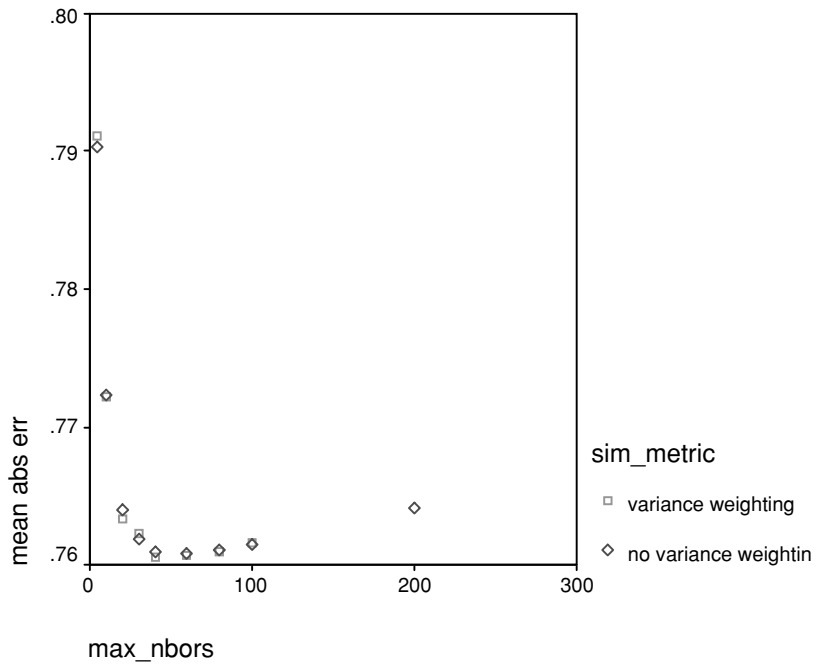

Figure 4. This graph shows the lack of difference in prediction accuracy once variance weighting was applied to the mn_sq_difference algorithm. These data points are no threshold, and weighted average deviation from mean combination.

terms had no significant effect on the accuracy of the prediction algorithm. These results are shown in figure 4. One explanation is that our variance-weighting scheme does not take into account the fact that a user who disagrees with the popular feeling provides a lot of information.

6. Selecting neighborhoods

Having assigned similarity weights to users in the database, the next step is to determine which other users' data will be used in the computation of a prediction for the active user—that is who will be selected to be in the active user's neighborhood. In theory, we could include every user in the database as a neighbor, and weight the contribution of the neighbors accordingly, with distant neighbors contributing less than close neighbors. However, commercial collaborative filtering systems are beginning to handle millions of users, making consideration of every user as a neighbor infeasible when trying to maintain real-time performance. The system must select a subset of the community to use as neighbors at prediction computation time in order to guarantee acceptable response time. Furthermore, many of the members of the community do not have similar tastes to the active user, so using them as predictors will only increase the error of the prediction.

Another consideration in selecting neighborhoods suggested by Breese is that high correlates (such as those with correlations greater than 0.5) can be exceptionally more valuable as predictors than those with lower correlations can.

Two techniques, correlation-thresholding and best-n-neighbors, have been used to determine how many neighbors to select. The first technique, used by Shardanand and Maes [shardanand], is to set an absolute correlation threshold, where all neighbors with absolute correlations greater than a given threshold are selected. The second technique, used in the GroupLens system, Resnick et al. (1994) is to pick the best n correlates for a given n.

6.1. Correlation weight threshold

Correlation thresholding sets a minimum correlation weight that a neighbor must have in order to be accepted into a user's neighborhood. This ensures that neighbors have a minimum proven predicting value. The downside of correlation thresholding is that it can significantly reduce the coverage of the prediction algorithm. If you set the correlation threshold too high, then very few people make it into a user's neighborhood. Then the ACF engine can only predict for items that those few neighbors have rated.

We tested the effect of correlation weight thresholding in combination with all the other factors. The effect of weight thresholding was consistent across all data runs. Examples of the effects of correlation weight threshold can be seen in Tables 5 and 6. In both figures, the inverse relationship between weight threshold and coverage is immediately apparent. In our data set, weight thresholds above 0.2 provide significant drops in coverage.

