Item-Based Top-*N* Recommendation Algorithms

MUKUND DESHPANDE and GEORGE KARYPIS
University of Minnesota

The explosive growth of the world-wide-web and the emergence of e-commerce has led to the development of *recommender systems*—a personalized information filtering technology used to identify a set of items that will be of interest to a certain user. User-based collaborative filtering is the most successful technology for building recommender systems to date and is extensively used in many commercial recommender systems. Unfortunately, the computational complexity of these methods grows linearly with the number of customers, which in typical commercial applications can be several millions. To address these scalability concerns model-based recommendation techniques have been developed. These techniques analyze the user–item matrix to discover relations between the different items and use these relations to compute the list of recommendations.

In this article, we present one such class of model-based recommendation algorithms that first determines the similarities between the various items and then uses them to identify the set of items to be recommended. The key steps in this class of algorithms are (i) the method used to compute the similarity between the items, and (ii) the method used to combine these similarities in order to compute the similarity between a *basket* of items and a candidate recommender item. Our experimental evaluation on eight real datasets shows that these *item-based* algorithms are up to two orders of magnitude faster than the traditional user-neighborhood based recommender systems and provide recommendations with comparable or better quality.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*data mining*; H.3.3 [**Information Storage and Retrieval**]: Search and Retrieval—*information filtering*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: world wide web, e-commerce, predicting user behavior.

This work was supported in part by National Science Foundation (NSF) grants EIA-9986042, ACI-9982274, and ACI-0133464; NASA NCC 21231, the Digital Technology Center at the University of Minnesota; and by the Army High-Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014.
The content of this article does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred.
Contact author: G. Karypis, Dept. of Computer Science & Engineering, 4-192 EE/CS Building, 200 Union Street SE, Minneapolis, MN 55455; email: karypis@cs.umn.edu.

1. INTRODUCTION

The explosive growth of the world-wide-web and the emergence of e-commerce has led to the development of *recommender systems* [Resnick and Varian 1997]. Recommender systems are personalized information filtering technology used to either predict whether a particular user will like a particular item (*prediction problem*) or to identify a set of N items that will be of interest to a certain user (top-N *recommendation problem*). In recent years, recommender systems have been used in a number of different applications [Shardanand and Maes 1995; Hill et al. 1995; Konstan et al. 1997; Terveen et al. 1997; Schafer et al. 1999; Kitts et al. 2000; Mobasher et al. 2000; Beeferman and Berger 2000] such as recommending products a customer will most likely buy; movies, TV programs, or music a user will find enjoyable; identifying web pages that will be of interest; or even suggesting alternate ways of searching for information. An excellent survey of different recommender systems for various applications can be found in Schafer et al. [1999] and Resnick and Varian [1997].

Over the years, various approaches for building recommender systems have been developed that utilize either demographic, content, or historical information [Hill et al. 1995; Balabanovic and Shoham 1997; Basu et al. 1998; Shardanand and Maes 1995; Terveen et al. 1997; Konstan et al. 1997]. Among them, collaborative filtering (CF), which relies on historical information, is probably the most successful and widely used technique for building recommender systems [Resnick et al. 1994; Konstan et al. 1997]. The term, *collaborative filtering* was first coined in Goldberg et al. [1992], where it was used to describe an e-mail filtering system called Tapestry, which was designed to filter e-mails received from mailing lists and newsgroup postings. In this system, each user could write a comment (annotation) about each e-mail message and share these annotations with a group of users. A user could then filter these e-mail messages by writing queries on these annotations. Though Tapestry allowed an individual user to benefit from annotations made by other users, the system required an individual user to write complicated queries. The first system to generate automated recommendations was the GroupLens system [Resnick et al. 1994; Konstan et al. 1997], which provided users with personalized recommendations on Usenet postings. The recommendations for each individual were obtained by identifying a neighborhood of similar users and recommending the articles that this group of users found useful.

Two approaches have been developed for building CF-based *top-N* recommender systems. The first approach, referred to as *user-based* [Shardanand and Maes 1995; Konstan et al. 1997; Breese et al. 1998; Resnick et al. 1994; Herlocker et al. 1999; Sarwar et al. 2000], relies on the fact that each person belongs in a larger group of similarly behaving individuals. As a result, items (e.g., products, movies, books, etc.) frequently purchased/liked by the various members of the group can be used to form a basis for recommended items. The second approach, known as *model-based* [Shardanand and Maes 1995; Billsus and Pazzani 1998; Breese et al. 1998; Aggarwal et al. 1999; Kitts et al. 2000], analyzes historical information to identify relations between different items such that the purchase of an item (or a set of items) often leads to the purchase

of another item (or a set of items), and then use these relations to determine the recommended items. Model-based schemes, by using precomputed models, produce recommendations very quickly but tend to require a significant amount of time to build these models. Furthermore, these recommendations are generally of lower-quality than those produced by user-based schemes. In contrast, user-based schemes tend to produce systems that lead to higher-quality recommendations but suffer serious scalability problems as the complexity of computing each recommendation grows linearly with the number of users and items.

The focus of this article is on a particular class of model-based *top-N* recommendation algorithms that build the recommendation model by analyzing the similarities between the various items and then use these similar items to identify the set of items to be recommended. These algorithms, referred to in this article as *item-based* top-*N recommendation algorithms*, have been used in various forms since the early days of CF-based recommender systems [Shardanand and Maes 1995; Kitts et al. 2000] and were shown to be computationally scalable (both in terms of model construction and model application) but tended to produce lower-quality recommendations when compared to user-based schemes.

The contributions of this article are two-fold. First, we present a detailed study of the two key steps that affect the performance of item-based *top-N* recommendation algorithms, which are (i) the method used to compute the similarity between the items and (ii) the method used to combine these similarities in order to compute the similarity between a *basket* of items and a candidate recommender item. For the first step, we study two different methods of computing the item-to-item similarity. One models the items as vectors in the user space, and uses the *cosine* function to measure the similarity, whereas the other computes the item-to-item similarities using a technique based on the conditional probability between two items. This conditional probability technique is extended so that it can differentiate between users with varying amounts of historical information as well as between frequently and infrequently purchased items. For the second step, we present a method for combining these similarities that accounts for item-neighborhoods of different density that can incorrectly bias the overall recommendation.

The second contribution is the extension of these item-based schemes to higher-order models, which obtain the final recommendations by exploiting relations between sets of items. We present a class of *interpolated higher-order item-based* top-*N recommendation* algorithms that construct a recommendation model by first determining the various itemset-item similarities and then combining them to determine the similarity between a user's *basket* and a candidate recommender item.

We present a detailed experimental evaluation of these algorithms and study the performance implications of the various parameters on two classes of datasets. The first class consists of eight real datasets arising in various applications, whereas the second class consists of 36 datasets that were synthetically generated by the widely-used synthetic transaction dataset generator provided by the IBM Quest group [Agrawal and Srikant 1994]. Our experiments show that the item-based algorithm when combined with the conditional probability-based similarity method produce higher-quality recommendations

than the user-based scheme on both real and synthetic datasets. Moreover, the higher-order schemes lead to additional improvements when the density of the datasets increases and when the users have many items in common. Furthermore, our computational complexity evaluation shows that the item-to-item based algorithms are up to two orders of magnitude faster than the traditional user-based algorithms. Some of the results in this paper were previously presented in Karypis [2001].

The paper is organized as follows: Section 2 provides the definitions and notations that will be used throughout the paper. Section 3 presents a brief survey of the related research on collaborative filtering-based recommender algorithms. Sections 4 and 5 describe the various phases and algorithms used in our first- and higher-order item-based *top-N* recommendation system. Section 6 provides an experimental evaluation of the various parameters of the proposed algorithms, and compares the proposed algorithms against user-based ones. Finally, Section 7 provides some concluding remarks.

2. DEFINITIONS AND NOTATIONS

Throughout the article, we will use the symbols n and m to denote the number of distinct users and the number of distinct items in a particular dataset, respectively. We will use the symbol N to denote the number of recommendations that needs to be computed for a particular user. In presenting the various algorithms we will assume that the underlying application domain is that of commercial retailing and we will use the terms *customers* and *products* as synonyms to users and items, respectively. We will use the term *dataset* to denote the set of transactions about the items that have been purchased by the various users. We will represent each dataset by an $n \times m$ binary matrix R that will be referred to as the *user–item matrix*, such that $R_{i,j}$ is one if the ith customer has purchased the jth item, and zero otherwise. We will refer to the user for which we want to compute the *top-N* recommendations as the *active user*, and to the set of items that the user has already purchased as the user's *basket*. Finally, the *top-N* recommendation problem is formally defined as follows:

Definition 2.1 (*top-N Recommendation Problem*). Given a user–item matrix R and a set of items U that have been purchased by a user, identify an ordered set of items X such that $|X| \leq N$ and $X \cap U = \emptyset$.

3. RELATED RESEARCH

User-based collaborative filtering is the most successful technology for building recommender systems to date and is extensively used in many commercial recommender systems. In general, user-based systems compute the *top-N* recommended items for a particular user by following a three-step approach [Shardanand and Maes 1995; Konstan et al. 1997; Sarwar et al. 2000]. In the first step, they identify the k users in the database that are the most similar to the active user. During the second step, they compute the union of the items purchased by these users and associate a weight with each item based on its importance in the set. In the third and final step, from this union they select and recommend the N items that have the highest weight and have not

already been purchased by the active user. Within this three-step framework, the method used to determine the k most similar users and the scheme used to determine the importance of the different items play the most critical role in the overall performance of the algorithm. Commonly, the similarity between the users is computed by treating them as vectors in the item-space and measuring their similarity via the cosine or correlation coefficient functions [Breese et al. 1998; Sarwar et al. 2000], whereas the importance of each item is determined by how frequently it was purchased by the k most similar users. However, alternate approaches for both of these steps have been explored and shown to lead to somewhat better results. A detailed survey of different user-based algorithms and a comparison of their performance can be found in Breese et al. [1998], Herlocker et al. [1999], and Sarwar et al. [2000].

Despite the popularity of user-based recommender systems, they have a number of limitations related to scalability and real-time performance. The computational complexity of these methods grows linearly with the number of customers, which in typical commercial applications can grow to be several millions. Furthermore, even though the user–item matrix is sparse, the user-to-user similarity matrix is quite dense. This is because even a few frequently purchased items can lead to dense user-to-user similarities. Moreover, real-time *top-N* recommendations based on the current basket of items, utilized by many e-commerce sites, cannot take advantage of pre-computed user-to-user similarities. Finally, even though the throughput of user-based recommendation algorithms can be increased by increasing the number of servers running the recommendation algorithm, they cannot decrease the latency of each *top-N* recommendation, which is critical for near real-time performance. One way of reducing the complexity of the nearest-neighbor computations is to cluster the users and then to either limit the nearest-neighbor search among the users that belong to the nearest cluster, or use the cluster centroids to derive the recommendations [Ungar and Foster 1998; Mobasher et al. 2000]. These approaches, though they can significantly speed up the recommendation algorithm, tend to decrease the quality of the recommendations.

To address the scalability concerns of user-based recommendation algorithms a variety of model-based recommendation techniques were developed. Billsus and Pazzani [1998] developed a model-based recommender system by treating the *top-N* recommendation problem as a classification problem, in which the goal was to classify the items purchased by an individual user into two classes: like and dislike. A classification model based on neural networks was built for each individual user where the items purchased by the user were thought of as the examples and the users as the attributes. A singular value decomposition of the user–item matrix reduced the dimensionality of the problem. The prediction on an item was computed by constructing an example for that item and feeding it to the classifier. The authors reported considerable improvements over the traditional user-based algorithms. Though this approach is quite powerful it requires building and maintaining a neural network model for each individual user in the database, which is not scalable to large databases. Breese et al. [1998] presented two model-based algorithms for computing both predictions and *top-N* recommendations. The first algorithm follows a probabilistic

approach in which the users are clustered and the conditional probability distribution of different items in the cluster is estimated. The probability that the active user belongs to a particular cluster given the basket of items is then estimated from the clustering solution and the probability distribution of items in the cluster. The clustering solution for this technique is computed using the expectation maximization (EM) principle. The second algorithm is based on Bayesian network models where each item in the database is modeled as a node having states corresponding to the rating of that item. The learning problem consists of building a network on these nodes such that each node has a set of parent nodes that are the best predictors for the child's rating. They presented a detailed comparison of these two model-based approaches with the user-based approach and showed that Bayesian networks model outperformed the clustering model as well as the user-based scheme. Heckerman et al. [2000] proposed a recommendation algorithm based on dependency networks instead of Bayesian networks. Though the accuracy of dependency networks is inferior to Bayesian networks they are more efficient to learn and have smaller memory requirements. Aggarwal et al. [1999] presented a graph-based recommendation algorithm where the users are represented as the nodes in a graph and the edges between the nodes indicate the degree of similarity between the users. The recommendations for a user are computed by traversing nearby nodes in this graph. The graph representation of the model allows it to capture transitive relations which cannot be captured by nearest neighbor algorithms and the authors reported better performance than the user-based schemes.

