Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 03-002

Item-Based Top-N Recommendation Algorithms

Mukund Deshpande and George Karypis

January 20, 2003

Item-Based Top-N Recommendation Algorithms*

Mukund Deshpande and George Karypis

University of Minnesota, Department of Computer Science
Minneapolis, MN 55455

{deshpand, karypis@cs.umn.edu}

Abstract

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 *N* 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 that in typical commercial applications can grow to be several millions. To address these scalability concerns item-based recommendation techniques have been developed that analyze the user-item matrix to identify relations between the different items, and use these relations to compute the list of recommendations.

In this paper we present one such class of item-based recommendation algorithms that first determine the similarities between the various items and then used 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 nine real datasets show that the proposed 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.

*This work was supported by NSF CCR-9972519, EIA-9986042, ACI-9982274, ACI-0133464, and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Related papers are available via WWW at URL: **http://www.cs.umn.edu/~karypis**

1 Introduction

The explosive growth of the world-wide-web and the emergence of e-commerce has led to the development of *recommender systems* [27]. Recommender systems is a 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 [36, 19, 21, 37, 34, 20, 26, 8], 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.

Various approaches for recommender systems have been developed that utilize either demographic, content, or historical information [19, 6, 7, 36, 37, 21]. Collaborative Filtering (CF), is probably the most successful and widely used techniques for building recommender systems [28, 21]. For each user, CF-based recommender systems use historical information to identify a neighborhood of people that in the past have exhibited similar behavior (*e.g.*, accessed the same type of information, purchased a similar set of products, liked/disliked a similar set of movies) and then analyze this neighborhood to identify new pieces of information that will be liked by the user. We will refer to this class of approaches as ***user-based recommendation algorithms***.

Despite their success, CF-based recommender systems have two major limitations. The first is related to sparsity and the second is related to scalability [34]. In many recommender systems, the amount of historical information for each user and for each item is often quite limited. As a result, CF-based recommender systems cannot accurately compute the neighborhood and identify the items to recommend–leading to poor recommendations. To address this problem, a variety of techniques that use either dimensionality reduction [32, 31] or content-based software agents to automatically generate ratings [15] have been developed that increase the density of the datasets.

Unfortunately, nearest neighbor algorithms require computations that grows linearly with the number of users and items. With millions of users and items, existing CF-based recommender systems suffer serious scalability problems. 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 [38, 26]. These approaches, even though they can significantly speed up the recommendation engine, they tend to decrease the quality of the recommendations. An alternate approach is to build recommendation models that are based on the items. In these approaches, the historical information is analyzed to identify relations between the 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) [9, 29, 39, 20]. These approaches, since they use the pre-computed model, can quickly recommend a set of items, and have been shown to produce recommendation results that in some cases are comparable to traditional, neighborhood-based CF recommender systems. We will refer to this class of approaches as ***item-based recommendation algorithms***.

In this paper we present one such class of model-based *top-N* recommendation algorithms. These algorithms first determine the similarities between the various items and then use 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. In particular, we present two different methods of computing the item-to-item similarity. The first method models the items as vectors in the user space, and uses the *cosine* measure to measure the similarity. The second method computes the item-to-item similarities using a technique inspired by the conditional probability between two items, extended so that it can differentiate between users with varying amounts of historical information as well as differentiate between frequently and infrequently purchased items. We present a method of combining these similarities that accounts for item-neighborhoods of different density, that can incorrectly bias the overall recommendation. Finally, we present a class of higher-order item-based algorithms that obtain the final recommendations by

exploiting relations between sets of items. We experimentally evaluate our algorithms on two classes of datasets. The first class consisted of nine real datasets arising in various applications, whereas the second class consisted of 36 different datasets that were synthetically generated. Our experiments show that the item-to-item based algorithms are up to two orders of magnitude faster than the traditional user-neighborhood based recommender system while achieving comparable or substantially higher quality.