A somewhat surprising result was that applying a correlation weight threshold never actually improved the accuracy of the predictions. Figure 5 shows the mean absolute error

Table 5. The interaction between correlation weight threshold, coverage and mean-absolute error.

Weight threshold	Coverage (%)	Mean absolute error	Mean absolute error weight threshold = 0
0.00	100	0.7528	0.7528
0.01	100	0.7530	0.7528
0.05	99.7	0.7533	0.7522
0.10	96	0.7590	0.7445 ($p = 0$)
0.20	72	0.7521	0.7260 ($p = 0$)
0.30	55	0.7559	0.7191 ($p = 0$)
0.40	36	0.7613	0.6925 ($p = 0$)
0.50	15	0.7837	0.6864 ($p = 0$)

As the weight threshold is increased above 0.01, coverage is lost. Column 4 of this table shows the performance of the algorithm with weight-threshold = 0, but only on the items that the weight-thresholded algorithm for the corresponding row covered. It is obvious that weight thresholding in this case has reduced the accuracy of the predictions. These data points are weighted-average-deviation-from-mean, negative correlations = 0, max-nbor = 100, devalue = 50. Mean absolute errors with p values listed indicated values that have a statistically significant difference compared with the error at threshold = 0.

Table 6. The interaction between correlation weight threshold and coverage. This table is similar to figure 5.

Weight threshold	Coverage (%)	Mean absolute error	Mean absolute error weight-threshold = 0
0.00	100	0.7467	0.7467
0.01	100	0.7468	0.7467
0.05	100	0.7473	0.7461 ($p = 0.05$)
0.10	96	0.7526	0.7384 ($p = 0$)
0.20	72	0.7432	0.7184 ($p = 0$)
0.30	55	0.7452	0.7108 ($p = 0$)
0.40	36	0.7475	0.6847 ($p = 0$)
0.50	15	0.7745	0.6753 ($p = 0$)

Note the "false" gain in prediction accuracy at threshold = 0.2. The mean absolute error appears to decrease compared to no threshold. However, if you consider the accuracy of the algorithm with no threshold calculated only on items that are covered by the 0.2 threshold algorithm, we find that the 0.2 threshold algorithm is still performing worse. These data points are weighted-average-zscore, negative correlations = 0, max-nbor = 60, devalue = 50.

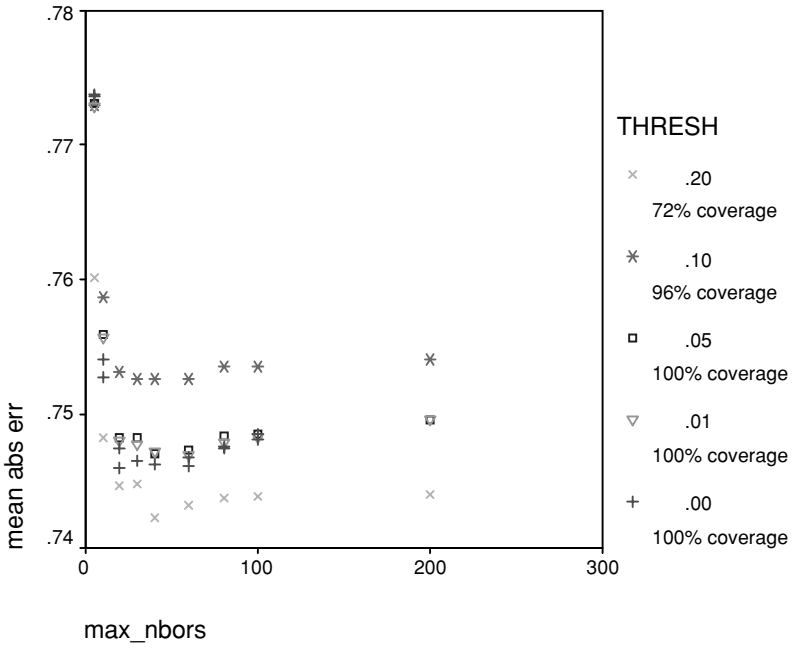

Figure 5. This graph depicts the relationship between correlation weight threshold and mean absolute error of the prediction for one class of algorithms. Points with less than 70% coverage are not shown. Notice that the 0.1 threshold increases the error. 0.2 threshold decreases the error slightly, but with a loss of 30% of coverage. The points shown in this graph are weighted-average-zscore, devalue = 50.

