Item-based Collaborative Filtering Recommendation Algorithms

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl

GroupLens Research Group/Army HPC Research Center
Department of Computer Science and Engineering
University of Minnesota, Minneapolis, MN 55455

{sarwar, karypis, konstan, riedl}@cs.umn.edu

Appears in WWW10, May 1-5, 2001, Hong Kong.

Abstract

Recommender systems apply knowledge discovery techniques to the problem of making personalized recommendations for information, products or services during a live interaction. These systems, especially the k-nearest neighbor collaborative filtering based ones, are achieving widespread success on the Web. The tremendous growth in the amount of available information and the number of visitors to Web sites in recent years poses some key challenges for recommender systems. These are: producing high quality recommendations, performing many recommendations per second for millions of users and items and achieving high coverage in the face of data sparsity. In traditional collaborative filtering systems the amount of work increases with the number of participants in the system. New recommender system technologies are needed that can quickly produce high quality recommendations, even for very large-scale problems. To address these issues we have explored item-based collaborative filtering techniques. Item-based techniques first analyze the user-item matrix to identify relationships between different items, and then use these relationships to indirectly compute recommendations for users.

In this paper we analyze different item-based recommendation generation algorithms. We look into different techniques for computing item-item similarities (e.g., item-item correlation vs. cosine similarities between item vectors) and different techniques for obtaining recommendations from them (e.g., weighted sum vs. regression model). Finally, we experimentally evaluate our results and compare them to the basic k-nearest neighbor approach. Our experiments suggest that item-based algorithms provide dramatically better performance than user-based algorithms, while at the same time providing better quality than the best available user-based algorithms.

1 Introduction

The amount of information in the world is increasing far more quickly than our ability to process it. All of us have known the feeling of being overwhelmed by the number of new books, journal articles, and conference proceedings coming out each year. Technology has dramatically reduced the barriers to publishing and distributing information. Now it is time to create the technologies that can help us sift through all the available information to find that which is most valuable to us.

One of the most promising such technologies is *collaborative filtering* [19, 27, 14, 16]. Collaborative filtering works by building a database of preferences for items by users. A new user, Neo, is matched against the database to discover *neighbors*, which are other users who have historically had similar taste to Neo. Items that the neighbors like are then

recommended to Neo, as he will probably also like them. Collaborative filtering has been very successful in both research and practice, and in both information filtering applications and E-commerce applications. However, there remain important research questions in overcoming two fundamental challenges for collaborative filtering recommender systems.

The first challenge is to improve the scalability of the collaborative filtering algorithms. These algorithms are able to search tens of thousands of potential neighbors in real-time, but the demands of modern systems are to search tens of millions of potential neighbors. Further, existing algorithms have performance problems with individual users for whom the site has large amounts of information. For instance, if a site is using browsing patterns as indications of content preference, it may have thousands of data points for its most frequent visitors. These "long user rows" slow down the number of neighbors that can be searched per second, further reducing scalability.

The second challenge is to improve the quality of the recommendations for the users. Users need recommendations they can trust to help them find items they will like. Users will "vote with their feet" by refusing to use recommender systems that are not consistently accurate for them.

In some ways these two challenges are in conflict, since the less time an algorithm spends searching for neighbors, the more scalable it will be, and the worse its quality. For this reason, it is important to treat the two challenges simultaneously so the solutions discovered are both useful and practical.

In this paper, we address these issues of recommender systems by applying a different approach–item-based algorithms. The bottleneck in conventional collaborative filtering algorithms is the search for neighbors among a large user population of potential neighbors [12]. Item-based algorithms avoid this bottleneck by exploring the relationships between items first, rather than the relationships between users. Recommendations for users are computed by finding items that are similar to other items the user has liked. Because the relationships between items are relatively static, item-based algorithms may be able to provide the same quality as the user-based algorithms with less online computation.

1.1 Related Work

In this section we briefly present some of the research literature related to collaborative filtering, recommender systems, data mining and personalization.

Tapestry [10] is one of the earliest implementations of collaborative filtering-based recommender systems. This system relied on the explicit opinions of people from a close-knit community, such as an office workgroup. However, recommender system for large communities cannot depend on each person knowing the others. Later, several ratings-based automated recommender systems were developed. The GroupLens research system [19, 16] provides a pseudonymous collaborative filtering solution for Usenet news and movies. Ringo [27] and Video Recommender [14] are email and web-based systems that generate recommendations on music and movies respectively. A special issue of Communications of the ACM [20] presents a number of different recommender systems.

Other technologies have also been applied to recommender systems, including Bayesian networks, clustering, and Horting. Bayesian networks create a model based on a training set with a decision tree at each node and edges representing user information. The model can be built off-line over a matter of hours or days. The resulting model is very small, very fast, and essentially as accurate as nearest neighbor methods [6]. Bayesian networks may prove practical for environments in which knowledge of user preferences changes slowly with respect to the time needed to build the model but are not suitable for environments in which user preference models must be updated rapidly or frequently.

Clustering techniques work by identifying groups of users who appear to have similar preferences. Once the clusters are created, predictions for an individual can be made by averaging the opinions of the other users in that cluster. Some clustering techniques represent each users with partial participation in several clusters. The prediction is then an average across the clusters, weighted by degree of participation. Clustering techniques usually produce less-personal recommendations than other methods, and in some cases, the clusters have worse accuracy than nearest neighbor algorithms [6]. Once the clustering is complete, however, performance can be very good, since the size of the group that must be analyzed is much smaller. Clustering techniques can also be applied as a "first step" for

shrinking the candidate set in a nearest neighbor algorithm or for distributing nearest-neighbor computation across several recommender engines. While dividing the population into clusters may hurt the accuracy or recommendations to users near the fringes of their assigned cluster, pre-clustering may be a worthwhile trade-off between accuracy and throughput.