The rest of this paper is organized as follows. Section 2 presents an overview of the traditional user-based *top-N* recommendation algorithms and provides a survey of various item-based algorithms. Sections 3 and 4 describe the various phases and algorithms used in our first- and higher-order item-based *top-N* recommendation system. Section 5 provides the experimental evaluation of the various parameters of the proposed algorithms and compares it against the user-based algorithms. Finally, Section 6 provides some concluding remarks and an outline of the future research.

2 Background, Motivation, and Related Research

In the last five years several collaborative filtering/recommender systems have been developed in the academic as well as commercial domain. Recommender system is one of the few data mining techniques that is widely used in the real world and is available in several commercial products. The term *collaborative filtering* was first coined about a decade ago in [14], where it was used to describe an email filtering system called Tapestry. Tapestry was designed to filter emails received via mailing list and newsgroup postings; each user could annotate (write a comment) email messages, these annotations were shared among a group of users. A user could filter email messages by writing queries on these annotations. Though Tapestry allowed an individual user to benefit from annotations made by other users, hence the name collaborative filtering, the system was not automated and required an individual user to write complicated queries. The first system to generate automated recommendations was the GroupLens system [28, 21], it provided users with personalized recommendations on use-net 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. An excellent survey of different recommender systems for various applications can be found in [34, 27].

2.1 User-Based Collaborating Filtering

User-based collaborative filtering is the most successful technology for building recommender systems to date, and is extensively used in many commercial recommender systems. These schemes rely on the fact that each person belongs to a larger group of similarly-behaving individuals. Consequently, items (*i.e.*, products) frequently purchased by the various members of the group can be used to form the basis of the recommended items. A detailed survey of different user-based algorithms and a comparison of their performance can be found in [18, 31].

Let R be an $n \times m$ user-item matrix containing historical purchasing information of n customers on m items. In this matrix, $r_{i,j}$ is one if the ith customer has purchased the jth item, and zero otherwise. Let U be the set of items that have already been purchased by the customer for which we want to compute the *top-N* recommendations. We will refer to this customer as the *active* customer and in order to simplify the presentation we will assume that the active customer does not belong to the n customers stored in matrix R. User-based CF recommender systems compute the *top-N* recommended items for that customer as follows.

First they identify the k most similar customers in the database. This is often done by modeling the customers and items with the vector-space model, widely used for information retrieval [30, 29, 31]. In this model each of the n customers as well as the active customer is treated as a vector in the m-dimensional item space, and the similarity between the active and the existing customers is measured by computing the cosine between these vectors. Once this set of the k most similar customers have been discovered, their corresponding rows in R are aggregated to identify the set C of items purchased by the group as well as their frequency. Using this set, user-based CF techniques then recommend the N most frequent items in C that are not already in U (*i.e.*, the active user has not already purchased).

Note that the frequency of the items in the set C can be computed by either just counting the actual occurrence frequency or by first normalizing each row of R to be of the same length (*i.e.*, $||r_{i,*}||_2 = 1$). This latter normalization gives less emphasis to items purchased by customers that are frequent buyers and leads to somewhat better results.

Despite the popularity of user-based CF 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 that 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 engines can be increased by increasing the number of servers running the recommendation engine, they cannot decrease the latency of each *top-N* recommendation that is critical for near real-time performance.

2.2 Item-Based Collaborating Filtering

To address the scalability concerns of user-based recommendation algorithms, item-based recommendation techniques (also known as model-based) have been developed. These approaches analyze the user-item matrix to identify relations between the different items, and then use these relations to compute the list of *top-N* recommendations. The key motivation behind these schemes is that a customer will more likely purchase items that are similar or related to the items that he/she has already purchased. Since these schemes do not need to identify the neighborhood of similar customers when a recommendation is requested, they lead to much faster recommendation engines. A number of different schemes have been proposed to compute the relations between the different items based on either probabilistic approaches or more traditional item-to-item correlations.