of a given set of algorithms for different correlation weight thresholds. It appears from figure 5 that a correlation weight threshold of 0.2 decreases the mean absolute error. We see in this in Table 5 with a decreased mean absolute error for 0.2, and we see the same for weight thresholds of 0.2 and 0.3 in Table 6. However, this turns out to be a "false" increase in mean absolute error. If we compare the mean absolute error only on points that both algorithms covered, we see that the weight threshold $= 0$ algorithm always outperforms other higher weight thresholds.

For the data set examined, correlation weight thresholding appears to have no redeeming value. In all examined cases, weight thresholding only made matters worse, decreasing both the coverage and the accuracy of the algorithm. Should the number of users in a system increase (the tested data set had 943 users), the value of using correlation weight thresholding could increase, since there is more likelihood of encountering high-correlating users. However, as the number of users increases significantly beyond 943, it become less and less practical to examine all users—forcing the system to rely on sampling. Unless new intelligent sampling techniques are developed, the sample set will be very similar to working with a smaller set of users. Furthermore, this movie rating set is relatively dense—there are a relatively large number of users who have seen the same movies as you. With more sparse data sets, such as web sites, it will extremely hard to find high correlates that happen to have seen the same web pages as you.

The improvements in prediction accuracy for the non-thresholding algorithm on only the items that were covered by the thresholding algorithms (see column 4 of Tables 5 and 6) suggest that when high correlates participate in a prediction, the quality of that prediction is likely to rise. However, the low correlates still provide a good amount of value, since discarding them resulted in lower accuracies (compare column 3 to column 4 in Tables 5 and 6).

6.2. Maximum number of neighbors used

The data shows that the "maximum number of neighbors to use" (max_nbors) parameter affects the error of the algorithm in a reasonably consistent manner. This is demonstrated in figure 6, and can also be seen in figures 1–3 on previous pages. When the size of neighborhoods is restricted by using a small max_nbors (less than 20), the accuracy of the system suffers, with increased mean absolute error (figure 6).

The loss of accuracy when using small neighborhoods was an expected result. The problem lies in the fact that, with the MovieLens data set, even the top neighbors are imperfect predictors of the active user's taste. Each user's different experiences produce many different subtleties of taste, which makes it very unlikely that there is an excellent predictor for a given user within a database of 943 users. Even if there was one user with a perfect match in tastes, the other members of the neighborhood might cloud the recommendations.

As the size of the neighborhood is allowed to increase, more users contribute to the prediction. Therefore, variance introduced by individual users is averaged out over the larger numbers. Some serendipity may be lost, since consensus is required for a high prediction, but overall accuracy increases.

We have found that even in the full MovieLens database, which contains approximately 80,000 users, there aren't many exceptionally high correlations between users once there