Horting is a graph-based technique in which nodes are users, and edges between nodes indicate degree of similarity between two users [1]. Predictions are produced by walking the graph to nearby nodes and combining the opinions of the nearby users. Horting differs from nearest neighbor as the graph may be walked through other users who have not rated the item in question, thus exploring transitive relationships that nearest neighbor algorithms do not consider. In one study using synthetic data, Horting produced better predictions than a nearest neighbor algorithm [1].

Schafer et al., [26] present a detailed taxonomy and examples of recommender systems used in E-commerce and how they can provide one-to-one personalization and at the same can capture customer loyalty. Although these systems have been successful in the past, their widespread use has exposed some of their limitations such as the problems of sparsity in the data set, problems associated with high dimensionality and so on. Sparsity problem in recommender system has been addressed in [23, 11]. The problems associated with high dimensionality in recommender systems have been discussed in [4], and application of dimensionality reduction techniques to address these issues has been investigated in [24].

Our work explores the extent to which item-based recommenders, a new class of recommender algorithms, are able to solve these problems.

1.2 Contributions

This paper has three primary research contributions:

1. Analysis of the item-based prediction algorithms and identification of different ways to implement its subtasks.
2. Formulation of a precomputed model of item similarity to increase the online scalability of item-based recommendations.
3. An experimental comparison of the quality of several different item-based algorithms to the classic user-based (nearest neighbor) algorithms.

1.3 Organization

The rest of the paper is organized as follows. The next section provides a brief background in collaborative filtering algorithms. We first formally describe the collaborative filtering process and then discuss its two variants memory-based and model-based approaches. We then present some challenges associated with the memory-based approach. In section 3, we present the item-based approach and describe different sub-tasks of the algorithm in detail. Section 4 describes our experimental work. It provides details of our data sets, evaluation metrics, methodology and results of different experiments and discussion of the results. The final section provides some concluding remarks and directions for future research.

2 Collaborative Filtering Based Recommender Systems

Recommender systems systems apply data analysis techniques to the problem of helping users find the items they would like to purchase at E-Commerce sites by producing a predicted likeliness score or a list of *top–N* recommended items for a given user. Item recommendations can be made using different methods. Recommendations can be based on demographics of the users, overall top selling items, or past buying habit of users as a predictor of future items. Collaborative Filtering (CF) [19, 27] is the most successful recommendation technique to date. The basic idea of CF-based algorithms is to provide item recommendations or predictions based on the opinions of other like-minded users. The opinions of users can be obtained *explicitly* from the users or by using some *implicit* measures.

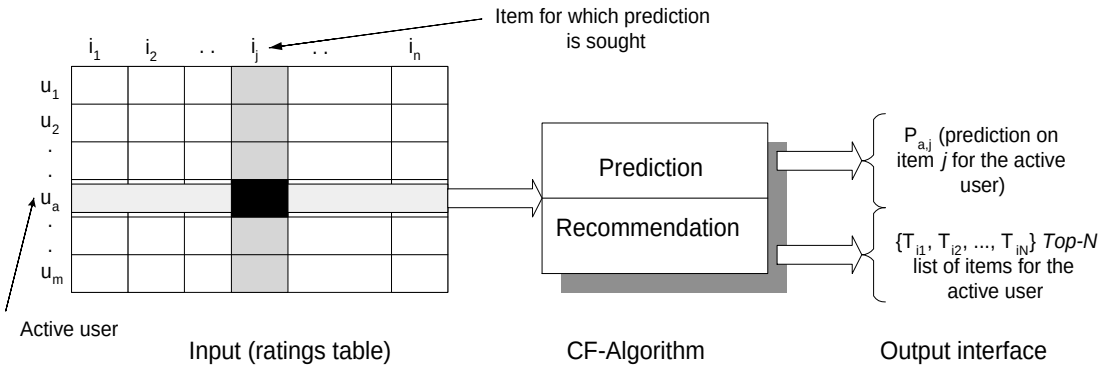

Figure 1: The Collaborative Filtering Process.

2.0.1 Overview of the Collaborative Filtering Process

The goal of a collaborative filtering algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users. In a typical CF scenario, there is a list of m users $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and a list of n items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$. Each user u_i has a list of items I_{u_i}, which the user has expressed his/her opinions about. Opinions can be explicitly given by the user as a *rating score*, generally within a certain numerical scale, or can be implicitly derived from purchase records, by analyzing timing logs, by mining web hyperlinks and so on [28, 16]. Note that $I_{u_i} \subseteq \mathcal{I}$ and it is possible for I_{u_i} to be a *null-set*. There exists a distinguished user $u_a \in \mathcal{U}$ called the *active user* for whom the task of a collaborative filtering algorithm is to find an item likeliness that can be of two forms.

- **Prediction** is a numerical value, $P_{a,j}$, expressing the predicted likeliness of item $i_j \notin I_{u_a}$ for the active user u_a. This predicted value is within the same scale (e.g., from 1 to 5) as the opinion values provided by u_a.
- **Recommendation** is a list of N items, $I_r \subset \mathcal{I}$, that the active user will like the most. Note that the recommended list must be on items not already purchased by the active user, i.e., $I_r \cap I_{u_a} = \Phi$. This interface of CF algorithms is also known as *Top-N recommendation*.

Figure 1 shows the schematic diagram of the collaborative filtering process. CF algorithms represent the entire $m \times n$ user-item data as a ratings matrix, $\mathcal{A}$. Each entry $a_{i,j}$ in $\mathcal{A}$ represent the preference score (ratings) of the ith user on the jth item. Each individual ratings is within a numerical scale and it can as well be 0 indicating that the user has not yet rated that item. Researchers have devised a number of collaborative filtering algorithms that can be divided into two main categories—*Memory-based (user-based)* and *Model-based (item-based)* algorithms [6]. In this section we provide a detailed analysis of CF-based recommender system algorithms.

Memory-based Collaborative Filtering Algorithms Memory-based algorithms utilize the entire user-item data-base to generate a prediction. These systems employ statistical techniques to find a set of users, known as *neighbors*, that have a history of agreeing with the target user (i.e., they either rate different items similarly or they tend to buy similar set of items). Once a neighborhood of users is formed, these systems use different algorithms to combine the preferences of neighbors to produce a prediction or *top-N* recommendation for the active user. The techniques, also known as *nearest-neighbor* or user-based collaborative filtering are more popular and widely used in practice.