One of the first item based recommender system was developed by Billus *et al* [9]. In this system the recommendation task was thought of as a classification problem, in which the goal was to classify the items purchased by an individual user into two classes *like* (user likes the item) and *dislike* (user dislikes the item). A classification model 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; the attribute value was decided based on if the corresponding user liked/disliked the item. To reduce the dimensionality a singular value decomposition (SVD) of user-item matrix was obtained. The neural network technique was used for building the classification model. The prediction on an item were obtained by constructing an example that item and feeding it to the classifier. The authors report considerable improvement 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 might not be scalable to large databases.

In [29], Breese *et al*. present two item-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 can be estimated from the clustering solution and the probability distribution of items in the cluster. The clustering solution for this technique is estimated 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 its rating. The paper contains a detailed comparison of these two item based approaches with the user based approaches. The Bayesian networks model outperformed the clustering model as well as the user based approaches. In [17] proposes a recommendation algorithm that uses dependency network instead of Bayesian networks. Though the accuracy of dependency networks is inferior to Bayesian networks they are more efficient to learn, and have a smaller memory footprint.

Agrawal *et al* [39] present a graph based recommendation algorithm where the users are represented as nodes of

the 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 a graph. The graph representation of the model allows it capture transitive relations which cannot be captured by nearest neighbor algorithms, the authors report better performance than the user based models. Sarwar *et al* [33] present an item based recommendation algorithm specifically for computing predictions of items. The predictions are computed by computing similarities between two item vectors in the user-item matrix. The paper discusses three similarity measures, and two techniques for aggregating predictions. The authors report an considerable improvement in performance over the user-based algorithms.

A number of different item-based approaches have been developed that use association rules. In [13] association rules are used for obtaining *top-N* recommendations. Given a basket of items, for an active user, the association rules are ordered based on similarity of the items making up the rule and the items in the basket. The recommendations are computing by choosing the items present in the consequent of the top ranked association rules. The similarity between a rule and active user's basket is defined as the product of the confidence of the rule and the Euclidean distance between items in the antecedent of association rule and the items in the user's basket. This approach is compared with dependency networks based recommendation algorithm described [17]. The author further extends this approach by first clustering the items and then discovering association rules on these item clusters. The association rule based algorithm outperforms the recommender system based on dependency networks [17], however the cosine based item similarity technique outperforms the association rule based scheme. Lin *et al* [22] propose a similar approach using association rules, in their approach they ensure that association rules corresponding to all items in the database are discovered. Mobasher *et al* [25] also use association rules to recommend webpages to a user based on the pages visited by that user.

3 Item-Based *top-N* Recommendation Algorithms

In this section we present a class of item-based *top-N* recommendation algorithms that use item-to-item similarity to compute the relations between the items. During the model building phase, for each item j, the k most similar items $\{j_1, j_2, \ldots, j_k\}$ are computed, and their corresponding similarities $\{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$ are recorded. Now, for each customer that has purchased a set (*i.e.*, basket) U of items, this information is used to compute the *top-N* recommended items as follows. First, we identify the set C of candidate recommended items by taking the union of the k most similar items for each item $j \in U$, and removing from the union any items that are already in U. Then, for each item $c \in C$ we compute its *similarity* to the set U as the sum of the similarities between all the items $j \in U$ and c, using only the k most similar items of j. Finally, the items in C are sorted in non-increasing order with respect to that similarity, and the first N items are selected as the *top-N* recommended set.

3.1 Item Similarity

The critical step in the proposed item-based recommendation algorithm is the method used to determine the similarity between the items. In the rest of this section we describe two different classes of similarity algorithms that we developed. One is derived from the vector-space model and the other is derived from probabilistic methods.

3.1.1 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* measure 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 v and u is defined as the cosine of the n dimensional vectors corresponding to the vth and uth column of matrix R. The cosine between these vectors is given by

$$\text{sim}(v, u) = \cos(\vec{v}, \vec{u}) = \frac{\vec{v} \cdot \vec{u}}{||\vec{v}||_2 ||\vec{u}||_2}, \tag{1}$$

where '$\cdot$' denotes the vector dot-product operation.