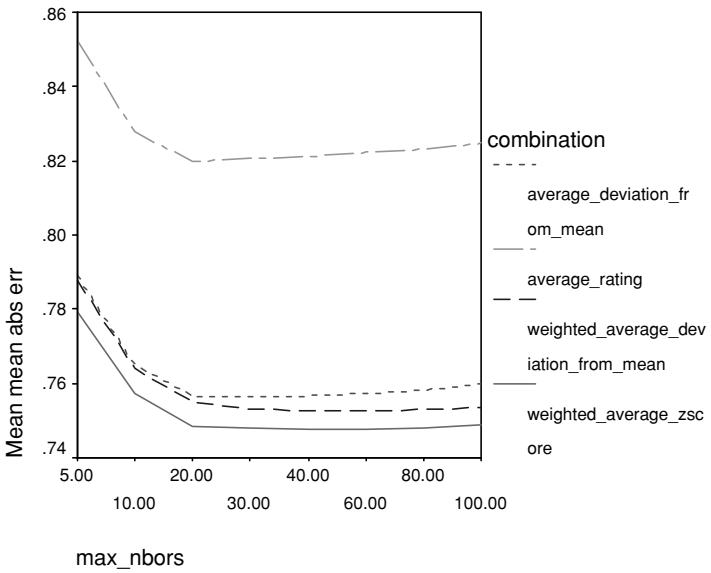

Figure 6. Graph showing the performance of algorithms when the "maximum number of neighbors to use" parameter is varied. Small neighborhood sizes cause considerable increased error. The error stabilizes around 20 neighbors. Eventually, as neighborhood sizes are increased beyond 20, the error starts to trend upwards. 32 data points are plotted, with algorithms using a significance weighting of $n/25$, and not using negative correlations. The different combination types are discussed in Section 7.

are more than 20 shared items. So, there doesn't seem to be potential for using small neighborhood sizes. In most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable, providing enough neighbors to average out extremes.

Another observation is the error increases slowly as the max_nbors parameter is increased beyond 20. As the number of neighbors consulted during a prediction increases, the probability of having to include low-correlating neighbors increases. However, this effect is much less pronounced than the small neighbor size. Consider Table 7, which shows the results of applying a paired-sample t-test to each pair of data points on the weighted_average_deviation_from_mean line of figure 6. Max_nbor values of 5 and 10 are clearly worse than max_nbor of 20, but larger values of max_nbor do not provide statistically significant differences in mean absolute error. Small values of max_nbor always resulted in an increase in error that was statistically significant.

7. Producing a prediction

Once the neighborhood has been selected, the ratings from those neighbors are combined to compute a prediction. The basic way to combine all the neighbors' ratings into a prediction is to compute an average of the ratings. The averaging technique has been used in all published work using neighborhood-based ACF algorithms. Additionally, it has been suggested by Pennock and associates that performing a weighted average of neighbor ratings is provably

Table 7. Significance of differences in MAE due to the parameter max_nbors.

Max_nbors	MAE	t-value
5	**0.7836**	**−10.18 ($p = 0$)**
10	**0.7605**	**−4.74 ($p = 0$)**
20	0.7524	0.00
30	0.7520	0.41
40	0.7511	1.11
60	0.7508	1.14
80	0.7518	0.37
100	0.7527	−0.17

Each row shows the MAE of the algorithm and the paired-sample t-value computed when comparing the algorithm to the max_nbors = 20 algorithm. This data is the same as the weighted_average_devaiation from mean curve shown in figure 6. Significant differences are in bold.

the best way to combine ratings, given certain well-accepted axioms of social choice theory (Pennock et al. 2000a).

We discuss two modifications to the ratings combination algorithm: rating normalization and weighting neighbor contributions. Both have been proposed as improvements to the ratings combination algorithm (Resnick et al. 1994).

7.1. Rating normalization

The basic weighted average assumes that all users rate on approximately the same distribution. From observation of collected data, we know that users do not all rate on the same distribution. Therefore, it makes sense to perform some sort of transformation so that user's ratings are in the same space.

The approach taken by GroupLens was to compute the average deviation of a neighbor's rating from that neighbor's mean rating, where the mean rating is taken over all items that the neighbor has rated. The deviation-from-mean approach is demonstrated in Eq. (1). The justification for this approach is that users may have rating distributions centered around different points. One user may tend to rate items higher, with good items getting 5s and poor items getting 3s, while other users may give primarily 1s, 2s, and 3s. Intuitively, if a user infrequently gives ratings of 5, then that user should not receive many predictions of 5 unless they are extremely significant. The average deviation from the mean computed across all neighbors is converted into the active user's rating distribution by adding it to the active user's mean rating.