Model-based Collaborative Filtering Algorithms Mo-del-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings on other items. The model building process is performed by different *machine learning* algorithms such as **Bayesian network, clustering,** and **rule-based** approaches. The Bayesian network model [6] formulates a

probabilistic model for collaborative filtering problem. Clustering model treats collaborative filtering as a classification problem [2, 6, 29] and works by clustering similar users in same class and estimating the probability that a particular user is in a particular class C, and from there computes the conditional probability of ratings. The rule-based approach applies association rule discovery algorithms to find association between co-purchased items and then generates item recommendation based on the strength of the association between items [25].

2.0.2 Challenges of User-based Collaborative Filtering Algorithms

User-based collaborative filtering systems have been very successful in past, but their widespread use has revealed some potential challenges such as:

- **Sparsity.** In practice, many commercial recommender systems are used to evaluate large item sets (e.g., Amazon.com recommends books and CDnow.com recommends music albums). In these systems, even active users may have purchased well under 1% of the items (1% of 2 million books is $20,000$ books). Accordingly, a recommender system based on nearest neighbor algorithms may be unable to make any item recommendations for a particular user. As a result the accuracy of recommendations may be poor.

- **Scalability.** Nearest neighbor algorithms require computation that grows with both the number of users and the number of items. With millions of users and items, a typical web-based recommender system running existing algorithms will suffer serious scalability problems.

The weakness of nearest neighbor algorithm for large, sparse databases led us to explore alternative recommender system algorithms. Our first approach attempted to bridge the sparsity by incorporating semi-intelligent filtering agents into the system [23, 11]. These agents evaluated and rated each item using syntactic features. By providing a dense ratings set, they helped alleviate coverage and improved quality. The filtering agent solution, however, did not address the fundamental problem of poor relationships among like-minded but sparse-rating users. To explore that we took an algorithmic approach and used Latent Semantic Indexing (LSI) to capture the similarity between users and items in a reduced dimensional space [24, 25]. In this paper we look into another technique, the model-based approach, in addressing these challenges, especially the scalability challenge. The main idea here is to analyze the user-item representation matrix to identify relations between different items and then to use these relations to compute the prediction score for a given user-item pair. The intuition behind this approach is that a user would be interested in purchasing items that are similar to the items the user liked earlier and would tend to avoid items that are similar to the items the user didn't like earlier. These techniques don't require to identify the neighborhood of similar users when a recommendation is requested, as a result they tend to produce much faster recommendations. A number of different schemes have been proposed to compute the association between items ranging from probabilistic approach [6] to more traditional item-item correlations [15, 13]. We present a detailed analysis of our approach in the next section.

3 Item-based Collaborative Filt-ering Algorithm

In this section we study a class of item-based recommendation algorithms for producing predictions to users. Unlike the user-based collaborative filtering algorithm discussed in Section 2 the item-based approach looks into the set of items the target user has rated and computes how similar they are to the target item i and then selects k most similar items $\{i_1, i_2, \ldots, i_k\}$. At the same time their corresponding similarities $\{s_{i1}, s_{i2}, \ldots, s_{ik}\}$ are also computed. Once the most similar items are found, the prediction is then computed by taking a weighted average of the target user's ratings on these similar items. We describe these two aspects namely, the similarity computation and the prediction generation in details here.

3.1 Item Similarity Computation

One critical step in the item-based collaborative filtering algorithm is to compute the similarity between items and then to select the most similar items. The basic idea in similarity computation between two items i and j is to first isolate

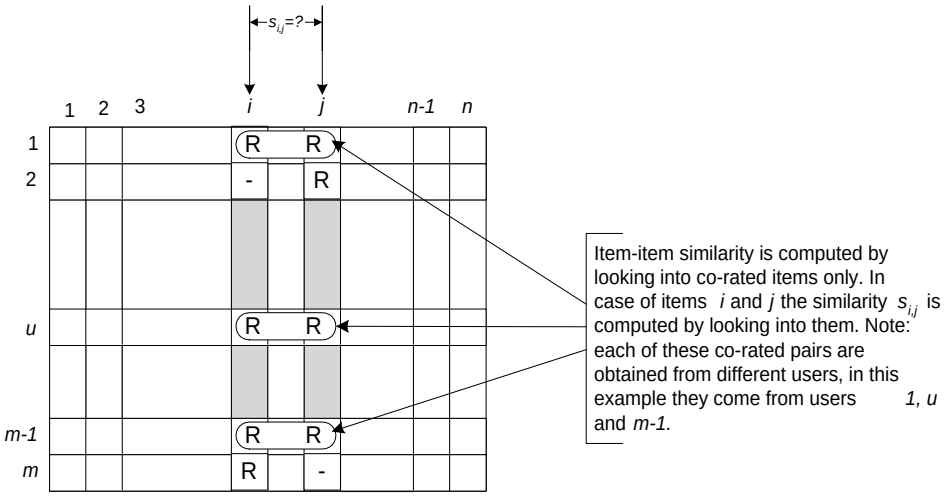

Figure 2: Isolation of the co-rated items and similarity computation

the users who have rated both of these items and then to apply a similarity computation technique to determine the similarity $s_{i,j}$. Figure 2 illustrates this process, here the matrix rows represent users and the columns represent items.

There are a number of different ways to compute the similarity between items. Here we present three such methods. These are cosine-based similarity, correlation-based similarity and adjusted-cosine similarity.

3.1.1 Cosine-based Similarity

In this case, two items are thought of as two vectors in the m dimensional user-space. The similarity between them is measured by computing the cosine of the angle between these two vectors. Formally, in the $m \times n$ ratings matrix in Figure 2, similarity between items i and j, denoted by $sim(i, j)$ is given by

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

where "$\cdot$" denotes the dot-product of the two vectors.