From Equation 1 we can see that the similarity between two items will be high if each customer that purchases one of the items also purchases the other item as well. Furthermore, one of the important feature of the cosine-based similarity is that it takes into account the purchasing frequency of the different items (achieved by the denominator in Equation 1). As a result, frequently purchased items will tend to be similar to other frequently purchased items and not to infrequent purchased items, and vice versa. This is important as it tends to eliminate obvious recommendations, *i.e.*, recommendations of very frequent items, as these items will tend to be recommended only if other frequently purchased items are in the current basket of items.

As it was the case with the user-based recommendation algorithms, the rows of R can either correspond to the original binary purchase information, or it can be scaled so that each row is of unit length (or any other norm), so that to differentiate between customers that buy a small or a large number of items. Depending on how the customers are represented, the cosine-based item similarity will be different. In the first case, for any pair of items, each customer will be treated equally, whereas in the second case, more importance will be given to customers that have purchased fewer items. The motivation for the second scheme is that co-purchasing information for customers that have bought few items tends to be more reliable than co-purchasing information for customers that buy many items, as the first group tends to represent consumers that are focused in certain product areas.

3.1.2 Conditional Probability-Based Similarity

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

$$P(u|v) = \frac{\text{Freq}(uv)}{\text{Freq}(v)},$$

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

One of the limitations of using conditional probabilities as a measure of similarity, is that each item v, will tend to have high conditional probabilities to items that are being purchased frequently. That is, quite often $P(u|v)$ is high, as a result of the fact that u occurs very frequently and not because v and u tend to occur together. This problem has been recognized earlier by researchers in information retrieval as well as recommendation systems [30, 29, 20, 10]. One way of correcting this problem is to divide $P(u|v)$ with a quantity that depends on the occurrence frequency of item u. 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 $P(u|v)$ by $-\log_2(P(u))$ [30], whereas the other one divides $P(u|v)$ by $P(u)$ [20].

Our experiments have shown that this scaling greatly affects the performance of the recommender system; furthermore, the *optimal* scaling degree is problem dependent. For these reasons, we use the following formula to compute the similarity between two items:

$$\text{sim}(v, u) = \frac{\text{Freq}(uv)}{\text{Freq}(v) \times (\text{Freq}(u))^{\alpha}}, \tag{2}$$

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

One of the limitations of using Equation 2 is that it provides no mechanism by which to discriminate between customers who purchase many items and customers who purchase few items. As discussed in Section 3.1.1, customers

that buy fewer items may be more reliable indicators when determining the similarity between items. For this reason we have extended the similarity measure of Equation 2 in the following way. First we normalize each row of matrix R to be of unit length. Then we define the similarity between items v and u as:

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

The only difference between Equation 3 and Equation 2 is that instead of using the co-occurrence frequency we use the sum of the corresponding non-zero entries of the uth 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; thus, giving emphasis to the purchasing decisions of the customers that have bought fewer items.

3.2 Similarity Normalization

Recall from Section 3 that given a basket of items U, the item-based *top-N* recommendation algorithm determines the items to be recommended by computing the similarity of each item not in U to all the items in U and selecting the N most similar items as the recommended set. The similarity between the set U and an item $v \notin U$ is determined by adding the similarities between each item $u \in U$ and v (if v is in the k most similar items of u).

One of the potential drawbacks of this approach is that the raw similarity between each item u 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 3.1, for each item u we first normalize the similarities so that they add-up to one. As the experiments presented in Section 5 show, this often lead to dramatic improvements in *top-N* recommendation quality.

3.3 Computational Complexity

The computational complexity of the item-based *top-N* recommendation algorithm depends on the amount of time required to build the model (*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 v to all the other items and then select the k most similar items. The upper bound on the complexity of this step is $O(m^2n)$, as we need to compute $m(m-1)$ similarities, each potentially requiring n operations. However, that 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 in general 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 purchased tend to be clustered. Consequently, by using sparse data structures to store R, and computing the similarities only between pairs of items that are purchased by at least one customer we can substantially reduce the computational complexity.