A number of different model-based approaches have been developed that use item-to-item similarities as well as association rules. Shardanand and Maes [1995] developed an item-based prediction algorithm within the context of the Ringo music recommendation system, referred to as *artist-artist*, that determines whether or not a user will like a particular artist by computing its similarity to the artists that the user has liked/disliked in the past. This similarity was computed using the Pearson correlation function. Sarwar et al. [2001] further studied this paradigm for computing predictions and they evaluated various methods for computing the similarity as well as approaches to limit the set of item-to-item similarities that need to be considered. The authors reported considerable improvements in performance over the user-based algorithm. Mobasher et al. [2000] presented an algorithm for recommending additional webpages to be visited by a user based on association rules. In this approach, the historical information about users and their web-access patterns were mined using a frequent itemset discovery algorithm and were used to generate a set of high confidence association rules. The recommendations were computed as the union of the consequent of the rules that were supported by the pages visited by the user. Lin et al. [2000] used a similar approach but they developed an algorithm that is guaranteed to find association rules for all the items in the database. Finally, within the context of using association rules to derive *top-N* recommendations, Demiriz [2001] studied the problem of how to weight the different rules that are supported by the active user. He presented a method that computes the similarity between a rule and the active user's basket as the product of the confidence of the rule and the Euclidean distance

between items in the antecedent of the association rule and the items in the user's basket. He compared this approach both with the item-based scheme described in Section 4 (based on our preliminary work presented in Karypis [2001]) and the dependency network-based algorithm [Heckerman et al. 2000]. His experiments showed that the proposed association rule-based scheme is superior to dependency networks but inferior to the item-based schemes.

4. ITEM-BASED *TOP-N* RECOMMENDATION ALGORITHMS

In this section, we study a class of model-based *top-N* recommendation algorithms that use item-to-item similarities to compute the relations between the different items. The primary motivation behind these algorithms is the fact that a customer is more likely to purchase items that are similar to the items that he/she has already purchased in the past; thus, by analyzing historical purchasing information (as represented in the user–item matrix) we can automatically identify these sets of similar items and use them to form the *top-N* recommendations. These algorithms are similar in spirit to previously developed item-based schemes [Shardanand and Maes 1995; Kitts et al. 2000] but differ in a number of key aspects related to how the similarity between the different items is computed and how these similarities are combined to derive the final recommendations.

At a high-level, these algorithms consist of two distinct components. The first component builds a model that captures the relations between the different items, whereas the second component applies this precomputed model to derive the *top-N* recommendations for an active user. The details on these components are presented in the remainder of this section.

4.1 Building the Model

The model used by the item-based *top-N* recommendation algorithm is constructed using the algorithm shown in Algorithm 4.1. The input to this algorithm is the $n \times m$ user–item matrix R and a parameter k that specifies the number of item-to-item similarities that will be stored for each item. The output is the model itself, which is represented by an $m \times m$ matrix $\mathcal{M}$ such that the jth column stores the k most similar items to item j. In particular, if $\mathcal{M}_{i,j} > 0$, then the ith item is among the k most similar items of j and the value of $\mathcal{M}_{i,j}$ indicates the degree of similarity between items j and i.

Algorithm 4.1: BUILDMODEL (R, k)

for $j \rightarrow 1$ **to** m

do
- **for** $i \rightarrow 1$ **to** m
- **do**
  - **if** $i \neq j$
    - **then** $\mathcal{M}_{i,j} \rightarrow \text{sim}(R_{*,j}, R_{*,i})$ (1)
    - **else** $\mathcal{M}_{i,j} \rightarrow 0$
- **for** $i \rightarrow 1$ **to** m
- **do**
  - **if** $\mathcal{M}_{i,j} \neq$ among the k largest values in $\mathrm{M}_{*,j}$ (2)
    - **then** $\mathcal{M}_{i,j} \rightarrow 0$

return $(\mathcal{M})$

The parameterization of $\mathcal{M}$ on k was motivated due to performance considerations and its choice represents a performance-quality trade-off. By using a small value of k, we can ensure that $\mathcal{M}$ is very sparse and thus can be stored in main memory even in collaborative filtering environments and applications in which m is very large. However, if k is too small, then the resulting model will contain limited information from which to build the recommendations, and thus it can potentially lead to lower quality. Fortunately, as our experimental evaluation will illustrate (Section 6.2.1), reasonably small values of k ($10 \leq k \leq 30$) lead to good results and higher values lead to either a very small or no improvement.

The actual algorithm for constructing $\mathcal{M}$ is quite simple. For each item j, the algorithm computes the similarity between j and the other items and stores the results in the jth column of $\mathcal{M}$ (line 1). Once these similarities have been computed, it then proceeds to zero-out all the entries in the jth column of $\mathcal{M}$ that contain smaller values than the k largest similarity values in that column. The resulting matrix $\mathcal{M}$ that contains at most k nonzero entries per column becomes the final model of the item-based algorithm. Note that by construction (line 2) the algorithm ensures that a particular item will not contain itself as one of its k most similar items. This is done to ensure that an item does not contribute towards recommending itself. Such recommendations are of little value because we require the recommended items to be different from the items in the active user's basket.

4.1.1 *Measuring the Similarity between Items.* The properties of the model $\mathcal{M}$ and consequently the effectiveness of the overall recommendation algorithm depend on the method used to compute the similarity between the various items (line 1 in Algorithm 4.1). In general, the similarity between two items i and j should be high if there are lot of customers that have purchased both of them, and it should be low if there are few such customers. There are also two somewhat less obvious aspects that we need to consider. The first has to do with whether or not we should be discriminating between customers that purchase few items and customers that purchase many items. For example, consider two customers C_1 and C_2, both of whom have purchased items i and j, but C_1 has purchased 5 additional items whereas C_2 has purchased 50 additional items. Should the fact that both of them purchased i and j contribute equally while determining the similarity between this pair of items? There may be cases in which the copurchasing information derived from customers that have bought fewer items is a more reliable indicator for the similarity of two copurchased items than the information derived from customers that tend to buy a large number of items. This is because the first group represents consumers that are focused in certain product areas. As our experiments in Section 6.2.1 will show, this is often the case, and being able to take it into account improves the overall *top-N* recommendation performance.

The second aspect that we need to consider has to do with whether or not the similarity between a pair of items should be symmetric (i.e., $\text{sim}(i, j) = \text{sim}(j, i)$) or not (i.e., $\text{sim}(i, j) \neq \text{sim}(j, i)$). This question usually arises when we need to compute the similarity between pairs of items that are purchased

at substantially different frequencies. For example, consider two items i and j such that i has been purchased significantly more frequently than j. Due to that frequency difference, the number of times that i and j are purchased together will be much smaller than the number of times that i is purchased alone. What should the similarity between i and j be? From i's point of view, its similarity to j is low, because only a small fraction of its occurrences will co-occur with j. However, from j's point of view, its similarity to i may be high, because a large fraction of its occurrences may co-occur with i. Thus, if we use an asymmetric similarity function, we will have that $\text{sim}(i, j) < \text{sim}(j, i)$. However, if we use a symmetric function the similarity between j and i (from j's point of view) will generally end up being smaller than it would be in the asymmetric case, as it would have to account for i's higher frequency. Each one of these two approaches has its advantages. A symmetric similarity function will tend to eliminate recommendations of very frequent items (that to a large extent are obvious), as these items will tend to be recommended only if other frequently purchased items are in the current basket. However, in datasets that have no items that are frequently purchased by the majority of the customers, a symmetric similarity function will unnecessarily penalize the recommendation of items whose frequency is relatively higher than the items that have been currently purchased by the active user.

In this study, we use two different similarity functions that are derived from the vector-space model and probabilistic methods, respectively. The key difference between them is that the first leads to similarities that are symmetric, whereas the second leads to similarities that are asymmetric. Furthermore, we have modified both similarity functions so that they can weight the customers differently based on how many products they have purchased. The details of these functions and their modifications are provided in the rest of this section.

4.1.1.1 *Cosine-Based Similarity.* One way of computing the similarity between two items is to treat each item as a vector in the space of customers and use the *cosine* between these vectors as a measure of similarity. Formally, if R is the $n \times m$ user–item matrix, then the similarity between two items i and j is defined as the cosine of the n dimensional vectors corresponding to the ith and jth column of matrix R. The cosine between these vectors is given by

$$\text{sim}(i, j) = \cos(\vec{R}_{*,i}, \vec{R}_{*,j}) = \frac{\vec{R}_{*,j} \cdot \vec{R}_{*,i}}{\|\vec{R}_{*,i}\|_2 \|\vec{R}_{*,j}\|_2}, \tag{1}$$

where '$\cdot$' denotes the vector dot-product operation. Note that since the cosine function measures the angle between the two vectors it is a symmetric similarity function. As a result, frequently purchased items will tend to be similar to other frequently purchased items and not to infrequently purchased items, and vice versa.

In its simplest form, the rows of R can correspond to the original binary purchase information, in which case, the cosine similarity function treats customers that purchase a small and a large number of items equally. However, each one of the rows can be scaled so that the resulting cosine-based similarity function will differentiate between these sets of customers. This can be done by

scaling each row to be of unit length (or any other norm). The effect of this scaling is that customers that have purchased fewer items will contribute a higher weight to the dot-product in Eq. (1) than customers that have purchased more items.

4.1.1.2 *Conditional Probability-Based Similarity.* An alternate way of computing the similarity between each pair of items i and j is to use a measure that is based on the conditional probability of purchasing one of the items given that the other has already been purchased. In particular, the conditional probability of purchasing j given that i has already been purchased $P(j|i)$ is nothing more than the number of customers that purchase both items i and j divided by the total number of customers that purchased i, that is,

$$P(j|i) = \frac{\text{Freq}(ij)}{\text{Freq}(i)},$$

where $\text{Freq}(X)$ is the number of customers that have purchased the items in the set X. Note that, in general, $P(j|i) \neq P(i|j)$ and using this as a measure of similarity leads to asymmetric relations.

As discussed earlier, one of the limitations of using an asymmetric similarity function is that each item i will tend to have high conditional probabilities with items that are being purchased frequently. This problem has been recognized by researchers in information retrieval and recommender systems [Salton 1989; Breese et al. 1998; Kitts et al. 2000; Chan 1999]. The problem can be corrected by dividing $P(j|i)$ with a quantity that depends on the occurrence frequency of item j. Two different methods have been proposed for achieving this. The first one, inspired from the inverse-document frequency scaling performed in information retrieval systems, multiplies $P(j|i)$ by $-\log_2(P(j))$ [Salton 1989], whereas the other one divides $P(j|i)$ by $P(j)$ [Kitts et al. 2000]. Note that this latter method leads to a symmetric similarity function. Our experiments have shown that this scaling greatly affects the performance of the recommender system and that the *optimal* scaling degree is problem dependent. For these reasons we use the following formula to compute the similarity between two items:

$$\text{sim}(i, j) = \frac{\text{Freq}(ij)}{\text{Freq}(i) \times (\text{Freq}(j))^{\alpha}}, \tag{2}$$

where α is a parameter that takes a value between 0 and 1. Note that, when $\alpha = 0$, Eq. (2) becomes identical to $P(j|i)$, whereas if $\alpha = 1$, it becomes similar (up to a scaling factor) to the formulation in which $P(j|i)$ is divided by $P(j)$.