3.1.2 Correlation-based Similarity

In this case, similarity between two items i and j is measured by computing the *Pearson-r* correlation $corr_{i,j}$. To make the correlation computation accurate we must first isolate the co-rated cases (i.e., cases where the users rated both i and j) as shown in Figure 2. Let the set of users who both rated i and j are denoted by U then the correlation similarity is given by

$$sim(i, j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}}.$$

Here $R_{u,i}$ denotes the rating of user u on item i, $\bar{R}_i$ is the average rating of the i-th item.

3.1.3 Adjusted Cosine Similarity

One fundamental difference between the similarity computation in user-based CF and item-based CF is that in case of user-based CF the similarity is computed along the rows of the matrix but in case of the item-based CF the similarity is computed along the columns i.e., each pair in the co-rated set corresponds to a different user (Figure 2). Computing similarity using basic cosine measure in item-based case has one important drawback–the difference in rating scale between different users are not taken into account. The adjusted cosine similarity offsets this drawback by subtracting

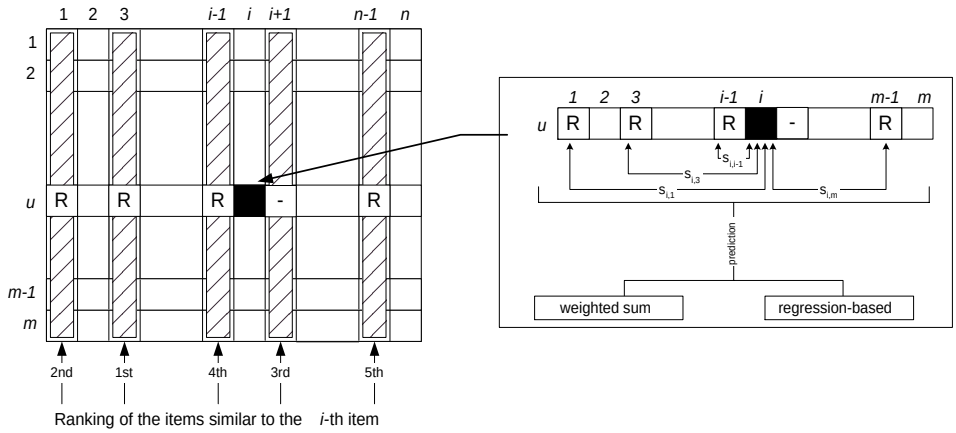

Figure 3: Item-based collaborative filtering algorithm. The prediction generation process is illustrated for 5 neighbors

the corresponding user average from each co-rated pair. Formally, the similarity between items i and j using this scheme is given by

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R_u})^2}}.$$

Here $\bar{R_u}$ is the average of the u-th user's ratings.

3.2 Prediction Computation

The most important step in a collaborative filtering system is to generate the output interface in terms of prediction. Once we isolate the set of most similar items based on the similarity measures, the next step is to look into the target users ratings and use a technique to obtain predictions. Here we consider two such techniques.

3.2.1 Weighted Sum

As the name implies, this method computes the prediction on an item i for a user u by computing the sum of the ratings given by the user on the items similar to i. Each ratings is weighted by the corresponding similarity $s_{i,j}$ between items i and j. Formally, using the notion shown in Figure 3 we can denote the prediction $P_{u,i}$ as

$$P_{u,i} = \frac{\sum_{\text{all similar items, N}}(s_{i,N} * R_{u,N})}{\sum_{\text{all similar items, N}}(|s_{i,N}|)}$$

Basically, this approach tries to capture how the active user rates the similar items. The weighted sum is scaled by the sum of the similarity terms to make sure the prediction is within the predefined range.

3.2.2 Regression

This approach is similar to the weighted sum method but instead of directly using the ratings of similar items it uses an approximation of the ratings based on regression model. In practice, the similarities computed using cosine or correlation measures may be misleading in the sense that two rating vectors may be distant (in Euclidean sense) yet may have very high similarity. In that case using the raw ratings of the "so called" similar item may result in poor prediction. The basic idea is to use the same formula as the weighted sum technique, but instead of using the similar item N's "raw" ratings values $R_{u,N}$'s, this model uses their approximated values $R'_{u,N}$ based on a linear regression model. If we denote the respective vectors of the target item i and the similar item N by R_i and R_N the linear regression model can be expressed as

$$\bar{R'_N} = \alpha \bar{R_i} + \beta + \epsilon$$

The regression model parameters α and β are determined by going over both of the rating vectors. ϵ is the error of the regression model.

3.3 Performance Implications

The largest E-Commerce sites operate at a scale that stresses the direct implementation of collaborative filtering. In neighborhood-based CF systems, the neighborhood formation process, especially the user-user similarity computation step turns out to be the performance bottleneck, which in turn can make the whole process unsuitable for real-time recommendation generation. One way of ensuring high scalability is to use a model-based approach. Model-based systems have the potential to contribute to recommender systems to operate at a high scale. The main idea here to isolate the neighborhood generation and prediction generation steps.

In this paper, we present a model-based approach to precompute item-item similarity scores. The similarity computation scheme is still correlation-based but the computation is performed on the item space. In a typical E-Commerce scenario, we usually have a set of item that is static compared to the number of users that changes most often. The static nature of items leads us to the idea of precomputing the item similarities. One possible way of precomputing the item similarities is to compute all-to-all similarity and then performing a quick table look-up to retrieve the required similarity values. This method, although saves time, requires an $\mathcal{O}(n^2)$ space for n items.

The fact that we only need a small fraction of similar items to compute predictions leads us to an alternate model-based scheme. In this scheme, we retain only a small number of similar items. For each item j we compute the k most similar items, where $k \ll n$ and record these item numbers and their similarities with j. We term k as the *model size*. Based on this model building step, our prediction generation algorithm works as follows. For generating predictions for a user u on item i, our algorithm first retrieves the precomputed k most similar items corresponding to the target item i. Then it looks how many of those k items were purchased by the user u, based on this intersection then the prediction is computed using basic item-based collaborative filtering algorithm.