Finally, the amount required to compute the *top-N* recommendations for a given basket U is given by $O(k|U|)$, because we need to access the k most similar items for each of the items in U, and identify the overall N most similar items.

4 Higher-Order Item-Based *top-N* Recommendation Algorithms

Our discussion so far has been focused on item-based *top-N* recommendation algorithms in which the recommendations were computed by taking into account relations between pairs of items, *i.e.*, 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 in cases in which 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. This is done by extending the approach described in Section 3 as follows. During the model building phase for each combination of items $\{q_1, q_2, \ldots, q_r\}$ up to size l (*i.e.*, $r \leq l$), the k most similar items $\{j_1, j_2, \ldots, j_k\}$ are computed and their corresponding similarities $\{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$ are recorded. Now, for each customer that has purchased a set of items U, we first, identify the set C of candidate recommended items by taking the union of the k most similar items for all itemsets of U up to size l (*i.e.*, $\{q_1, q_2, \ldots, q_r\}$ with $q_i \in U$ and $r \leq l$), and removing from the union any items that are already in U. Then, for each item $c \in C$ using the pre-computed k highest itemset-item similarities, we compute its *similarity* as the sum of the similarities between all the itemsets of U up to size l and c. Finally, the items in C are sorted in non-increasing order with respect to that similarity, and the first N items are selected as the *top-N* recommended set.

Essentially, in this approach, during model building instead of determining the k most similar items only for each individual item, we do so for all possible itemsets up to a particular size l. During the recommendation time, we combine 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***. Note that when $l = 1$, the above scheme becomes identical to the one described in Section 3 and for this reason we will sometimes refer to it as the first-order model. Note that the name *interpolated* was motivated by the interpolated Markov models used in DNA sequence analysis [12] 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).

4.1 Itemset-Item Similarity

As it was the case with 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 3.1.

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

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

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 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 Equation 1. In the case of the conditional-probability-based approach, the similarity is computed using an approach similar to Equation 3 as follows:

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

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, $r_{i,q_1} = r_{i,q_2} = \cdots = r_{i,q_r}$.

4.2 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* [1, 2] developed by the data mining community, we use computationally efficient algorithms [1, 4, 16, 35] to find all frequent itemsets up to size l that occur in $\sigma\%$ 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*.

This 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, whereas a low value may lead to an exponentially large number of itemsets. Unfortunately, there are no good ways to a priori select the value of support, and may require extensive experimentation to obtain a good balance between computational efficiency and *top-N* recommendation quality.

Second, since our higher-order models now contain information only 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 2) that have been shown to actually perform worse [13] than the first-order item-based schemes described in Section 3. 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. This is because, 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 to any frequent itemset, it 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 3.2) and are used as the k-most similar items for that itemset. Thus,

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.

5 Experimental Results

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

5.1 Experimental Design and Metrics

The goal of our experiments was to evaluate the quality and performance of the *top-N* recommendations provided by the various recommender algorithms. In order 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 non-zero entries of each row to be part of the test set, and used the remaining entries for training[1]. Then for each customer/user we obtained the *top-N* recommendations by using the items present in the training set as the *basket* for that customer/user. In the case of the item-based algorithms, the *top-N* recommendation were computed using only the training set to build the item similarity models. Similarly, in the case of 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 customer/user. We computed two quality measures that we will refer to them as the ***recall*** and ***uninterpolated precision***. Both of these measures were motivated by similar measures used to evaluate information retrieval systems [5], but were modified to be applicable within the context of *top-N* recommendation algorithms.