The similarity function as defined in Eq. (2) does not discriminate between customers that have purchased different number of items. To achieve this discrimination and give higher weight to the customers that have purchased fewer items, we have extended the similarity measure of Eq. (2) in the following way: First we normalize each row of matrix R to be of unit length, and then we define the similarity between items i and j as:

$$\text{sim}(i, j) = \frac{\sum_{\forall q: R_{q,j} > 0} R_{q,j}}{\text{Freq}(i) \times (\text{Freq}(j))^{\alpha}}. \tag{3}$$

The only difference between Eq. (3) and Eq. (2) is that instead of using the co-occurrence frequency we use the sum of the corresponding nonzero entries of the jth column in the user–item matrix. Since the rows are normalized to be of unit length, customers that have purchased more items will tend to contribute less to the overall similarity. This gives emphasis to the purchasing decisions of the customers that have bought fewer items.

4.2 Applying the Model

The algorithm for applying the item-based model is shown in Algorithm 4.2. The input to this algorithm is the model $\mathcal{M}$, an $m \times 1$ vector U that stores the items that have already been purchased by the active user, and the number of items to be recommended (N). The active user's information in vector U is encoded by setting $U_i = 1$ if the user has purchased the ith item and zero otherwise. The output of the algorithm is an $m \times 1$ vector x whose nonzero entries correspond to the *top-N* items that were recommended. The weight of these nonzero entries represent a measure of the *recommendation strength* and the various recommendations can be ordered in non-increasing recommendation strength weight. In most cases x will have exactly N nonzero entries; however, the actual number of recommendations can be less than N as it depends on the value of k used to build $\mathcal{M}$ and the number of items that have already been purchased by the active user.

Algorithm 4.2: ApplyModel ($\mathcal{M}$, U, N)

$x \leftarrow \mathcal{M}\, U$ (1)

for $j \leftarrow 1$ **to** m (2)

do { **if** $U_i \neq 0$ **then** $x_i \leftarrow 0$

for $j \leftarrow 1$ **to** m (3)

do { **if** $x_i \neq$ among the N largest values in x **then** $x_i \leftarrow 0$

return (x)

The vector x is computed in three steps. First, the vector x is computed by multiplying $\mathcal{M}$ with U (line 1). Note that the nonzero entries of x correspond to the union of the k most similar items for each item that has already been purchased by the active user, and that the weight of these entries is nothing more than the sum of these similarities. Second, the entries of x that correspond to items that have already been purchased by the active user are set to zero (loop at line 2). Finally, in the third step, the algorithm sets to zero all the entries of x that have a value smaller than the N largest values of x (loop at line 3).

One potential drawback with Algorithm 4.2 is that the raw similarity between each item j and its k most similar items may be significantly different. That is, the item neighborhoods are of different density. This is especially true for items that are purchased somewhat infrequently, since a moderate overlap with other infrequently purchased items can lead to relatively high similarity values. Consequently, these items can exert strong influence in the selection of the *top-N* items, sometimes leading to wrong recommendations. For this

reason, instead of using the actual similarities computed by the various methods described in Section 4.1.1, for each item j we first normalize the similarities so that they add-up to one. That is, $\|\mathcal{M}_{*,j}\| = 1$, for $j = 1, \ldots, m$. As the experiments presented in Section 6 show, this always improves the *top-N* recommendation quality.

4.3 Computational Complexity

The computational complexity of the item-based *top-N* recommendation algorithm depends on the amount of time required to build the model $\mathcal{M}$ (i.e., for each item identify the other k most similar items), and the amount required to compute the recommendation using this model.

During the model building phase we need to compute the similarity between each item j and the other items in R and then select the k most similar items. The upper bound on the complexity of this step is $O(m^2 n)$ as we need to compute $m(m-1)$ similarities, each potentially requiring n operations. However, the actual complexity is significantly smaller because the resulting item-to-item similarity matrix is extremely sparse. In our datasets, the item-to-item similarity matrix was generally more than 99% sparse. The reason for these sparsity levels is that each customer purchases a relatively small number of items and the items they purchase tend to be clustered. Consequently, by using sparse data structures to store R and only computing the similarities between pairs of items that are purchased by at least one customer we can substantially reduce the computational complexity.

Finally, the time required to compute the *top-N* recommendations for an active user that has purchased q items is given by $O(kq)$ because we need to access the k most similar items for each one of the items that the user has already purchased and identify the overall N most similar items.

5. HIGHER-ORDER ITEM-BASED *TOP-N* RECOMMENDATION ALGORITHMS

Our discussion so far has focused on item-based *top-N* recommendation algorithms in which the recommendations were computed by taking into account relations between pairs of items, that is, for each item in the active user's basket, similar items were determined and these similar items were aggregated to obtained the desired *top-N* recommendations. These schemes effectively ignore the presence of other items in the active user's basket while computing the k most similar items for each item. Even though this allows such schemes to be computationally efficient, they can potentially lead to suboptimal recommendations when the joint distribution of a set of items is different from the distributions of the individual items in the set.

To solve this problem, we developed item-based *top-N* recommendation schemes that use all combinations of items (i.e., *itemsets*) up to a particular size l when determining the set of items to be recommended to a user. In this approach, during the model building phase, instead of only determining the k most similar items for each individual item, we do so for all possible itemsets up to a particular size l. During the model application time, we compute the *top-N* recommendations by combining these sets of k item-neighborhoods not

just for individual items, but for all itemsets up to size l that are present in the active user's basket.

We will refer to the parameter l as the *order* of the item-based model, and we will refer to this class of item-based *top-N* recommendation algorithms as the *interpolated higher-order models*. When $l = 1$, the above scheme becomes identical to the one described in Section 4 and for this reason we will sometimes refer to it as the first-order model. The name, *interpolated*, was motivated by the interpolated Markov models used in DNA sequence analysis [Delcher et al. 1998] and is used to indicate that the final predictions are computed by combining models that use itemsets of different size (i.e., the final solution is an *interpolation* of predictions computed by models that use one, two, ..., up to l itemsets).

The remainder of this section describes these higher-order item-based *top-N* recommendation algorithms in detail and discusses various issues associated with their efficient implementation.

5.1 Building the Model

During the model building phase we use the algorithm shown in Algorithm 5.1 to compute l different model matrices $\mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^l$ of size $m \times m, m \times m^2, \ldots, m \times m^l$, respectively. For a particular value of r, $\mathcal{M}^r$ is constructed by generating all possible combinations of r items $\{q_1, q_2, \ldots, q_r\}$ (loop at line 1), computing the similarity between these sets and all the other m items in the dataset (loop at line 2), and among them retaining only the k largest similarities in the corresponding columns of $\mathcal{M}^r$ (loop at line 3).

Algorithm 5.1: BUILDHIGHERORDERMODEL (R, l, k)

for $r \leftarrow 1$ **to** l

do
- **for** $j \leftarrow 1$ **to** m^r
  - **for** $i \leftarrow 1$ **to** r (1)
    - **do** $\{q_i \leftarrow ((j \bmod m^{r-i+1}) \operatorname{div} m^{r-i}) + 1$
  - **for** $i \leftarrow 1$ **to** m (2)
    - **do** **if** $i \notin \{q_1, \ldots, q_r\}$
      - **then** $\mathcal{M}^r_{i,j} \leftarrow \text{sim}(\{R_{*,q_1}, \ldots, R_{*,q_r}\}, R_{*,i})$
      - **else** $\mathcal{M}_{j,j} \leftarrow 0$
  - **for** $i \leftarrow 1$ **to** m (3)
    - **do** **if** $\mathcal{M}^r_{i,j} \neq$ among the k largest values in $\mathcal{M}^r_{*,j}$
      - **then** $\mathcal{M}^r_{i,j} \leftarrow 0$

return $\mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^l$

5.1.1 *Itemset-Item Similarity.* As in the first-order model, the key step in the proposed higher-order item-based *top-N* recommendation algorithm is the method used to determine the similarity between an itemset and the various items of the dataset. In our scheme these similarities are computed by using relative straightforward extensions of the cosine- and conditional probability-based approaches described in Section 4.1.1.

Specifically, the similarity between an itemset $\{q_1, q_2, \ldots, q_r\}$ and an other item u is computed as follows. In the case of the cosine-based approach, we first

construct an n-element vector $\vec{v}$ such that

$$\vec{v}(i) = \begin{cases} 0, & \text{if at least one of the } R_{i,q_j} = 0 \text{ for } j = 1, 2, \ldots, r, \\ \sum_{j=1}^{r} \frac{R_{i,q_j}}{\| R_{*,q_j} \|_2}, & \text{otherwise.} \end{cases}$$

Essentially, $\vec{v}$ is the sum of the individual unit-length normalized item-vectors of the items in the itemset with the added constraint that if a particular row of the matrix does not contain all r items it will be set to zero. Using this vector, the cosine similarity between the itemset represented by $\vec{v}$ and the item u is computed using Eq. (1).

In the case of the conditional probability-based approach, the similarity is computed using an approach similar to Eq. (3) as follows:

$$\text{sim}(\{q_1, q_2, \ldots, q_r\}, u) = \frac{\sum_{\forall i: R_{i,q_j} > 0, \text{for } j=1,2,\ldots,r} R_{i,q_1}}{\text{Freq}(\{q_1, q_2, \ldots, q_r\}) \times (\text{Freq}(u))^{\alpha}}. \tag{4}$$

Note that $\text{Freq}(\{q_1, q_2, \ldots, q_r\})$ is the number of rows in the matrix that contain all the items in the set. Also, since the rows of the user–item matrix R have been normalized to be of unit length $R_{i,q_1} = R_{i,q_2} = \cdots = R_{i,q_r}$.

5.2 Applying the Model

The *top-N* recommendations for an active user are computed using the algorithm shown in Algorithm 5.2, where $\mathcal{M}^1, \ldots, \mathcal{M}^l$ are the different model matrices, U is the $m \times 1$ vector storing the items that have already been purchased by the user, and N is the number of items to be recommended. The format of U and the format of the returned vector are identical to that used by the earlier item-to-item similarity algorithm (Algorithm 4.2).

Algorithm 5.2: ApplyHigherOrderModel ($\mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^l, U, N$)

for $r \leftarrow 1$ **to** l (1)
do
  for $j \leftarrow 1$ **to** m^r
  do
    for $i \leftarrow 1$ **to** r
    do $\{q_i \leftarrow ((j \textbf{ mod } m^{r-i+1}) \textbf{ div } m^{r-i}) + 1$
    if $U_{q_1} == 1$ **and** $U_{q_2} == 1$ **and** $\cdots U_{q_r} == 1$
    then $U_j^r \leftarrow 1$
    else $U_j^r \leftarrow 0$

$x \leftarrow \sum_{r=1}^{l} \mathcal{M}^r U^r$ (2)

for $j \leftarrow 1$ **to** m (3)
do
  if $U_i \neq 0$
  then $x_i \leftarrow 0$

for $j \leftarrow 1$ **to** m (4)
do
  if $x_i \neq$ among the N largest values in x
  then $x_i \leftarrow 0$

return (x)

Algorithm 5.2 first generates l different vectors $U^1, U^2, \ldots, U^l$ of size $m \times 1$, $m^2 \times 1, \ldots, m^l \times 1$, respectively (loop at line 1). For a particular value of r, U^r is constructed by generating every possible combination of r items $\{q_1, q_2, \ldots, q_r\}$ and setting the corresponding entry of U^r to one if the active user has purchased all of these items and zero otherwise. Note that the row-index j of each itemset is constructed so that it is identical to the column-index of the same itemset used to populate the corresponding $\mathcal{M}^r$ matrix. The algorithm then computes the vector x by adding the various matrix-vector products of the corresponding $\mathcal{M}^r$ and U^r pairs (line 2). Finally, the algorithm proceeds to first filter out the items that the active user has already purchased (loop at line 3) and then retain only the N most similar items (loop at line 4).

5.3 Practical Considerations

Unfortunately, the higher-order item-based models described in the previous section are not computationally feasible because the model parameters that we need to compute and store (i.e., the k most similar items of the various itemsets) grows exponentially with the order of the model. Moreover, for most datasets, the occurrence frequency of many of these itemsets will be either zero or very small making it impossible to accurately estimate the k most similar items for each itemset. For this reason, our higher-order algorithms do not compute and store the itemset-to-item similarities for all itemsets but only for those itemsets that occur a sufficiently large number of times in the user–item matrix R. In particular, using the notion of *frequent itemsets* [Agrawal et al. 1993, 1996] developed by the data mining community, we use a computationally efficient algorithm [Seno and Karypis 2001] to find all frequent itemsets up to size l that occur in σ% of the rows (i.e., transactions), and compute the k most similar other items only for these frequent itemsets. Note that the threshold σ is commonly referred to as the *minimum support constraint*.