We observe a quality-performance trade-off here: to ensure good quality we must have a large model size, which leads to the performance problems discussed above. In one extreme, we can have a model size of n, which will ensure exact same quality as original scheme but will have high space complexity. However, our model building step ensures that we retain the most similar items. While generating predictions, these items contribute the most to the prediction scores. Accordingly, we hypothesize that this model-based approach will provide reasonably good prediction quality with even a small model size and hence provide a good performance. We experimentally validate our hypothesis later in this paper. In particular, we experiment with the model size by varying the number of similar items to be stored. Then we perform experiments to compute prediction and response-time to determine the impact of the model size on quality and performance of the whole system.

4 Experimental Evaluation

4.1 Data set

We used experimental data from our research website to evaluate different variants of item-based recommendation algorithms.

Movie data We used data from our MovieLens recommender system, MovieLens is a web-based research recommender system that debuted in Fall 1997. Each week hundreds of users visit MovieLens to rate and receive recommendations for movies. The site now has over 43000 users who have expressed opinions on 3500+ different movies. We randomly selected enough users to obtain $100,000$ ratings from the database (we only considered users that had rated 20 or more movies). We divided the database into a training set and a test set. For this purpose, we introduced a variable that determines what percentage of data is used as training and test sets, we call this variable x. A value of $x = 0.8$ would indicate 80% of the data was used as training set and 20% of the data was used as test set. The data set was converted into a user-item matrix A that had 943 rows (i.e., 943 users) and 1682 columns (i.e., 1682 movies that were rated by at least one of the users). For our experiments, we also take another factor into consideration, *sparsity*

level of data sets. For the data matrix R This is defined as $1 - \frac{\text{nonzero entries}}{\text{total entries}}$. The sparsity level of the Movie data set is, therefore, $1 - \frac{100,000}{943 \times 1682}$, which is 0.9369. Throughout the paper we term this data set as *ML*.

4.2 Evaluation Metrics

Recommender systems research has used several types of measures for evaluating the quality of a recommender system. They can be mainly categorized into two classes:

- *Statistical accuracy metrics* evaluate the accuracy of a system by comparing the numerical recommendation scores against the actual user ratings for the user-item pairs in the test dataset. *Mean Absolute Error* (MAE) between ratings and predictions is a widely used metric. MAE is a measure of the deviation of recommendations from their true user-specified values. For each ratings-prediction pair $< p_i, q_i >$ this metric treats the absolute error between them i.e., $|p_i - q_i|$ equally. The MAE is computed by first summing these absolute errors of the N corresponding ratings-prediction pairs and then computing the average. Formally,

$$MAE = \frac{\sum_{i=1}^{N} |p_i - q_i|}{N}$$

  The lower the MAE, the more accurately the recommendation engine predicts user ratings. *Root Mean Squared Error* (RMSE), and *Correlation* are also used as statistical accuracy metric

- *Decision support accuracy metrics* evaluate how effective a prediction engine is at helping a user select high-quality items from the set of all items. These metrics assume the prediction process as a binary operation-either items are predicted (good) or not (bad). With this observation, whether a item has a prediction score of 1.5 or 2.5 on a five-point scale is irrelevant if the user only chooses to consider predictions of 4 or higher. The most commonly used decision support accuracy metrics are *reversal rate*, *weighted errors* and *ROC sensitivity* [23].

We used MAE as our choice of evaluation metric to report prediction experiments because it is most commonly used and easiest to interpret directly. In our previous experiments [23] we have seen that MAE and ROC provide the same ordering of different experimental schemes in terms of prediction quality.

4.2.1 Experimental Procedure

Experimental steps We started our experiments by first dividing the data set into a training and a test portion. Before starting full experimental evaluation of different algorithms we determined the sensitivity of different parameters to different algorithms and from the sensitivity plots we fixed the optimum values of these parameters and used them for the rest of the experiments. To determine the parameter sensitivity, we work only with the training data and further subdivide it into a training and test portion and carried on our experiments on them. For conducted a 10-fold cross validation of our experiments by randomly choosing different training and test sets each time and taking the average of the MAE values.

Benchmark user-based system To compare the performance of item-based prediction we also entered the training ratings set into a collaborative filtering recommendation engine that employs the Pearson nearest neighbor algorithm (user-user). For this purpose we implemented a flexible prediction engine that implements user-based CF algorithms. We tuned the algorithm to use the best published Pearson nearest neighbor algorithm and configured it to deliver the highest quality prediction without concern for performance (i.e., it considered every possible neighbor to form optimal neighborhoods).

Experimental platform All our experiments were implemented using *C* and compiled using optimization flag -06. We ran all our experiments on a linux based PC with Intel Pentium III processor having a speed of 600 MHz and 2GB of RAM.

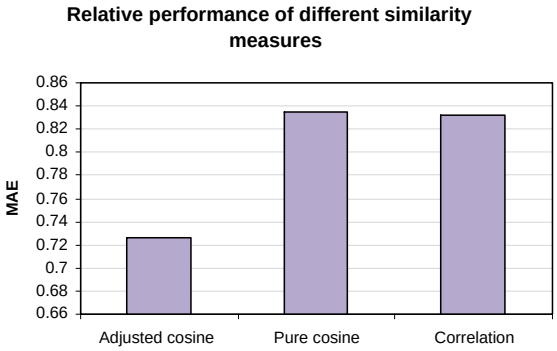

Figure 4: Impact of the similarity computation measure on item-based collaborative filtering algorithm.

4.3 Experimental Results

In this section we present our experimental results of applying item-based collaborative filtering techniques for generating predictions. Our results are mainly divided into two parts–quality results and performance results. In assessing the quality of recommendations, we first determined the sensitivity of some parameters before running the main experiment. These parameters include the neighborhood size, the value of the training/test ratio x, and effects of different similarity measures. For determining the sensitivity of various parameters, we focused only on the training data set and further divided it into a *training* and a *test* portion and used them to learn the parameters.