If n is the total number of customers/users, the recall of the recommendation algorithm was computed as:

$$recall = \frac{Number\ of\ hits}{n}. \tag{5}$$

A recall value of 1.0 indicates that the algorithm was able to always recommend the hidden item, whereas a recall value of 0.0 indicates that the algorithm was not able to recommend any of the hidden items. One limitation of the recall 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 uninterpolated precision measure that rewards each hit based on where it occurred in the *top-N* list. If N is the number of items that are recommended, and 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 uninterpolated precision or UIP for short is equal to

$$UIP = \frac{\sum_i 1/p_i}{n}. \tag{6}$$

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 UIP is equal to the recall and occurs when all the hits occur in the first position, whereas the lowest value of the UIP is equal to $Recall/N$ when all the hits occur in the last position in the list of the *top-N* recommendations.

In order to ensure that our results were statistically accurate, for each of the experiments we performed ten different

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

runs, each time using a different random partitioning into training and test. The results reported in the rest of this section are the averages over these ten trials. Finally, in all of experiments we used $N = 10$, as the number of items top be recommended by the *top-N* recommendation algorithms.

5.2 Evaluation on Real Datasets

We evaluated the performance of the different *top-N* recommendation algorithms using nine different datasets whose characteristics are shown in Table 1. For each user-item matrix R, the columns labeled "No. Rows", "No. Columns", and "No. of Non-Zeros" show the number of customers/users, number of items, and total number of transactions, respectively. Finally, the column labeled "Density" shows the percentage of non-zero entries in the user-item matrix, and the column labeled "Avg. Basket Size" shows that average number of items in each transaction.

Name	No. Rows	No. Columns	No. NonZeros	Density	Avg. Basket Size
catalog1	58565	502	209715	0.71%	3.58
catalog2	23480	55879	1924122	0.15%	81.95
catalog3	58565	39080	453219	0.02%	7.74
ccard	42629	68793	398619	0.01%	9.35
ecommerce	6667	17491	91222	0.08%	13.68
em	8002	1648	769311	5.83%	96.14
ml	943	1682	100000	6.31%	106.04
msweb	32711	2000	98654	0.15%	3.02
skills	4374	2125	82612	0.89%	18.89

Table 1: The characteristics of the various datasets used in evaluating the *top-N* recommendation algorithms.

The *catalog1*, *catalog2*, and *catalog3* datasets correspond to the catalog purchasing transactions of two major mail-order catalog retailer. Note that *catalog1* and *catalog3* correspond to the same set of transactions, but they differ on what constitutes an item. The items of the *catalog3* dataset correspond to individual SKU numbers, whereas the items of the *catalog1* dataset correspond to the top-level product categories, that is, a particular non-zero entry in the user-item matrix is a transaction indicating that a particular user has purchased an item from a particular product category. The *ecommerce* dataset corresponds to web-based purchasing transactions of an e-commerce site. The *ccard* dataset corresponds to the store-branded credit card purchasing transactions of a major department store. The *skills* dataset corresponds to the IT-related skills that are present in the resumes of various individuals and were obtained from a major online job portal. The *msweb* dataset corresponds to web-pages that were accessed by different users at Microsoft's website and was obtained from the machine learning datasets available at UCI [24]. For the purpose of our experiments we used only the web-accessed that define the training set. Finally, the *em* and *ml* datasets correspond to movie ratings and were obtained from the *EachMovie* [23] and the *MovieLens* [11] research projects, respectively. Note that in our experiments, we ignored the actual ratings in these two datasets.

5.2.1 Effect of Similarity Normalization

Our first experiment was designed to evaluate the effect of the similarity normalization that is discussed in Section 3.2. Table 2 shows the recall and UIP 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 "NoSNorm') 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.