The frequent-itemset based approach solves the issues associated with computational complexity but introduces two new problems. First, we need to develop a method that can be used to select the value of the minimum support constraint. A high value will result in a higher-order scheme that uses very few itemsets and as such it does not utilize its full potential, whereas a low value may lead to an exponentially large number of itemsets, making it computationally intractable. Unfortunately, there are no good ways to a priori select the value of support. This is because for a given value of σ the number of frequent itemsets that exist in a dataset depends on the dataset's density and the item co-occurrence patterns in the various rows. The same support value can lead to very few patterns in one dataset and a huge number of patterns in another. For this reason, selecting the right value of σ may require extensive experimentation to obtain a good balance between computational efficiency and *top-N* recommendation quality.

Second, since our higher-order models now only contain information about a small subset of the possible itemsets, there may be a number of itemsets that can be constructed from the items present in the active user's basket U that are not present in the model. One solution to this problem may be to just ignore

those itemsets while computing *top-N* recommendations. Such an approach is similar in spirit to some of the association rule-based *top-N* recommendation algorithms that are described in the related research section (Section 3) that have been shown to actually perform worse [Demiriz 2001] than the first-order item-based schemes described in Section 4. One of the reasons why such an approach may not be advisable is that if we consider the contributions that each item in U makes in determining the *top-N* recommended items, items that appear in frequent itemsets will tend to contribute more that items that do not. For example, an item that is present in a size-two and a size-three frequent itemset will have been used to determine the k most similar items of three different contributors to the final result (i.e., the k-most similar lists of the item itself and its size-two and size-three itemsets). However, an item that is not present in any frequent itemset will only contribute once to the final result. This creates an asymmetry on how the different items of a user's basket are used that can lead to relatively poor *top-N* recommendation performance.

For this reason, while computing the *top-N* recommendations for an active user we do not ignore any infrequent itemsets that it contains but use information from the first-order model to derive an approximation of its k most similar items. This is done as follows. For each infrequent itemset $\{u, v, w\}$ that is derived from U, our algorithm treats it as a new basket and computes a *top-k* recommendation using the information from the first-order model (i.e., the k most similar items of each item). The weights associated with these *top-k* recommended items are scaled to be of unit length (for the same reasons discussed in Section 4.2) and are used as the k-most similar items for that itemset. Thus, by using such an approach our algorithm does not discriminate between items that are present in frequent itemsets and items that are not, while still maintaining the computational advantages of building higher-order models based only on frequent itemsets.

6. EXPERIMENTAL RESULTS

In this section, we experimentally evaluate the item-based *top-N* recommendation algorithms and compare their performance against the performance of the user-based *top-N* recommendation algorithm. All experiments were performed on a Intel Xeon based workstation running at 1.7GHz, 1GBytes of memory, and Linux-based operating system.

6.1 Experimental Design and Metrics

To evaluate the quality of the *top-N* recommendations, we split each of the datasets into a *training* and *test* set by randomly selecting one of the nonzero entries of each row to be part of the test set, and used the remaining entries for training.[1] For each user we obtained the *top-N* recommendations by using the items present in the training set as the *basket* for that user. In the case of the item-based algorithms, the *top-N* recommendations were computed using only the training set to build the item similarity models. Similarly, in the case of

[1]Our datasets were such that each row had at least two nonzero entries.

the user-based algorithms, the nearest neighbors and *top-N* recommendations were computed only using the training set.

The quality was measured by looking at the number of *hits* and their position within the *top-N* items that were recommended by a particular scheme. The number of hits is the number of items in the test set that were also present in the *top-N* recommended items returned for each user. We computed two quality measures that we will refer to them as the *hit-rate* (HR) and the *average reciprocal hit-rank* (ARHR) that are defined as follows. If n is the total number of customers/users, the hit-rate of the recommendation algorithm was computed as:

$$\textit{hit-rate (HR)} = \frac{\textit{Number of hits}}{n}. \tag{5}$$

An HR value of 1.0 indicates that the algorithm was able to always recommend the hidden item, whereas an HR value of 0.0 indicates that the algorithm was not able to recommend any of the hidden items. One limitation of the hit-rate measure is that it treats all hits equally regardless of where they appear in the list of the *top-N* recommended items. That is, a hit that occurs in the first position is treated equally with a hit that occurs in the Nth position. This limitation is addressed by the average reciprocal hit-rank measure that rewards each hit based on where it occurred in the *top-N* list. If h is the number of hits that occurred at positions $p_1, p_2, \ldots, p_h$ within the *top-N* lists (i.e., $1 \leq p_i \leq N$), then the average reciprocal hit-rank is equal to

$$\textit{average reciprocal hit-rank (ARHR)} = \frac{1}{n}\sum_{i=1}^{h}\frac{1}{p_i}. \tag{6}$$

That is, hits that occur earlier in the *top-N* lists are weighted higher than hits that occur later in the list. The highest value of ARHR is equal to the hit-rate and occurs when all the hits occur in the first position, whereas the lowest value of the ARHR is equal to *hit-rate*/N when all the hits occur in the last position in the list of the *top-N* recommendations.

In order to ensure that our results are not sensitive to the particular training-test partitioning of each dataset, for each of the experiments we performed ten different runs, each time using a different random partitioning into training and test sets. The results reported in the rest of this section are the averages over these ten trials. Furthermore, to better compare the various results we used two different statistical tests to compare the averages obtained from the ten different random partitionings that are based on the paired t-test for pairwise comparisons and on the Bonferroni test for multiple comparisons. Both tests were performed at a 95% confidence interval.

Finally, in all of experiments we used $N = 10$ as the number of items top be recommended by the *top-N* recommendation algorithms.

6.2 Evaluation on Real Datasets

We evaluated the performance of the different *top-N* recommendation algorithms using eight different datasets whose characteristics are shown in Table I. For each dataset, this table shows the number of users, the number

Table I. The Characteristics of the Various Datasets used in Evaluating the *top-N* Recommendation Algorithms

Name	Number of Users	Number of Items	Number of Transactions	Density	Average Basket Size
ctlg1	58565	502	209715	0.71%	3.58
ctlg2	23480	55879	1924122	0.15%	81.95
ctlg3	58565	39080	453219	0.02%	7.74
ccard	42629	68793	398619	0.01%	9.35
ecmrc	6667	17491	91222	0.08%	13.68
em	8002	1648	769311	5.83%	96.14
ml	943	1682	100000	6.31%	106.04
skill	4374	2125	82612	0.89%	18.89

of items, and the total number of transactions (i.e., nonzeros in the resulting user–item matrix). In addition, the column labeled "Density" shows the percentage of nonzero entries in the user–item matrix and the column labeled "Avg. Basket Size" shows that average number of items in each transaction.

These datasets can be broadly classified into two categories. The first category (containing the first five datasets) was derived from customer purchasing transactions and is typical of datasets arising in e-commerce and traditional marketing applications of *top-N* recommender systems. Specifically, the *ctlg1*, *ctlg2*, and *ctlg3* datasets correspond to the catalog purchasing transactions of two major mail-order catalog retailers. Note that *ctlg1* and *ctlg3* correspond to the same set of transactions but they differ on what constitutes an item. The items of *ctlg3* correspond to individual products, whereas the items of *ctlg1* correspond to the top-level product categories, that is, a particular nonzero entry in the user–item matrix is a transaction indicating that a particular user has purchased an item from a particular product category. The *ecmrc* dataset corresponds to web-based purchasing transactions of an e-commerce site. The *ccard* dataset corresponds to credit card purchasing transactions of a major department store's credit card.

The second category (containing the remaining datasets) was obtained from two different application areas and corresponds to non-traditional uses of *top-N* recommender systems. In particular, the *em* and *ml* datasets correspond to movie ratings and were obtained from the *EachMovie* [McJones and DeTreville 1997] and the *MovieLens* [MovieLens 2003] research projects, respectively. Note that these two datasets contain multi-value ratings that indicate how much each user liked a particular movie or not. For the purpose of our experiments we ignored the values of these ratings and treated them as an indication that the user has seen that particular movie. By performing this conversion we focus on the problem of predicting whether or not a user will see a particular movie and not whether or not he or she will like it. Finally, the *skill* dataset corresponds to the information technology related skills that are present in the resumes of various individuals and were obtained from a major online job portal. The *top-N* recommendation problem in this dataset is that of predicting a set of other related skills that can potentially act as a suggestion to the user on how to improve his or her career.

Table II. The Effect of the Similarity Normalization on the Recommendation Quality Achieved by the First-Order Cosine- and Conditional Probability-Based Recommendation Algorithms

	Top-10 Hit-Rate				Top-10 Average Reciprocal Hit-Rank			
	Cosine		Cond. Probability		Cosine		Cond. Probability	
	SNorm+	SNorm−	SNorm+	SNorm−	SNorm+	SNorm−	SNorm+	SNorm−
ctlg1	**0.406**	0.396	**0.415**	0.404	**0.208**	0.203	**0.213**	0.206
ctlg2	**0.147**	0.143	**0.154**	0.127	**0.070**	0.069	**0.074**	0.064
ctlg3	**0.534**	0.529	**0.540**	0.515	**0.315**	0.310	**0.320**	0.303
ccard	**0.162**	0.160	**0.176**	0.167	**0.119**	0.118	**0.130**	0.126
ecmrc	**0.170**	0.166	0.174	0.174	**0.096**	0.093	0.098	0.097
em	**0.407**	0.400	**0.405**	0.395	**0.189**	0.186	**0.189**	0.183
ml	**0.271**	0.264	**0.272**	0.249	**0.119**	0.115	**0.119**	0.110
skill	**0.370**	0.358	**0.373**	0.313	**0.178**	0.172	**0.178**	0.151

Bold-faced entries correspond to schemes that perform statistically better at 95% confidence interval using the paired *t*-test.

6.2.1 *Parameter Evaluation.* Since there are a number of alternative options that control the various aspects of the proposed item-based *top-N* recommendation algorithm, it is not possible to provide an exhaustive comparison of all possible combinations without making this article unduly large. Instead, we provide comparisons of different alternatives for each option after making a reasonable choice for the other options.

6.2.1.1 *Effect of Similarity Normalization.* Our first experiment was designed to evaluate the effect of the similarity normalization that is discussed in Section 4.2. Table II shows the HR and ARHR results achieved by four different item-based recommendation algorithms. Two of them use the cosine as the similarity function whereas the other two use the conditional probability. The difference between each pair of algorithms is that one does not normalize the similarities (those labeled "SNorm–') whereas the other normalizes them (those labeled "SNorm+"). For all four algorithms, the rows of the matrix were normalized so that they are of unit length, k (the number of nearest items to use in the model) was set to 20, and a value of $\alpha = 0.5$ was used for the schemes that are based on the conditional probability-based approach. In addition, all of these schemes correspond to first-order item-based models.