An extension to the GroupLens algorithm is to account for the differences in spread between users' rating distributions by converting ratings to z-scores, and computing a weighted average of the z-scores (Eq. (7)).

$$p_{a,i} = \bar{r}_a + \sigma_a \frac{\sum_{u=1}^{n} \left[\left(\frac{r_{u,i} - \bar{r}_u}{\sigma_u} \right) w_{a,u} \right]}{\sum_{u=1}^{n} w_{a,u}} \tag{7}$$

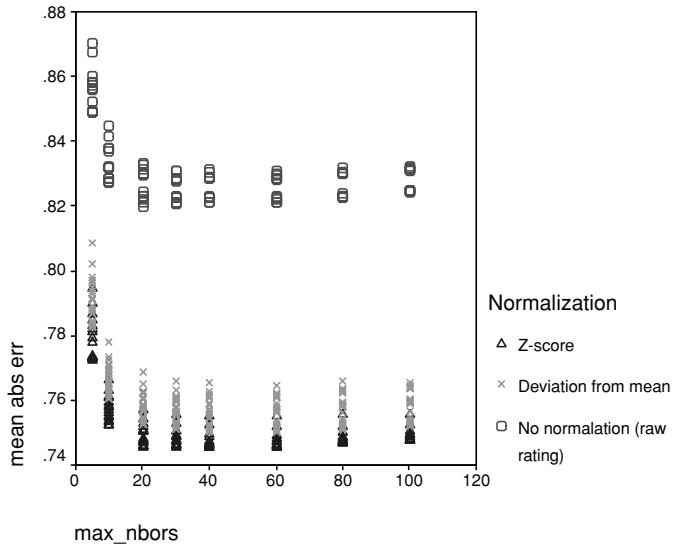

Figure 7. A scatterplot demonstrating the effect of rating normalization on prediction accuracy. Each point represents the accuracy of one algorithm. There are 404 algorithms shown here, all using Pearson correlation, but with varying significance weighting, varying values of max_nbors, and both with and without using negative correlations.

We compared the three different modes of rating normalization: no normalization, deviation-from-mean, and z-score. The results from 404 different algorithms are shown in figure 7. Note that the three different modes of normalization form three clearly identifiable bands in the chart. Performing rating normalization produces an obvious benefit. The deviation from mean normalization performs significantly better than the no normalization, while the z-score normalization only performs slightly better than the deviation from mean normalization.

The increase in accuracy due to normalization comes about because of the differences in rating distributions among users. These differences exist because of both different perceptions of the world, and different perceptions of the ratings scale. The most significant difference between user rating distributions is a lateral shift in the distribution. For example, some users tend to give primarily positive ratings, saving the less than average ratings for the worst movies. Other users may rate most movies less than average, and give the above-average ratings to the few excellent movies. Therefore, their ratings distributions are shifted when compared with each other. The rate-high user will have a high mean and median rating, while the rate-low user will have a low mean and median rating. A movie that the rate-high user rates 4 will probably only be a 3 for the rate-low user. By performing the deviation-from-mean normalization, we account for shifts in rating distributions.

The z-score normalization accounts for differences in widths of the rating distributions in addition to shifts. By widths, we mean the variance of the rating distribution. The intuition here is that some users are willing to give plentiful ratings on the extreme of their scale

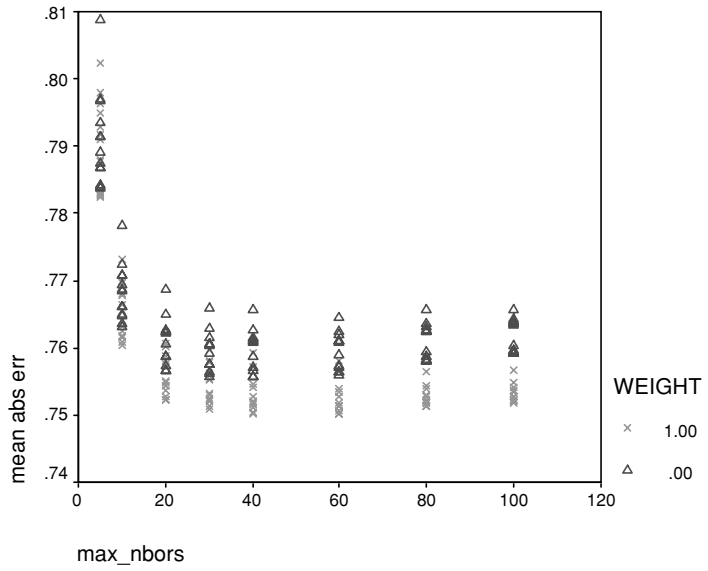

Figure 8. Demonstrates the effect of weighting neighbor's contributions to a prediction. The diamonds represent algorithms where weighting was used. The crosses represent where weighting was not used. This graph shows 204 algorithms, all using deviation-from-mean normalization.