4.3.1 Effect of Similarity Algorithms

We implemented three different similarity algorithms basic cosine, adjusted cosine and correlation as described in Section 3.1 and tested them on our data sets. For each similarity algorithms, we implemented the algorithm to compute the neighborhood and used weighted sum algorithm to generate the prediction. We ran these experiments on our training data and used test set to compute Mean Absolute Error (MAE). Figure 4 shows the experimental results. It can be observed from the results that offsetting the user-average for cosine similarity computation has clear advantage, as the MAE is significantly lower in this case. Hence, we select the adjusted cosine similarity for the rest of our experiments.

4.3.2 Sensitivity of Training/Test Ratio

To determine the sensitivity of density of the data set we carried out an experiment where we varied the value of x from 0.2 to 0.9 in an increment of 0.1. For each of these training/test ratio values we ran our experiments using the two prediction generation techniques–basic weighted sum and regression based approach. Our results are shown in Figure 5. We observe that the quality of prediction increase as we increase x. The regression-based approach shows better results than the basic scheme for low values of x but as we increase x the quality tends to fall below the basic scheme. From the curves, we select $x = 0.8$ as an optimum value for our subsequent experiments.

4.3.3 Experiments with neighborhood size

The size of the neighborhood has significant impact on the prediction quality [12]. To determine the sensitivity of this parameter, we performed an experiment where we varied the number of neighbors to be used and computed MAE. Our results are shown in Figure 5. We can observe that the size of neighborhood does affect the quality of prediction. But the two methods show different types of sensitivity. The basic item-item algorithm improves as we increase the neighborhood size from 10 to 30, after that the rate of increase diminishes and the curve tends to be flat. On the other hand, the regression-based algorithm shows decrease in prediction quality with increased number of neighbors. Considering both trends we select 30 as our optimal choice of neighborhood size.

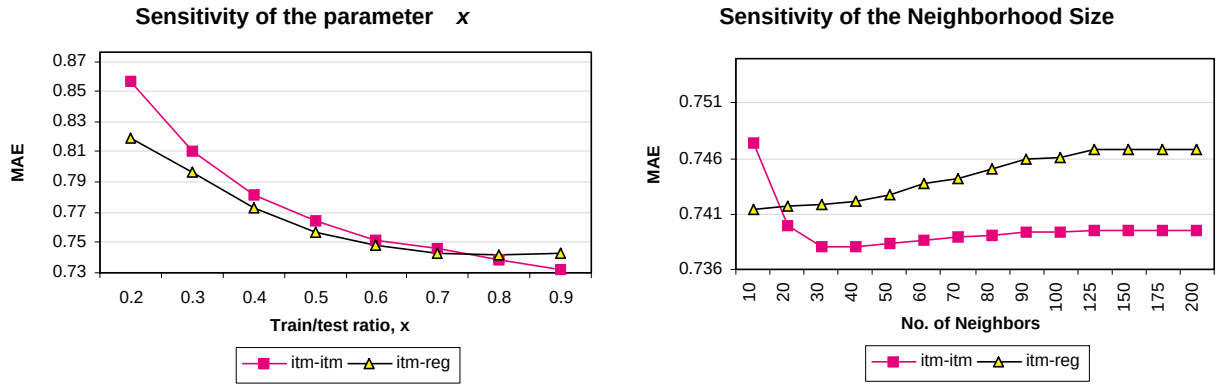

Figure 5: Sensitivity of the parameter x on the neighborhood size

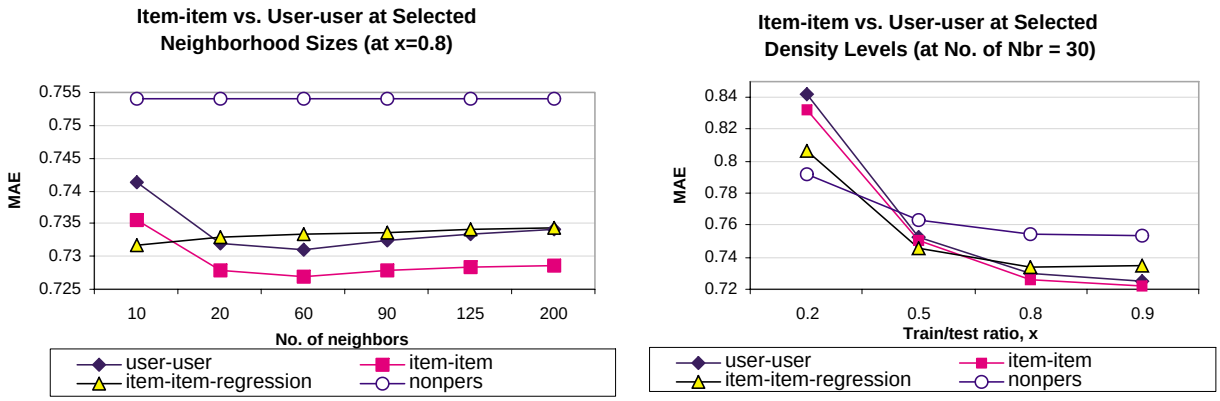

Figure 6: Comparison of prediction quality of *item-item* and *user-user* collaborative filtering algorithms. We compare prediction qualities at $x = 0.2, 0.5, 0.8$ and 0.9.

4.3.4 Quality Experiments

Once we obtain the optimal values of the parameters, we compare both of our item-based approaches with the benchmark user-based algorithm. We present the results in Figure 6. It can be observed from the charts that the basic item-item algorithm out performs the user based algorithm at all values of x (neighborhood size = 30 and all values of neighborhood size ($x = 0.8$). For example, at $x = 0.5$ user-user scheme has an MAE of 0.755 and item-item scheme shows an MAE of 0.749. Similarly at a neighborhood size of 60 user-user and item-item schemes show MAE of 0.732 and 0.726 respectively. The regression-based algorithm, however, shows interesting behavior. At low values of x and at low neighborhood size it out performs the other two algorithms but as the density of the data set is increased or as we add more neighbors it performs worse, even compared to the user-based algorithm. We also compared our algorithms against the naive nonpersonalized algorithm described in [12].