Looking at the results in Table II, we can see that the algorithms that use similarity normalization achieve better results (both in terms of HR and ARHR) compared to their counterparts that do not use such normalization. As we can see from these results, in all cases the scheme that normalizes the similarity values performs better than the scheme that does not. The actual improvement is dataset and algorithm dependent. In general, the relative improvements tend to be higher for the conditional probability based scheme than the cosine-based scheme. On average, the HR of the cosine-based scheme improves by 2.15%, whereas the HR of the conditional probability-based scheme improves by 7.85%. Similar trends are observed when comparing the performance of the various algorithms using the ARHR measure. To ensure that these differences

Table III. The Effect of Row Normalization on the Recommendation Quality Achieved by the Cosine- and Conditional Probability-Based Recommendation Algorithms

	Top-10 Hit-Rate				Top-10 Average Reciprocal Hit-Rank			
	Cosine		Cond. Probability		Cosine		Cond. Probability	
	RNorm+	RNorm−	RNorm+	RNorm−	RNorm+	RNorm−	RNorm+	RNorm−
ctlg1	0.406	0.406	**0.415**	0.406	0.208	0.208	**0.213**	0.208
ctlg2	**0.147**	0.143	**0.154**	0.143	0.070	0.069	**0.074**	0.069
ctlg3	0.534	**0.536**	**0.540**	0.536	0.315	**0.317**	**0.320**	0.317
ccard	0.162	**0.179**	0.176	**0.179**	0.119	**0.132**	0.130	**0.132**
ecmrc	0.170	**0.173**	**0.174**	0.173	0.096	**0.097**	0.098	0.097
em	**0.407**	0.395	**0.405**	0.395	**0.189**	0.186	**0.189**	0.186
ml	**0.271**	0.261	**0.272**	0.260	**0.119**	0.112	**0.119**	0.112
skill	**0.370**	0.344	**0.373**	0.344	**0.178**	0.165	**0.178**	0.165

For each experiment, bold-faced entries correspond to schemes that perform statistically better using the paired t-test.

are statistically significant we tested them using the paired t-test. Table II highlights with a bold-faced font the HR and ARHR entries of the scheme that is significantly better than the others. As we can see from these results, for all datasets, the differences are indeed statistically significant. Due to this performance advantage in the rest of our experiments, we will always use similarity normalization.

6.2.1.2 *Effect of Row Normalization.* The second experiment was designed to evaluate the effect of row-normalization so that customers that purchase many items will weigh less during the item similarity calculations. Table III shows the HR and ARHR achieved by four different item-based recommendation algorithms. Two of them use the cosine as the similarity function whereas the other two use the conditional probability. The difference between each pair of algorithms is that one does not normalize the rows (those labeled "RNorm−") whereas the other normalizes them (those labeled "RNorm+"). Also, the entries in Table III that correspond to the schemes that perform statistically better based on the paired t-test are highlighted using a bold-faced font. For all experiments k was set to 20, and for the two conditional probability-based algorithms, a value of $\alpha = 0.5$ was used. In addition, all of these schemes correspond to first-order item-based models.

From the results in Table III we can see that on average, the row-normalized version performs somewhat better for both the cosine- and the conditional probability-based schemes. Specifically, the average improvement in terms of HR for all eight datasets is 0.69% for the cosine- and 3.05% for conditional probability-based scheme. Similar observations can be made by looking at the ARHR results as well. Comparing the statistical significance of these results, we can see that for the conditional probability-based scheme the scheme that normalizes the rows performs statistically better on seven out of the eight datasets. However, in the case of the cosine-based scheme there is a certain amount of variation in which scheme performs statistically better, and some of these improvements are not statistically significant. Because of these improvements in the rest of our experiments we will always use row normalization.

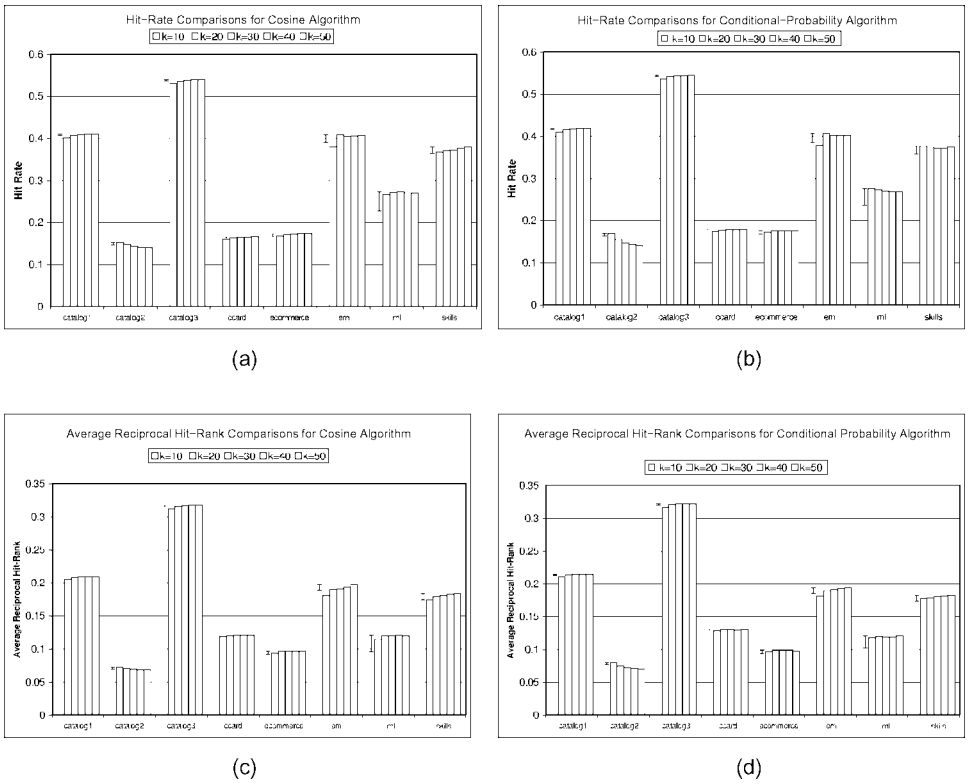


Fig. 1. The HR and ARHR as a function of the number of most similar items (k) used in computing the *top-N* recommendations for the cosine- and conditional probability-based recommendation algorithms. The error bars associated with each dataset correspond to the minimum required difference in either HR or ARHR in order for two schemes to be statistically different.

6.2.1.3 *Model Size Sensitivity.* Recall from Section 4.1 that the item-based recommendations are computed using a model that utilizes the k most similar items for each one of the different items. To evaluate the sensitivity of the different algorithms on the value of k we performed an experiment in which we let k take the values of 10, 20, 30, 40, and 50. The recommendation performance in terms of HR and ARHR for these experiments is shown in Figure 1 for the cosine- and conditional probability-based algorithms. For each dataset Figure 1 also shows the minimum required difference in the respective performance measure in order for a particular value of k to perform significantly better (or worse) than the remaining k values. These differences were computed using the Bonferroni test and are shown using the error-bars. For both classes of algorithms we used the first-order models and in the case of the conditional probability-based schemes the experiments were performed using a value of $\alpha = 0.5$.

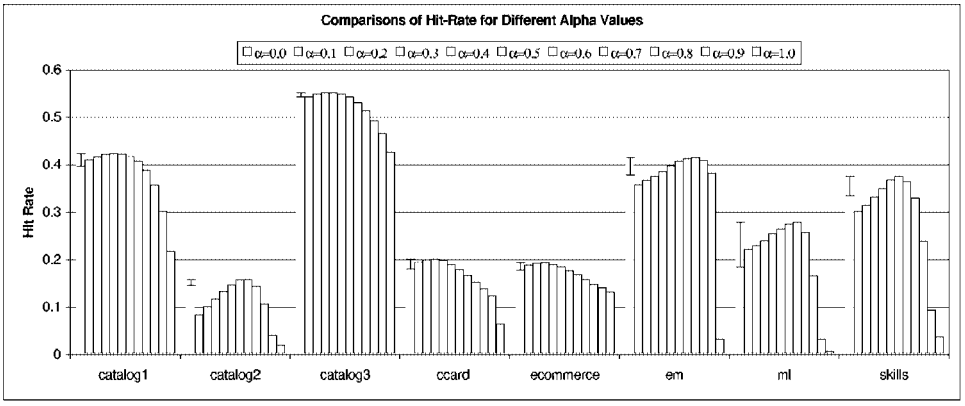
As we can see from these experiments, the overall recommendation accuracy of the item-based algorithms does tend to improve as we increase the value of k. The only exception is the *ctlg2* dataset for which both the HR and the

ARHR tend to consistently decrease as we increase k. Overall, the average HR for the cosine-based algorithm improves by 1.8% as we vary k from 10 to 50 items; whereas in the case of the conditional probability-based algorithm the average improvement in HR is 0.65%. Similar minor improvements are achieved in terms of ARHR as well. However, as the figure illustrates, most of these improvements are not statistically significant and no particular value of k dominates the rest. These results indicate that (i) even for small values of k the item-based recommendation algorithms provide reasonably accurate recommendations; and (ii) increasing the value of k does not lead to significant improvements. This is particularly important since small values of k lead to fast recommendation rates (i.e., low computational requirements) without materially affecting the overall quality of the recommendations. Note that the diminishing incremental improvements achieved by increasing the value of k are a direct consequence of the fact that we are only looking for 10 recommended items (i.e., $N = 10$). As a result, once k is sufficiently large, to ensure that the various item-to-item lists have sufficient common items, any further increases in k will not change the order of the *top-N* items.

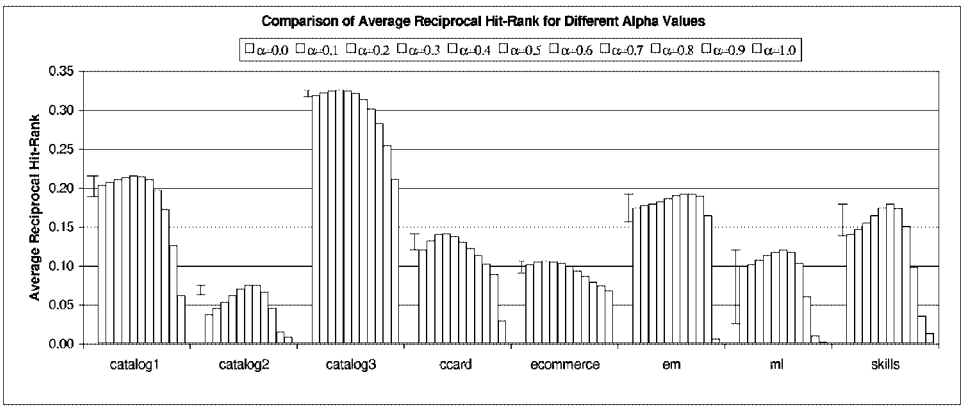
6.2.1.4 *Item Frequency Scaling Sensitivity.* One of the parameters of the conditional probability-based *top-N* recommendation algorithm is the value of α used to control the extent to which the similarity to frequently purchased items will be de-emphasized. To study the sensitivity of the recommendation algorithm on this parameter we performed a sequence of experiments in which we varied α from 0.0 to 1.0 in increments of 0.1. Figure 2 shows the HR and ARHR achieved on the various datasets for the different values of α. As with the results of the previous study, the minimum required differences computed using the Bonferroni test are shown using error bars. Note that these results were obtained using the first-order item-based model and $k = 20$.

From these results we can see that for all datasets the value of α has a significant impact on the recommendation quality, as different values of α lead to substantially different values of HR and ARHR. Despite this variability, for almost all datasets, if $0.3 \leq \alpha \leq 0.6$, then the conditional probability-based scheme achieves consistently good performance. Also note that as we increase the value of α, the changes in the HR and ARHR are fairly smooth, and follow a $\cap$-shaped curve. This suggests that the optimal value of α can be easily estimated for each particular dataset by hiding a portion of the training set and using it to find the value of α that leads to the highest HR or ARHR. Moreover, since the values of α that lead to both the highest HR or ARHR values are consistent for most of the datasets, we can learn the value of α that optimizes one of the two measures as it will also lead to optimal or near-optimal performance with respect to the other measure.

A further study of the values of α that lead to the highest HR and ARHR values and the properties of the various datasets used in our experiment reveal another interesting trend. If we compare the highest HR value to the HR value achieved for $\alpha = 0.0$ we see that for *ctlg1*, *ccard*, *ecmrc*, and *ctlg3*, the highest value is usually less than 3.2% better than that for $\alpha = 0.0$. On the other hand, the improvement for *skill*, *em*, *ctlg2*, and *ml* ranges from 16% to 91%.



(a)



(b)

Fig. 2. The HR and ARHR as a function of the item-frequency-based scaling achieved by the α parameter for conditional probability-based recommendation algorithms. The error bars associated with each dataset correspond to the minimum required difference in either HR or ARHR in order for two schemes to be statistically different.

Similar trends can be observed by focusing on the ARHR measure. Thus, there is a group of datasets for which there is a clear benefit in trying to optimize the value of α. Moreover, the datasets for which we achieve significant HR (or ARHR) improvements are those datasets that according to the statistics shown in Table I have some of the highest densities and the largest number of items per user.