(such as 1s and 5s), while other users may rarely give extreme ratings. Thus, their rating distributions will have different variances. By performing z-score normalization, we appropriately adjust each prediction into ratings distribution of the user receiving the prediction. As can be seen from figure 7, z-score normalization provides a small increase in accuracy over deviation-from-mean normalization. However, performing z-score normalization is relatively cheap computationally and requires only one additional storage element per user to record the user's variance of ratings.

7.2. Weighting neighbor contributions

Having computed a similarity measure to locate the closest neighbors, it makes sense to use that similarity measure to weight the contribution of each neighbor based on how close they are to the active user. The original GroupLens algorithm (see Eq. (1)), did perform this rating, while the Ringo algorithm (Shardanand and Maes 1995), did not. The Ringo algorithm simply averaged the ratings of users in the selected neighborhood. Figure 8 demonstrates the effect of weighting neighbor contributions on the accuracy of predictions. The crosses represent the accuracy of algorithms that weighted neighbor contributions, while the diamonds represent algorithms that did not weight neighbor contributions. Clearly, weighting neighbor contributions lowers the mean absolute error. There is some overlap between the two modes shown in the graph. This can be explained by variations in other parameters. What you are seeing is that the best performing algorithms without neighborhood weighting are about

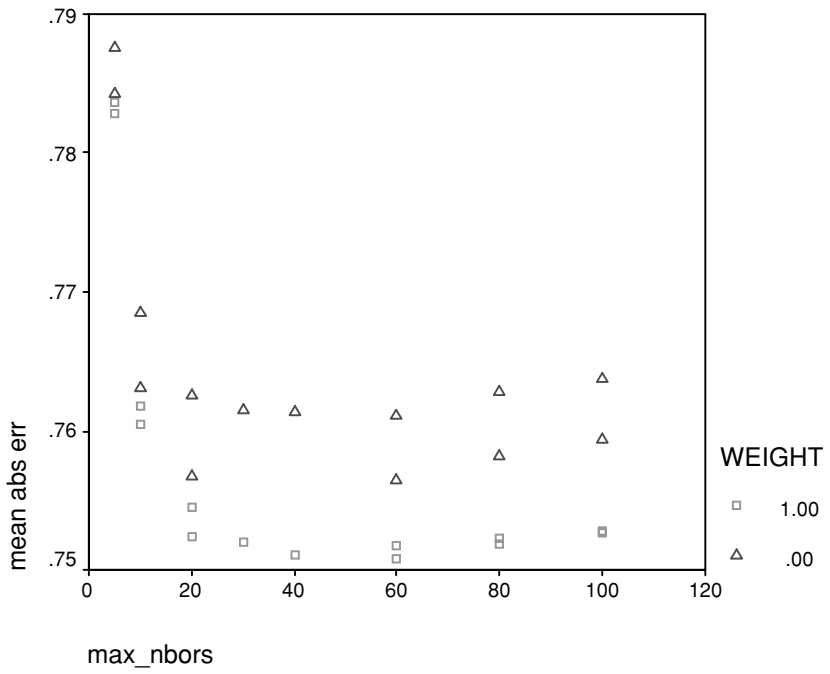

Figure 9. Indicates the clear separation between data points that weight neighbor contributions and those that don't. This was created from the data points shown in figure 8 by controlling the significance weighting factor (using $n/50$).

as good as the worst performing algorithms with neighborhood weighting. Figure 9 shows what the graph would look like when the significance weighting parameter is controlled.

8. Summary

Collaborative filtering is an exciting new approach to filtering information that can select and reject items from an information stream based on qualities beyond content, such as quality and taste. It has the potential to enhance existing information filtering and retrieval techniques.

In this article, we have presented an algorithmic framework that breaks the collaborative prediction process into components, and we provide empirical results regarding variants of each component, as well as present new algorithms that enhance the accuracy of predictions.