We draw two conclusions from these results. First, item-based algorithms provide better quality than the user-based algorithms at all sparsity levels. Second, regression-based algorithms perform better with very sparse data set, but as we add more data the quality goes down. We believe this happens as the regression model suffers from data overfiting at high density levels.

4.3.5 Performance Results

Having clearly established the superior quality of item-based algorithms over the user-based ones, we focus on the scalability challenges. As we discussed earlier, item-based similarity is more static and allows us to precompute the item neighborhood. This precomputation of the model has certain performance benefits. To make the system even

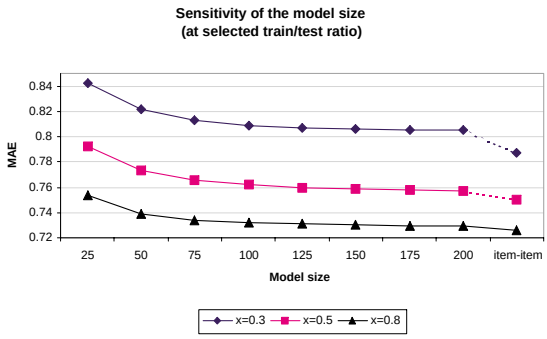

Figure 7: Sensitivity of the model size on item-based collaborative filtering algorithm

more scalable we looked into the sensitivity of the model size and then looked into the impact of model size on the response time and throughput.

4.4 Sensitivity of the Model size

To experimentally determine the impact of the model size on the quality of the prediction, we selectively varied the number of items to be used for similarity computation from 25 to 200 in an increment of 25. A model size of l means that we only considered l best similarity values for model building and later on used k of them for the prediction generation process, where $k < l$. Using the training data set we precomputed the item similarity using different model sizes and then used only the weighted sum prediction generation technique to provide the predictions. We then used the test data set to compute MAE and plotted the values. To compare with the full model size (i.e., model size = no. of items) we also ran the same test considering all similarity values and picked best k for prediction generation. We repeated the entire process for three different x values (training/test ratios). Figure 7 shows the plots at different x values. It can be observed from the plots that the MAE values get better as we increase the model size and the improvements are drastic at the beginning, but gradually slows down as we increase the model size. The most important observation from these plots is the high accuracy can be achieved using only a fraction of items. For example, at $x = 0.3$ the full item-item scheme provided an MAE of 0.7873, but using a model size of only 25, we were able to achieve an MAE value of 0.842. At $x = 0.8$ these numbers are even more appealing–for the full item-item we had an MAE of 0.726 but using a model size of only 25 we were able to obtain an MAE of 0.754, and using a model size of 50 the MAE was 0.738. In other words, at $x = 0.8$ we were within 96% and 98.3% of the full item-item scheme's accuracy using only 1.9% and 3% of the items respectively!

This model size sensitivity has important performance implications. It appears from the plots that it is useful to precompute the item similarities using only a fraction of items and yet possible to obtain good prediction quality.

4.4.1 Impact of the model size on run-time and throughput

Given the quality of prediction is reasonably good with small model size, we focus on the run-time and throughput of the system. We recorded the time required to generate predictions for the entire test set and plotted them in a chart with varying model size. We plotted the run time at different x values. Figure 8 shows the plot. Note here that at $x = 0.25$ the whole system has to make prediction for $25,000$ test cases. From the plot we observe a substantial difference in the run-time between the small model size and the full item-item prediction case. For $x = 0.25$ the run-time is 2.002 seconds for a model size of 200 as opposed to 14.11 for the basic item-item case. This difference is even more prominent with $x = 0.8$ where a model size of 200 requires only 1.292 seconds and the basic item-item case requires 36.34 seconds.

These run-time numbers may be misleading as we computed them for different training/test ratios where the workload size i.e., number of predictions to be generated is different (recall that at $x = 0.3$ our algorithm uses $30,000$ ratings as training data and uses the rest of $70,000$ ratings as test data to compare predictions generated by the system to the actual ratings). To make the numbers comparable we compute the throughput (predictions generated per second)

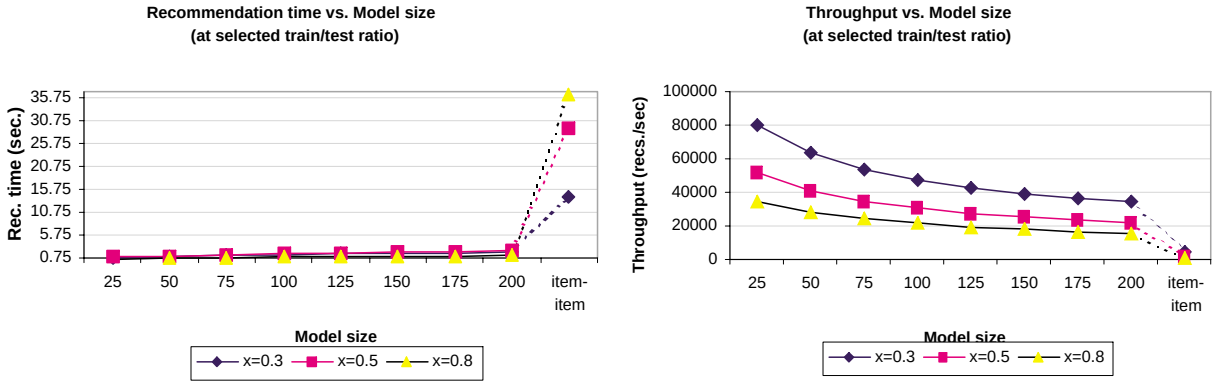

Figure 8: Recommendation time and throughput comparison between model-based scheme and full item-item scheme. The comparisons are shown at three different x values.

for the model based and basic item-item schemes. Figure 8 charts these results. We see that for $x = 0.3$ and at a model size of 100 the system generates $70,000$ ratings in 1.487 seconds producing a throughput rate of $47,361$ where as the basic item-item scheme produced a throughput of 4961 only. At $x = 0.8$ these two numbers are $21,505$ and 550 respectively.