6.2.1.5 *Model Order Sensitivity.* Our experiments so far focused on first-order item-based *top-N* recommendation algorithms. However, as discussed in Section 5, both the cosine- and the conditional probability-based schemes can be extended to higher order-models by using frequent itemsets of different length and using an interpolating approach to combine the recommendations performed by the different models. Table IV shows the HR and the ARHR results

Table IV. The Recommendation Quality as a Function of the Order of the Model that is Used

Top-10 Hit-Rate										
Name	σ (%)	F_2	F_3	Cos1	Cos2	Cos3	CPrb1	CPrb2	CPrb3	RqDiff
ctlg1	0.1	868	486	0.406	0.405	0.406	0.415	0.414	0.416	0.0039
ctlg2	5.0	764	835	0.147	0.147	0.147	0.154	0.154	0.154	0.0048
ctlg3	0.05	2150	1437	0.534	0.535	0.535	0.540	0.540	0.540	0.0022
ccard	0.01	3326	2056	0.162	0.162	0.162	0.176	0.175	0.175	0.0022
ecmrc	0.01	255	112	0.170	0.170	0.170	0.174	0.174	0.174	0.0094
em	20.0	4077	52434	0.407	0.419	0.416	0.405	0.418	0.415	0.0108
ml	10.0	9921	87090	0.271	0.267	0.270	0.272	0.279	0.275	0.0284
skill	1.0	4485	16820	0.370	0.361	0.367	0.373	0.380	0.379	0.0191
Top-10 Average Reciprocal Hit-Rank										
Name	σ (%)	F_2	F_3	Cos1	Cos2	Cos3	CPrb1	CPrb2	CPrb3	RqDiff
ctlg1	0.1	868	486	0.208	0.208	0.208	0.213	0.213	0.214	0.0037
ctlg2	5.0	764	835	0.070	0.070	0.070	0.074	0.074	0.074	0.0035
ctlg3	0.05	2150	1437	0.315	0.316	0.315	0.320	0.320	0.321	0.0027
ccard	0.01	3326	2056	0.119	0.118	0.119	0.130	0.128	0.129	0.0013
ecmrc	0.01	255	112	0.096	0.095	0.096	0.098	0.098	0.098	0.0031
em	20.0	4077	52434	0.189	0.201	0.200	0.189	0.199	0.197	0.0052
ml	10.0	9921	87090	0.119	0.114	0.118	0.119	0.120	0.119	0.0143
skill	1.0	4485	16820	0.178	0.172	0.176	0.178	0.184	0.184	0.0130

Underlined entries correspond to the higher-order schemes that perform statistically better than the corresponding first-order scheme.

obtained by using such higher-order interpolated models for both the cosine- and the conditional probability-based approaches. In particular, Table IV shows the results obtained by a first-, second-, and third-order interpolated models. Note that the first-order model results are identical to those presented in the previous sections.

One of the key parameters of higher-order models is the support threshold (σ) used by the frequent pattern discovery algorithm to identify the frequent itemsets to be used in the models. We used different values of the support threshold for each dataset depending on the density and the degree to which different items co-occur in the different datasets. These values are shown in the second column of Table IV. They were selected so that (i) they lead to a reasonable number of frequent itemsets and (ii) each frequent itemset has a sufficiently large support to ensure the statistical significance of the similarities that are computed between an itemset and the remaining items. The actual number of frequent size-two and size-three frequent itemsets that were discovered and used to build the interpolated second- and third-order models are shown in the columns labeled "F_2" and "F_3", respectively. The last column in these tables (labeled "Reqd. Diff") shows the minimum difference of the respective performance measure that is required in order for two schemes to be statistically different from each other using the Bonferroni test. For all experiments k was set to 20, and for the conditional probability-based algorithms we used a value of $\alpha = 0.5$.

As we can see from these results, higher-order item-based models do not lead to any significant improvements in either HR or ARHR. For most datasets, the results obtained across the different schemes (i.e., 1st-, 2nd-, and 3rd-order

Table V. The Quality of the Recommendations Obtained by the Naive, the Item-Based, and the User-Based Recommendation Algorithm

	Top-10 Hit-Rate				
	User	Frequent	Cosine	CProb-$\alpha = 0.5$	CProb-α =Opt
ctlg1	0.398	0.215	0.406	0.415	0.421
ctlg2	0.150	0.025	0.147	0.154	0.155
ctlg3	0.494	0.030	0.534	0.540	0.549
ccard	0.158	0.079	0.162	0.176	0.198
ecmrc	0.178	0.029	0.170	0.174	0.191
em	0.453	0.367	0.407	0.405	0.412
ml	0.281	0.131	0.271	0.272	0.276
skill	0.384	0.238	0.370	0.373	0.373
	Top-10 Average Reciprocal Hit-Rank				
	User	Frequent	Cosine	CProb-$\alpha = 0.5$	CProb-α =Opt
ctlg1	0.206	0.080	0.208	0.213	0.214
ctlg2	0.076	0.009	0.070	0.074	0.074
ctlg3	0.298	0.010	0.315	0.320	0.324
ccard	0.119	0.066	0.119	0.130	0.140
ecmrc	0.095	0.012	0.096	0.098	0.105
em	0.221	0.169	0.189	0.189	0.191
ml	0.128	0.046	0.119	0.119	0.119
skill	0.189	0.091	0.178	0.178	0.178

models) are very similar or within less than 1% of each other, which according to the Bonferroni test is not statistically significant. The only datasets for which higher-order models, and the second-order model in particular, did somewhat better than the first-order model are the *em*, *ml*, and *skill* datasets. In particular, the second-order model improved the HR in the above datasets by 1.8% to 3.2%, and the ARHR by 3.3% to 6.3%. Also note that these three datasets are the ones that contain the most size-two and size-three frequent itemsets, suggesting that when a particular dataset contains a sufficient number of frequent itemsets, the higher-order models can improve the quality of the *top-N* recommendations. However, among these datasets, only the improvements achieved for the *em* dataset (corresponding to the underlined entries) are statistical significant.

6.2.2 *Overall Comparisons.* To compare the performance of the various item-based recommendation algorithms against each other and with that achieved by user-based algorithms we performed an experiment in which we computed the *top-N* recommendations using both the item-based and the user-based recommendation algorithms. The results from these experiments are shown in Table V. The user-based recommendations were obtained using the algorithm described in Herlocker et al. [1999] and Sarwar et al. [2000] with user-neighborhoods of size 50 and unit-length normalized rows. We used a similarity-weighted approach to determine the frequency of each item, and we did not include neighbors that had an identical set of items as the active item (as these neighbors do not contribute at all in the recommendation).

Table V includes three different sets of item-based results obtained with $k = 20$. The results labeled "Cosine" correspond to the cosine-based results.

The results labeled "CProb-$\alpha = 0.5$" correspond to the conditional probability-based algorithm in which α was set to 0.5. The results labeled "CProb-$\alpha = $ Opt" correspond to the conditional probability-based algorithm that uses the value of α that achieved the highest performance in the experiments discussed in Section 6.2.1 for each dataset. All the item-based results were obtained using the first-order models. Table V also includes the *top-N* recommendation quality achieved by the naive algorithm, labeled "Frequent", which recommends the N most frequent items not already present in the active user's set of items.

Comparing the performance achieved by the item-based schemes with that achieved by the user-based scheme we can see that the cosine-based scheme performs better than the user-based scheme in three out of the eight datasets, whereas the conditional probability-based schemes that use $\alpha = 0.5$ and $\alpha = $ Opt outperform the user-based scheme in four out of eight and five out of eight datasets, respectively. On average, the cosine-based scheme does 1.15% and 4.04% worse than the user-based scheme in terms of HR and ARHR, respectively; the conditional probability-based scheme with $\alpha = 0.5$ does 1.10% better and 1.30% worse than the user-based scheme in terms of HR and ARHR, respectively; whereas the conditional probability-based scheme with the best choice for α does 4.65% and 1.29% better than the user-based scheme in terms of HR and ARHR, respectively. In general, all three item-based schemes seem to do worse than the user-based scheme for the denser datasets (e.g., *skill*, *em*, and *ml*), and do better for the sparser datasets (e.g., *ccard*, *ecmrc*, and *ctlg3*). Also the performance of the item-based schemes relative to the user-based scheme is somewhat worse when measured in terms of ARHR instead of HR. This suggests that in the case of user-based schemes the hidden items (i.e., hits) occur earlier in the list of *top-N* recommended items, even if in some cases the aggregate number hidden items that were able to recommend is smaller than the total number recommended by the item-based schemes.

Comparing the results achieved by the various item-based schemes we can see that the schemes based on conditional probability perform better than those based on cosine similarity. On average, in terms of HR, the conditional probability-based scheme with $\alpha = 0.5$ does 2.5% better than the cosine-based scheme, whereas the scheme using the optimal value of α performs 5.9% better. Finally, both the user- and item-based algorithms produce recommendations whose quality is substantially better than the recommendations produced by the naive "Frequent" algorithm.

To ensure that the above comparisons are significant we used the paired t-test to determine the number of datasets in which one scheme outperforms the other at a confidence interval of 95%. The results of this analysis are shown in Table VI for both the HR and the ARHR measures. Each entry in these two 5×5 tables contains three numbers that correspond to the number of datasets in which the scheme corresponding to the row performed statistically better, the same, or worse than the scheme corresponding to the column, respectively. As we can see from these results, in almost all cases, the performance differences between each pair of schemes are statistically significant.

Table VI. Statistical Significance Comparisons of the Various *top-N* Recommendation Algorithms Using the Paired *t*-Test

	Top-10 Hit-Rate				
	User	Frequent	Cosine	CProb-$\alpha = 0.5$	CProb-α =Opt
User	—	8, 0, 0	5, 0, 3	4, 0, 4	3, 0, 5
Frequent	0, 0, 8	—	0, 0, 8	0, 0, 8	0, 0, 8
Cosine	3, 0, 5	8, 0, 0	—	1, 1, 6	0, 0, 8
CProb-α=0.5	4, 0, 4	8, 0, 0	6, 1, 1	—	0, 1, 7
CProb-α=Opt	5, 0, 3	8, 0, 0	8, 0, 0	7, 1, 0	—
	Top-10 Average Reciprocal Hit-Rank				
	User	Frequent	Cosine	CProb-$\alpha = 0.5$	CProb-α =Opt
User	—	8, 0, 0	4, 1, 3	4, 0, 4	4, 0, 4
Frequent	0, 0, 8	—	0, 0, 8	0, 0, 8	0, 0, 8
Cosine	3, 1, 4	8, 0, 0	—	1, 2, 5	0, 2, 6
CProb-α=0.5	4, 0, 4	8, 0, 0	5, 2, 1	—	0, 3, 5
CProb-α=Opt	4, 0, 4	8, 0, 0	6, 2, 0	5, 3, 0	—

The three numbers in each cell show the number of datasets in which the scheme corresponding to the row performed statistically better, the same, or worse than the scheme corresponding to the column.

Table VII. The Computational Requirements for Computing the *top-N* Recommendations for Both the User- and Item-Based Algorithms

	User-based		Item-based		
Name	RcmdTime	RcmdRate	ModelTime	RcmdTime	RcmdRate
ctlg1	62.68	934	0.10	0.16	366031
ctlg2	83.53	281	19.35	1.82	12901
ctlg3	13.57	4315	0.69	0.78	75083
ccard	17.59	2427	0.98	0.79	53960
ecmrc	0.48	13889	0.10	0.08	83337
em	49.25	162	1.74	0.33	24248
ml	0.46	2049	0.24	0.05	18859
skill	1.64	2667	0.13	0.07	62485

6.2.2.1 *Computational Requirements.* One of the advantages of the item-based algorithm is that it has much smaller computational requirements than the user-based *top-N* recommendation algorithm. Table VII shows the amount of time required by the two algorithms to compute the *top-N* recommendations for each one of the eight datasets. The column labeled "ModelTime" shows the amount of time required to build the item-based recommendation model (i.e., compute the k most similar items), the columns labeled "RcmdTime" show the amount of time required to compute the n recommendations for each one of the datasets, and the columns labeled "RcmdRate" show the rate at which the *top-N* recommendations were computed in terms of *recommendations/second*. Note that our implementation of the user-based *top-N* recommendation algorithm takes advantage of the sparse user–item matrix, and uses inverted indices in order to identify the nearest users as quickly as possible. All the times in Table VII are in seconds.