The empirical conclusions in this article are drawn from an analysis of historical data collected from an operational movie prediction site. The data is representative of a large set of rating-based systems, where the domain of predictions is high volume targeted entertainment with a generally high level of quality control. Domains of this criteria include movies, videos, books, and music. There is reason to believe that these results are generalizable to other prediction domains, but we do not yet have empirical results to prove it. Our algorithmic

recommendations are certainly a good place to start when exploring a new and different prediction domain.

We have made new contributions in each of the three steps of the neighborhood-based prediction algorithm. We have shown that Spearman correlation is not an appropriate replacement for the Pearson correlation coefficient, taking longer to compute and producing less accuracy. We demonstrated that incorporating significance weighting by devaluing correlations that are based on small numbers of co-rated items provided a significant gain in prediction accuracy. While we hypothesized that decreasing the contributions of items which had a low rating variance across all users would increase predictions accuracy, it proved false, with variance weights decreasing the prediction accuracy. Best-*n* neighbors proved to be the best approach to selecting neighbors to form a neighborhood, while correlation-weight thresholding did not provide any clear value. When it comes to combining ratings to form a prediction, deviation-from-mean averaging was shown to increase prediction accuracy significantly over a normal weighted average, while z-score averaging provided no significant improvements over deviation-from-mean. Furthermore, weighting the contributions of neighbors by their correlation with the user did increase the accuracy of the end predictions.

For those who are considering using a neighborhood-based prediction algorithm to perform automated collaborative filtering, we have the following recommendations: Use Pearson correlation as a similarity measure—it remains the most accurate technique for computing similarity. If your rating scale is binary or unary, you may have to consider a different approach—see Breese et al. (1998) for more information. It is important to use a significance weight to devalue correlates with small numbers of co-rated items as it will often give you a larger gain in accuracy than your choice of similarity algorithm. Finally, users will rate on slightly different scales, so use the deviation-from-mean approach to normalization. These recommendations are summarized in Table 8.

In the progress of examining personalized algorithms, we also discovered a much more accurate non-personalized average algorithm. Automated collaborative filtering systems use non-personalized average algorithms to provide predictions when not enough is known

Table 8. A tabulation of recommendations based on the results presented in this chapter.

	Recommended	Not recommended
Similarity weighting (Section 5.1)	Pearson correlation	Spearman, entropy, vector similarity, mean-squared difference
Significance weighting (Section 5.2)	Yes	
Selecting neighbors (Section 6)	Set max number of neighbors (potentially in the range of 20–60 nbors)	Weight thresholding
Rating normalization (Section 7.1)	Deviation-from-mean or z-score	No normalization
Weighting neighbor contributions (Section 7.2)	Yes	

Table 9. Two more accurate overall-average prediction algorithms. Both the average-zscore and average-deviation-from mean algorithms are significantly more accurate than the base average-rating algorithm.

Average algorithm	MAE	t-value (compared to average rating)
Average rating	0.8354	–
Average z-score	**0.7900**	**9.77 ($p = 0$)**
Average deviation from mean	**0.7835**	**12.2 ($p = 0$)**

about the user to provide a personalized prediction. The normal approach is to compute the average rating of the item being predicted over all users (Eq. (8)).

$$p_{a,i} = \frac{\sum_{u=1}^{n} r_{u,i}}{n} \tag{8}$$

However, we have found that computing a deviation-from-mean average over all users (Eq. (9)) results in a much more accurate non-personalized prediction as is demonstrated in Table 9.

$$p_{a,i} = r_a + \frac{\sum_{u=1}^{n} (r_{u,i} - \bar{r}_u)}{n} \tag{9}$$

Notes

1. The data are publicly available—http://www.grouplens.org/data.
2. The MovieLens user interface centers on providing "top-n" recommendation lists. Users then have an opportunity to rate the movies that are recommended. As a result, there are many more ratings for generally good items than ratings for generally bad items.
3. The *max_nbors* parameter stands for "maximum number of neighbors used" and is discussed in detail in Section 6.2.
4. "Rate-rerate reliability" describes the likelihood that a user, having given a rating on an item, will give the same rating when asked again at a later date. Larger scales lead to lower reliability. For example, imagine we had users rate movies on a 100 point scale, and a user rated "Star Wars" 83. If we asked that same user a month later what he rated the item, it is unlikely that he would say exactly 83 again. However, if the scale is only five points, there is a good chance that he will give the same rating both times.