4.5 Discussion

From the experimental evaluation of the item-item collaborative filtering scheme we make some important observations. First, the item-item scheme provides better quality of predictions than the use-user (k-nearest neighbor) scheme. The improvement in quality is consistent over different neighborhood size and training/test ratio. However, the improvement is not significantly large. The second observation is that the item neighborhood is fairly static, which can be potentially pre-computed, which results in very high online performance. Furthermore, due to the model-based approach, it is possible to retain only a small subset of items and produce reasonably good prediction quality. Our experimental results support that claim. Therefore, the item-item scheme is capable in addressing the two most important challenges of recommender systems for E-Commerce–quality of prediction and high performance.

5 Conclusion

Recommender systems are a powerful new technology for extracting additional value for a business from its user databases. These systems help users find items they want to buy from a business. Recommender systems benefit users by enabling them to find items they like. Conversely, they help the business by generating more sales. Recommender systems are rapidly becoming a crucial tool in E-commerce on the Web. Recommender systems are being stressed by the huge volume of user data in existing corporate databases, and will be stressed even more by the increasing volume of user data available on the Web. New technologies are needed that can dramatically improve the scalability of recommender systems.

In this paper we presented and experimentally evaluated a new algorithm for CF-based recommender systems. Our results show that item-based techniques hold the promise of allowing CF-based algorithms to scale to large data sets and at the same time produce high-quality recommendations.

6 Acknowledgments

Funding for this research was provided in part by the National Science Foundation under grants IIS 9613960, IIS 9734442, and IIS 9978717 with additional funding by Net Perceptions Inc. This work was also supported by NSF CCR-9972519, EIA-9986042, ACI-9982274 by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008.

Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. We like to thank anonymous reviewers for their valuable comments.

References

[1] Aggarwal, C. C., Wolf, J. L., Wu K., and Yu, P. S. (1999). Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering. In *Proceedings of the ACM KDD'99 Conference*. San Diego, CA. pp. 201-212.

[2] Basu, C., Hirsh, H., and Cohen, W. (1998). Recommendation as Classification: Using Social and Content-based Information in Recommendation. In *Recommender System Workshop '98*. pp. 11-15.

[3] Berry, M. W., Dumais, S. T., and O'Brian, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review, 37(4)*, pp. 573-595.

[4] Billsus, D., and Pazzani, M. J. (1998). Learning Collaborative Information Filters. In *Proceedings of ICML '98*. pp. 46-53.

[5] Brachman, R., J., Khabaza, T., Kloesgen, W., Piatetsky-Shapiro, G., and Simoudis, E. 1996. Mining Business Databases. *Communications of the ACM, 39(11)*, pp. 42-48, November.

[6] Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 43-52.

[7] Cureton, E. E., and D'Agostino, R. B. (1983). Factor Analysis: An Applied Approach. *Lawrence Erlbaum associates pubs.* Hillsdale, NJ.

[8] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science, 41(6)*, pp. 391-407.

[9] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., Eds. (1996). Advances in Knowledge Discovery and Data Mining. *AAAI press/MIT press*.

[10] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*. December.

[11] Good, N., Schafer, B., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. (1999). Combining Collaborative Filtering With Personal Agents for Better Recommendations. In *Proceedings of the AAAI-'99 conference*, pp 439-446.

[12] Herlocker, J., Konstan, J., Borchers, A., and Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of ACM SIGIR'99*. ACM press.

[13] Herlocker, J. (2000). Understanding and Improving Automated Collaborative Filtering Systems. *Ph.D. Thesis, Computer Science Dept., University of Minnesota.*

[14] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and Evaluating Choices in a Virtual Community of Use. In *Proceedings of CHI '95*.

[15] Karypis, G. (2000). Evaluation of Item-Based Top-N Recommendation Algorithms. *Technical Report CS-TR-00-46*, Computer Science Dept., University of Minnesota.

[16] Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. (1997). GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40(3), pp. 77-87.

[17] Ling, C. X., and Li C. (1998). Data Mining for Direct Marketing: Problems and Solutions. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pp. 73-79.

[18] Peppers, D., and Rogers, M. (1997). The One to One Future : Building Relationships One Customer at a Time. *Bantam Doubleday Dell Publishing*.

[19] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of CSCW '94*, Chapel Hill, NC.

[20] Resnick, P., and Varian, H. R. (1997). Recommender Systems. Special issue of Communications of the ACM. 40(3).

[21] Reichheld, F. R., and Sasser Jr., W. (1990). Zero Defections: Quality Comes to Services. *Harvard Business School Review*, 1990(5): pp. 105-111.

[22] Reichheld, F. R. (1993). Loyalty-Based Management. *Harvard Business School Review*, 1993(2): pp. 64-73.

[23] Sarwar, B., M., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., and Riedl, J. (1998). Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System. In *Proceedings of CSCW '98*, Seattle, WA.

[24] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Application of Dimensionality Reduction in Recommender System–A Case Study. In *ACM WebKDD 2000 Workshop*.

[25] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce. In *Proceedings of the ACM EC'00 Conference*. Minneapolis, MN. pp. 158-167

[26] Schafer, J. B., Konstan, J., and Riedl, J. (1999). Recommender Systems in E-Commerce. In *Proceedings of ACM E-Commerce 1999 conference*.

[27] Shardanand, U., and Maes, P. (1995). Social Information Filtering: Algorithms for Automating 'Word of Mouth'. In *Proceedings of CHI '95*. Denver, CO.

[28] Terveen, L., Hill, W., Amento, B., McDonald, D., and Creter, J. (1997). PHOAKS: A System for Sharing Recommendations. *Communications of the ACM, 40(3)*. pp. 59-62.

[29] Ungar, L. H., and Foster, D. P. (1998) Clustering Methods for Collaborative Filtering. In *Workshop on Recommender Systems at the 15th National Conference on Artificial Intelligence*.