Looking at the results of Table VII we can see that the recommendation rates achieved by the item-based algorithm are 6 to 391 times higher than those achieved by the user-based algorithm. If we add the various "RcmdTime" for all eight data sets we can see that the overall recommendation rate for the item-based algorithm is 56715 recommendations/second compared to only 930 recommendations/second achieved by the user-based algorithm. This translates to one recommendation every 17 μs for the item-based algorithm, versus 1075 μs for the user-based algorithm. Also, as discussed in Section 4.3, the amount of time required to build the models for the item-based algorithm is quite small. In particular, even accounting for the model building time, the item-based algorithm is still 2 to 240 times faster than the user-based algorithm. Note that the reason that the user-based scheme is still slower even when we take into account the time required to build the models is the fact that the resulting user-user similarity matrix that needs to be computed is much denser than the corresponding item-item similarity matrix. This is because the density of the user–user similarity matrix depends on the existence of some frequently purchased items (i.e., dense columns in the matrix) which happens quite often, whereas in the case of the item–item similarity matrix, it is rare to have any dense rows (i.e., users that have purchased most of the items).

6.3 Evaluation on Synthetic Datasets

The performance of recommender systems is highly dependent on various characteristics of the dataset such as the number of items, the number of users, its sparsity, and the behavioral variability of the various users in terms of the items they buy/see. Furthermore, as the results in Section 6.2.2 have shown, the relative performance of various *top-N* recommendation algorithms do not vary uniformly across different datasets and it is quite likely that a particular scheme will outperform the rest for a particular dataset, whereas the same scheme might underperform when the dataset characteristics are changed. This dataset-specific behavior of recommendation schemes makes it hard to decide the best scheme for a particular application. The goal of this section is to study the influence of two key dataset characteristics, *sparsity* and *user's behavioral variability*, on the performance of the recommendation system and gain some insights as to which *top-N* recommendation algorithm is better-suited to which characteristics of a dataset. We conduct this study on synthetically generated datasets as they provide us the flexibility to individually isolate a dataset characteristic and vary its value while keeping the other characteristics constant.

We make use of the IBM synthetic dataset generator [Agrawal and Srikant 1994], which is widely used to mimic the transactions in the retail environment. The dataset generator is based on the observation that people tend to make purchases in sets of items. For example, if a user's basket contains {*pillow covers*, *sheets*, *comforter*, *milk*, *bread*, *eggs*}, then it can be thought of as made of two sets of items, the first set consists of items {*pillow covers*, *sheets*, *comforter*} and the second set is made of {*milk*, *bread*, *eggs*}. This set of items is referred as *itemset*. It is observed that the size of such *itemsets* is clustered around a

Table VIII. Parameters Taken by Synthetic Dataset Generator

Description	Symbol	IBM Symbol	Value
Number of users	n	$\|\mathcal{D}\|$	5000
Number of items	m	N	1000
Average size of user's basket	S_u	$\|\mathcal{T}\|$	15, 30, & 45
Average size of itemset	S_{is}	$\|\mathcal{I}\|$	4, 6, & 8
Number of itemsets	N_{is}	$\|\mathcal{L}\|$	800, 1200, 1600, & 2000

mean with a few large *itemsets*. Similarly, the size of the user's basket is also clustered around a mean with a few users making lots of purchases.

The IBM dataset generator first creates a list of itemsets and then builds each user's basket from these itemsets. Some of the key parameters that are used by the generator to define the characteristics of the synthetic dataset are shown in Table VIII. The first two parameters, n and m, determine the size of the dataset by identifying the number of customers and the number of items (i.e., the $n \times m$ user–item matrix). The generator creates N_{is} itemsets whose size is governed by a Poisson distribution having a mean of S_{is}. The items making up the itemset are chosen randomly with some care taken to ensure that there is some overlap in the different itemsets. After creating the itemsets to be used the dataset is generated by creating a basket of items for each user. The size of the basket follows a Poisson distribution with mean S_u. Once the size is identified, the basket is filled with itemsets. If an itemset does not fit in the basket then it is added to the basket anyway in half the cases and moved to the next basket in the rest of the cases. To ensure that some itemsets occur more frequently than the rest, each itemset is assigned a weight and the probability of an itemset being selected is governed by that weight. The weights are assigned according to an exponential distribution with mean equal to one. In addition, to create transactions with higher variability, the generator randomly changes some of the items in each itemset as it is inserted into the transaction.

In order to evaluate the effect of the sparsity and the variability in the user's behavior on the overall performance of the various item- and user-based *top-N* recommendation algorithms, we generated 36 different datasets in which we fixed n and m but we varied S_u, S_{is}, and N_{is}. The range of values used to generate the different datasets is shown in the last column of Table VIII. We generated datasets of different sparsity by increasing the average size of the user's basket (S_u) while keeping the other two parameters fixed. Specifically, we generated datasets in which each user contained 15, 30, and 45 items on average. We generated datasets with different user's behavioral variability by varying the number of different itemsets (N_{is}) and their average size (S_{is}). Assuming that S_u and S_{is} is kept fixed, by changing the number of different itemsets that can be *packed* to create the various transactions, we can influence how many distinct user-groups exist in the dataset. This is because on average, the generator will combine S_u/S_{is} itemsets randomly selected from the N_{is} itemsets to form a particular transaction. By increasing N_{is} we increase the pool of possible combinations of S_u/S_{is} itemsets and thus increase the variability (in terms of what items are included in the user's transactions) in the dataset. A somewhat different way of changing the variability of the dataset can be

performed by changing the average size of each itemset. In particular, if we fix S_u and N_{is}, then by increasing S_{is} we decrease the number of possible itemset-combinations that can exist, since now, on average, S_u/N_{is} itemsets will be included. However, because each such itemset is now larger, this affects the *complexity* of the purchasing decision represented by that particular itemset.

6.3.1 *Results.* Table IX shows the HR and ARHR achieved by both the user-based scheme and the first- and second-order interpolated item-based schemes that use either the cosine- or the conditional probability-based similarity measure. The results for the user-based scheme were obtained using exactly the same algorithm used in Section 6.2.2, whereas the item-based results were obtained by using $k = 20$, and for each dataset we used the α value that resulted in the highest HR value for the first-order conditional probability-based model. The specific values of α that were used are shown in the column labeled "α". Also, the second-order models were obtained by using a value of the support threshold of 0.01% for $S_u = 15$, 0.1% for $S_u = 30$, and 0.5% for $S_u = 45$. The number of frequent patterns that were discovered and used in these models is shown in the column labeled "F_2".

The results of Table IX provide a comprehensive comparison of the various algorithms under a wide-range of dataset characteristics. To facilitate the various comparisons we plotted some of the results of Table IX in the graphs shown in Figure 3. Each plot in this graph shows the performance achieved by the various schemes when two out of the three parameters (i.e., S_u, N_{is} and S_{is}) were kept constant. Note that the performance trends in these plots are representative of the performance achieved for different values of the fixed parameters. In the rest of this section, we provide an overview of some of the key trends that can be inferred by comparing and analyzing these results.

First, as illustrated in Figure 3(a,b), the performance (either in terms of HR or ARHR) of the various algorithms decreases as we increase the number of itemsets (N_{is}) from 800 to 2000. This performance degradation was expected because as discussed in Section 6.3, by increasing the number of itemsets used to generate the user–item matrix we essentially increase the different types of user-groups that exist in the dataset. Since the overall size of the dataset (in terms of the number of users) remains fixed, the problem of learning accurate *top-N* recommendations for each user becomes harder.

Second, as the sparsity decreases, (S_u increases from 15 to 45), and S_{is} and N_{is} remain fixed, the overall performance of the different schemes decreases (Figure 3(c,d)). We believe that this is also due to the fact that the inherent variability in the dataset also increases, since each user now contains a larger number of itemsets.

Third, the performance of the user-based and second-order item-based algorithms increases as we increase the average size of the itemsets (S_{is}) from four to eight, whereas the performance of the first-order item-based schemes tends to decrease (Figure 3(e,f) and Table IX). When S_{is} is small, the first-order item-based schemes consistently (and in some cases substantially) outperform the user-based scheme. However, as S_{is} increases, the relative performance gap between these two algorithms shrinks to a point at which the

Table IX. The HR and ARHR for Different Values of S_u, N_{is} and S_{is}

Avg. Size of User's Basket (S_u) = 15, *Sparsity* = 1.4 %												
		Top-10 HR ($S_{is} = 4$)					Top-10 ARHR ($S_{is} = 4$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.3	0.645	0.871	0.908	0.877	0.913	0.511	0.681	0.724	0.692	0.737	44100
1200	0.3	0.567	0.804	0.870	0.816	0.877	0.449	0.609	0.676	0.629	0.693	38718
1600	0.3	0.527	0.750	0.841	0.764	0.849	0.424	0.560	0.653	0.580	0.669	33546
2000	0.3	0.497	0.700	0.816	0.715	0.824	0.405	0.513	0.629	0.533	0.645	31274
		Top-10 HR ($S_{is} = 6$)					Top-10 ARHR ($S_{is} = 6$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.2	0.735	0.858	0.946	0.871	0.952	0.619	0.689	0.786	0.710	0.804	34158
1200	0.2	0.686	0.768	0.923	0.784	0.928	0.589	0.603	0.758	0.625	0.775	29660
1600	0.2	0.672	0.700	0.897	0.720	0.903	0.583	0.546	0.740	0.570	0.756	29318
2000	0.2	0.668	0.638	0.876	0.663	0.884	0.582	0.494	0.722	0.515	0.740	29942
		Top-10 HR ($S_{is} = 8$)					Top-10 ARHR ($S_{is} = 8$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.2	0.811	0.810	0.968	0.826	0.970	0.709	0.654	0.826	0.683	0.841	29675
1200	0.2	0.792	0.703	0.949	0.727	0.953	0.703	0.555	0.810	0.585	0.825	30456
1600	0.2	0.798	0.629	0.932	0.654	0.937	0.711	0.498	0.795	0.525	0.811	32873
2000	0.2	0.804	0.570	0.913	0.593	0.919	0.716	0.447	0.780	0.471	0.794	35347
Avg. Size of User's Basket (S_u) = 30, *Sparsity* = 2.8 %												
		Top-10 HR ($S_{is} = 4$)					Top-10 ARHR ($S_{is} = 4$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.6	0.555	0.791	0.850	0.802	0.840	0.398	0.427	0.589	0.491	0.577	112965
1200	0.6	0.505	0.732	0.783	0.737	0.770	0.357	0.421	0.529	0.473	0.514	108244
1600	0.5	0.469	0.686	0.732	0.685	0.730	0.332	0.406	0.495	0.398	0.494	102016
2000	0.6	0.427	0.633	0.691	0.632	0.675	0.300	0.393	0.462	0.414	0.447	98129
		Top-10 HR ($S_{is} = 6$)					Top-10 ARHR ($S_{is} = 6$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.5	0.690	0.831	0.881	0.831	0.880	0.521	0.607	0.642	0.607	0.642	107981
1200	0.4	0.609	0.734	0.824	0.736	0.829	0.459	0.533	0.596	0.529	0.606	92674
1600	0.4	0.544	0.663	0.783	0.670	0.790	0.411	0.482	0.571	0.485	0.583	87771
2000	0.4	0.497	0.596	0.751	0.606	0.760	0.381	0.430	0.544	0.435	0.556	85295
		Top-10 HR ($S_{is} = 8$)					Top-10 ARHR ($S_{is} = 8$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.3	0.730	0.780	0.907	0.794	0.915	0.570	0.582	0.679	0.605	0.704	95648
1200	0.3	0.635	0.669	0.866	0.687	0.879	0.501	0.500	0.651	0.517	0.675	88176
1600	0.4	0.582	0.596	0.838	0.611	0.849	0.465	0.446	0.633	0.462	0.650	82415
2000	0.4	0.545	0.533	0.816	0.549	0.828	0.442	0.398	0.617	0.412	0.633	85814
Avg. Size of User's Basket (S_u) = 45, *Sparsity* = 4.2 %												
		Top-10 HR ($S_{is} = 4$)					Top-10 ARHR ($S_{is} = 4$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.8	0.464	0.502	0.794	0.711	0.709	0.298	0.108	0.492	0.364	0.431	12337
1200	0.8	0.406	0.567	0.714	0.655	0.618	0.262	0.122	0.442	0.381	0.370	9334
1600	0.7	0.377	0.566	0.653	0.612	0.614	0.243	0.150	0.403	0.294	0.375	7002
2000	0.7	0.350	0.519	0.595	0.562	0.553	0.225	0.141	0.365	0.306	0.338	6217