References

Aggarwal CC, Wolf JL, Wu K-L and Yu PS (1999) Horting hatches an Egg: A new graph-theoretic approach to collaborative filtering. In: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.

Basu C, Hirsh H and Cohen WW (1998) Recommendation as classification: Using social and content-based information in recommendation. In: Rich C and Mostow J, eds., Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98). AAAI Press, Menlo Park, CA, pp. 714–720.

Billsus D and Pazzani MJ (1998) Learning collaborative information filters. In: Rich C and Mostow J, eds., Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98). AAAI Press, Menlo Park, CA, pp. 46–53.

Breese JS, Heckerman D and Kadie C (1998) Empirical analysis of predictive algorithms for collaborative filtering. In: Cooper GF and Moral S, eds., Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98). Morgan Kaufmann, San Francisco, pp. 43–52.

Condliff MK, Lewis DD, Madigan D and Posse C (1999) Bayesian mixed-effect models for recommender systems. In: 1999 SIGIR Workshop on Recommender Systems.

Delgado J and Ishii N (1999) Memory-based weighted majority prediction for recommender systems. In: 1999 SIGIR Workshop on Recommender Systems.

Goldberg K, Roeder T, Guptra D and Perkins C (2001) Eigentaste: A constant-time collaborative filtering algorithm. Information Retrieval, 4(2):133–151.

Good N, Schafer JB, Konstan JA, Borchers A, Sarwar BM, Herlocker JL and Riedl J (1999) Combining collaborative filtering with personal agents for better recommendations. In: Hendler J and Subramanian D, eds., Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99). AAAI Press, Menlo Park, pp. 439–446.

Herlocker JL (2000) Understanding and improving automated collaborative filtering systems. Ph.D. Dissertation. University of Minnesota, 9-30-2000.

Herlocker JL, Konstan JA, Borchers A and Riedl J (1999) An algorithmic framework for performing collaborative filtering. In: Hearst MA, Gey F and Tong R, eds., Proceedings of the 22nd International Conference on Research and Development in Information Retrieval (SIGIR '99). ACM Press, New York, pp. 230–237.

Hill W, Stead L, Rosenstein M and Furnas GW (1995) Recommending and evaluating choices in a virtual community of use. In: Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems. ACM Press, New York, pp. 194–201.

Hofmann T and Puzicha J (1999) Latent class models for collaborative filtering. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pp. 688–693.

Pennock DM, Horvitz E and Giles CL (2000a) Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In: Kautz H and Porter B, eds., Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000). AAAI Press, Menlo Park, CA, pp. 729–734.

Pennock DM, Horvitz E, Lawrence S and Giles CL (2000b) Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In: Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000). Morgan Kaufmann, San Francisco, pp. 473–480.

Press WH, Flannery BP, Teukolsky SA and Yan T (1986) Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, New York, NY.

Resnick P, Iacovou N, Suchak M, Bergstrom P and Riedl J (1994) GroupLens: An open architecture for collaborative filtering of netnews. In: Furuta R and Neuwirth C, eds., Proceedings of the 1994 Conference on Computer Supported Collaborative Work. ACM Press, New York, pp. 175–186.

Sarwar BM, Karypis G, Konstan JA and Riedl J (2000a) Analysis of recommendation algorithms for e-Commerce. In: Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00). ACM Press, New York, pp. 285–295.

Sarwar BM, Karypis G, Konstan JA and Riedl J (2000b) Application of dimensionality reduction in recommender system—A case study. In: ACM WebKDD 2000 Web Mining for E-Commerce Workshop.

Sarwar BM, Konstan JA, Borchers A, Herlocker JL, Miller BN, and Riedl J (1998) Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In: Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work (CSCW '98). ACM Press, New York.

Shardanand U and Maes P (1995) Social information filtering: Algorithms for automating "word of mouth." In: Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems. ACM, New York, pp. 210–217.