(*Continued*)

Table IX. Continued

		Top-10 HR ($S_{is} = 6$)					Top-10 ARHR ($S_{is} = 6$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.6	0.580	0.775	0.812	0.778	0.789	0.410	0.469	0.549	0.512	0.525	9661
1200	0.5	0.538	0.688	0.721	0.688	0.720	0.381	0.421	0.484	0.417	0.484	7628
1600	0.5	0.504	0.625	0.671	0.624	0.670	0.356	0.404	0.455	0.401	0.454	6519
2000	0.5	0.468	0.562	0.627	0.561	0.626	0.331	0.369	0.421	0.367	0.420	5367
		Top-10 HR ($S_{is} = 8$)					Top-10 ARHR ($S_{is} = 8$)					
N_{is}	α	User	Cos1	Cos2	CPrb1	CPrb2	User	Cos1	Cos2	CPrb1	CPrb2	F_2
800	0.4	0.672	0.757	0.817	0.759	0.829	0.495	0.533	0.565	0.534	0.586	9568
1200	0.4	0.619	0.645	0.747	0.648	0.764	0.452	0.459	0.515	0.454	0.536	7658
1600	0.5	0.561	0.571	0.707	0.572	0.705	0.408	0.406	0.485	0.407	0.485	6006
2000	0.5	0.507	0.509	0.676	0.509	0.675	0.369	0.362	0.465	0.362	0.464	4652

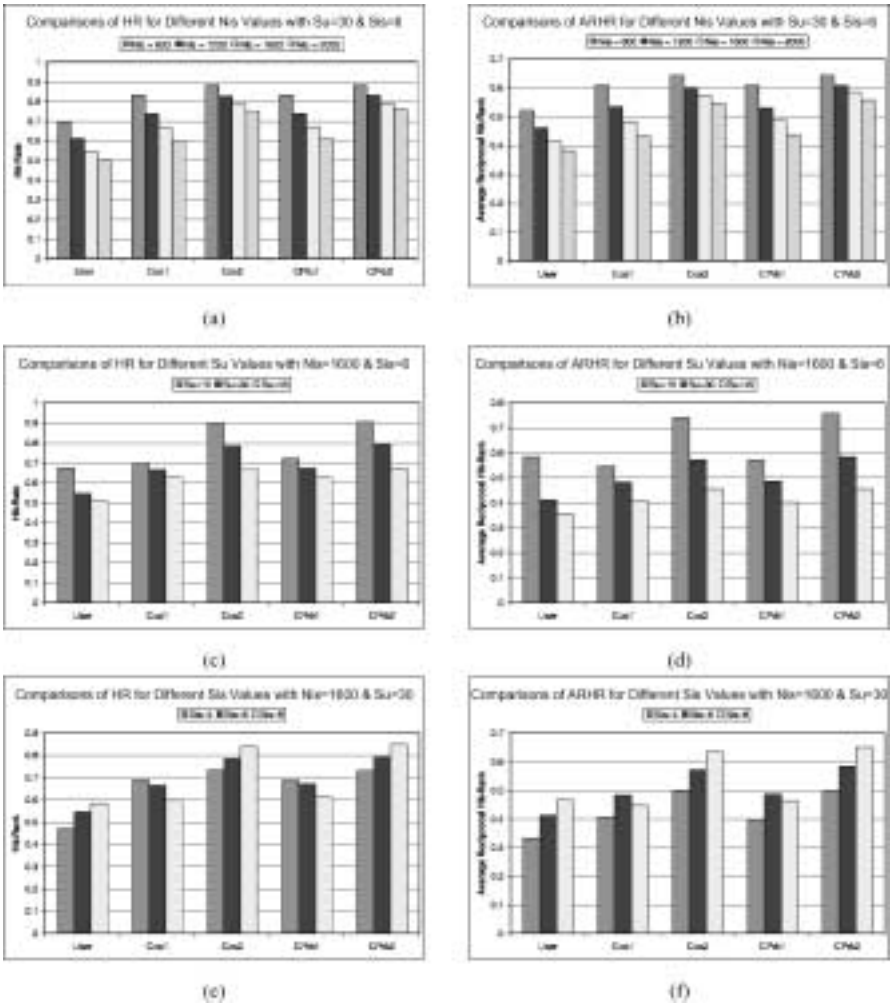


Fig. 3. The HR and ARHR for different values of S_u, N_{is} and S_{is}.

user-based scheme outperforms the first-order item-based schemes when S_{is} is eight and N_{is} is large (e.g., $S_u = 15$, $N_{is} = 2000$, and $S_{is} = 8$). Note that the relative performance advantage of user- versus item-based schemes disappears when we consider the second-order item-based schemes that always and substantially outperform the user-based scheme. The performance gains achieved by the user-based scheme can be explained by the fact that longer itemsets lead to datasets that have lower variability (i.e., each user is described by a small number of itemsets) and they are easier to identify the *correct* user-neighborhood as they will now overlap in a large number of items. However, the reason that the first-order item-based schemes perform worse while the corresponding second-order schemes perform better is somewhat more complicated. By using longer itemsets, the degree of overlap between the different itemsets that are put together to form a transaction increases as well. As a result, for each item j its similarity distribution to the other k most similar items becomes less uniform (it tends to have much higher similarities to items that co-occur with j in the itemset overlaps). Consequently, when these individual item-to-item similarities are combined to form the recommendations, they tend to be biased toward the overlapping items. This problem can be corrected by increasing the model size (i.e., the number of neighbors k that we store for each item). In fact, we performed a set of experiments in which k was increased from 20 (used to obtain the results in Table IX) to 50, and this eliminated the degradation in performance of the first-order schemes.

Fourth, comparing the various item-based schemes we can see that, as it was the case with the real datasets, the conditional probability-based approach consistently outperforms the cosine-based approach. Moreover, comparing the values of α that lead to the best performance of the conditional probability-based approach we notice a similar trend as that described in Section 6.2.1.4, as larger α-values tend to work better for denser datasets. Finally, the results of Table IX illustrate that for many datasets the second-order item-based models provide a substantial performance improvement. In many cases, the second-order models lead to improvements in the range of 50% to 80%.

7. CONCLUSIONS

In this article, we presented and experimentally evaluated a class of model-based *top-N* recommendation algorithms that use item-to-item or itemset-to-item similarities to compute the recommendations. Our results showed that both the conditional probability-based item similarity scheme and higher-order item-based models lead to recommender systems that provide reasonably accurate recommendations that are comparable or better than those provided by traditional user-based CF techniques. Furthermore, the proposed algorithms are substantially faster; allowing real-time recommendations independent of the size of the user–item matrix.

REFERENCES

AGGARWAL, C., WOLF, J., WU, K., AND YU, P. 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*, ACM, New York.

AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of 1993 ACM-SIGMOD International Conference on Management of Data* (Washington, D.C). ACM, New York.

AGRAWAL, R., MANNILA, H., SRIKANT, R., TOIVONEN, H., AND VERKAMO, A. 1996. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, U. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, Eds. AAAI/MIT Press, Cambridge, Mass., 307–328.

AGRAWAL, R. AND SRIKANT, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference* (Santiago, Chile.). 487–499.

BALABANOVIC, M. AND SHOHAM, Y. 1997. FAB: Content-based collaborative recommendation. *Commun. ACM 40*, 3 (Mar.).

BASU, C., HIRSH, H., AND COHEN, W. 1998. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the 1998 Workshop on Recommender Systems*. AAAI Press, Reston, Va. 11–15.

BEEFERMAN, D. AND BERGER, A. 2000. Agglomerative clustering of a search engine query log. In *Proceedings of ACM SIGKDD International Conference*. ACM, New York, 407–415.

BILLSUS, D. AND PAZZANI, M. J. 1998. Learning collaborative information filters. In *Proceedings of ICML*. 46–53.

BREESE, J., 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*. 43–52.

CHAN, P. 1999. A non-invasive learning approach to building web user profiles. In *Proceedings of ACM SIGKDD International Conference*. ACM, New York.

DELCHER, A. L., HARMON, D., KASIF, S., WHITE, O., AND SALZBERG, S. L. 1998. Improved microbial gene identification with glimmer. *Nucleic Acid Res. 27*, 23, 4436–4641.

DEMIRIZ, A. 2001. An association mining-based product recommender. In *NFORMS Miami 2001 Annual Meeting Cluster: Data Mining*.

GOLDBERG, D., NICHOLS, D., OKI, B. M., AND TERRY, D. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM 35*, 12, 61–70.

HECKERMAN, D., CHICKERING, D., MEEK, C., ROUNTHWAITE, R., AND KADIE, C. 2000. Dependency networks for inference, collaborative filtering, and data visualization. *J. Mach. Learn. Res. 1*, 49–75.

HERLOCKER, J., KONSTAN, J., BORCHERS, A., AND RIEDL, J. 1999. An algorithm framework for performing collaborative filtering. In *Proceedings of SIGIR*. ACM, New York, 77–87.

HILL, W., STEAD, L., ROSENSTEIN, M., AND FURNAS, G. 1995. Recommending and evaluating choices in a virtual community of use. In *Proceedings of CHI*.

KARYPIS, G. 2001. Experimental evaluation of item-based top-*n* recommendation algorithms. In *Proceedings of the ACM Conference on Information and Knowledge Management*. ACM, New York.

KITTS, B., FREED, D., AND VRIEZE, M. 2000. Cross-sell: A fast promotion-tunable customer–item recommendation method based on conditional independent probabilities. In *Proceedings of ACM SIGKDD International Conference.* , ACM, New York, 437–446.

KONSTAN, J., MILLER, B., MALTZ, D., HERLOCKER, J., GORDON, L., AND RIEDL, J. 1997. GroupLens: Applying collaborative filtering to Usenet news. *Commun. ACM 40*, 3, 77–87.

LIN, W., ALVAREZ, S., AND RUIZ, C. 2000. Collaborative recommendation via adaptive association rule mining. In *Proceedings of the International Workshop on Web Mining for E-Commerce (WEBKDD'2000)*.

MCJONES, P. AND DETREVILLE, J. 1997. Each to each programmer's reference manual. Tech. Rep. 1997-023, Systems Research Center. http://research.compaq.com/SRC/eachmovie/.

MOBASHER, B., COOLEY, R., AND SRIVASTAVA, J. 2000. Automatic personalization based on web usage mining. *Commun. ACM 43*, 8, 142–151.

MOBASHER, B., DAI, H., LUO, T., NAKAGAWA, M., AND WITSHIRE, J. 2000. Discovery of aggregate usage profiles for web personalization. In *Proceedings of the WebKDD Workshop*.

MOVIELENS 2003. Available at http://www.grouplens.org/data.

RESNICK, P. AND VARIAN, H. R. 1997. Recommender systems. *Commun. ACM 40*, 3, 56–58.

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*.

SALTON, G. 1989. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, Mass.

SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. 2000. Analysis of recommendation algorithms for e-commerce. In *Proceedings of ACM E-Commerce*. ACM, New York.

SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW10*.

SCHAFER, J., KONSTAN, J., AND RIEDL, J. 1999. Recommender systems in e-commerce. In *Proceedings of ACM E-Commerce*. ACM, New York.

SENO, M. AND KARYPIS, G. 2001. Lpminer: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *Proceedings of the IEEE International Conference on Data Mining*. Also available as a UMN-CS technical report, TR# 01-026.

SHARDANAND, U. AND MAES, P. 1995. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the ACM CHI'95 Conference on Human Factors in Computing Systems*. ACM, New York, 210–217.

TERVEEN, L., HILL, W., AMENTO, B., MCDONALD, D., AND CRETER, J. 1997. PHOAKS: A system for sharing recommendations. *Commun. ACM 40*, 3, 59–62.

UNGAR, L. H. AND FOSTER, D. P. 1998. Clustering methods for collaborative filtering. In *Workshop on Recommendation Systems at the 15th National Conference on Artificial Intelligence*.

Received January 2003; revised June 2003; accepted September 2003