	Top-10 Recall					
	Cosine			Conditional Probability		
	SNorm	NoSNorm	SNorm/NoSNorm	SNorm	NoSNorm	SNorm/NoSNorm
catalog1	0.406	0.396	2.53%	0.415	0.404	2.72%
catalog2	0.147	0.143	2.80%	0.154	0.127	21.26%
catalog3	0.534	0.529	0.95%	0.540	0.515	4.85%
ccard	0.162	0.160	1.25%	0.176	0.167	5.39%
ecommerce	0.170	0.166	2.41%	0.174	0.174	0.00%
em	0.407	0.400	1.75%	0.405	0.395	2.53%
ml	0.271	0.264	2.65%	0.272	0.249	9.24%
msweb	0.520	0.515	0.97%	0.526	0.503	4.57%
skills	0.370	0.358	3.35%	0.373	0.313	19.17%
Average			2.07%			7.75%

	Top-10 UIP					
	Cosine			Conditional Probability		
	SNorm	NoSNorm	SNorm/NoSNorm	SNorm	NoSNorm	SNorm/NoSNorm
catalog1	0.208	0.203	2.53%	0.213	0.206	3.53%
catalog2	0.070	0.069	1.30%	0.074	0.064	16.43%
catalog3	0.315	0.310	1.63%	0.320	0.303	5.57%
ccard	0.119	0.118	1.11%	0.130	0.126	3.15%
ecommerce	0.096	0.093	2.41%	0.098	0.097	0.54%
em	0.189	0.186	1.75%	0.189	0.183	3.19%
ml	0.119	0.115	3.59%	0.119	0.110	8.00%
msweb	0.294	0.285	2.98%	0.310	0.289	7.49%
skills	0.178	0.172	3.35%	0.178	0.151	18.18%
Average			2.29%			7.34%

Table 2: The effect of the similarity normalization on the recommendation quality achieved by the first-order cosine- and conditional-probability-based recommendation algorithms.

Looking at the results in Table 2, we can see that the algorithms that use similarity normalization achieve better results (either in terms of recall or UIP) compared to their counterparts that do not. To facilitate these comparisons, we computed the relative percentage improvements achieved by the schemes that use similarity normalization over those that do not. These results are shown in the columns of Table 2 labeled "SNorm/NoSNorm", and the average improvement over all nine datasets is shown in the last row of each table. As we can see, 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. The performance in terms of recall of the cosine-based scheme improves by 0% to 3.35% with an average improvement of 2.07%, whereas the performance of the conditional probability-based scheme improves by 0% to 21.26% with an average improvement of 7.75%. Similar trends are observed when comparing the performance of the various algorithms using the UIP measure. Due to this performance advantage, in the rest of our experiments we will always use similarity normalization.

5.2.2 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 3 shows the recall and UIP 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 "NoRNorm") whereas the other normalizes them (those labeled "RNorm"). 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 3 we can see that the row-normalized version performs in general better for most datasets for both the cosine- and the conditional probability-based schemes. The only exception is the *ccard* dataset for which the cosine-based scheme that does not use row normalization leads to substantially better results. Also, as it was the case with the similarity normalization results shown in Section 5.2.1, the improvements of row-normalization are

	Top-10 Recall					
	Cosine			Conditional Probability		
	RNorm	NoRNorm	RNorm/NoRNorm	RNorm	NoRNorm	RNorm/NoRNorm
catalog1	0.406	0.406	0.00%	0.415	0.406	2.22%
catalog2	0.147	0.143	2.80%	0.154	0.143	7.69%
catalog3	0.534	0.536	-0.37%	0.540	0.536	0.75%
ccard	0.162	0.179	-9.50%	0.176	0.179	-1.68%
ecommerce	0.170	0.173	-1.73%	0.174	0.173	0.58%
em	0.407	0.395	3.04%	0.405	0.395	2.53%
ml	0.271	0.261	3.83%	0.272	0.260	4.62%
msweb	0.520	0.519	0.19%	0.526	0.519	1.35%
skills	0.370	0.344	7.56%	0.373	0.344	8.43%
Average			0.65%			2.94%

	Top-10 UIP					
	Cosine			Conditional Probability		
	RNorm	NoRNorm	RNorm/NoRNorm	RNorm	NoRNorm	RNorm/NoRNorm
catalog1	0.208	0.208	0.00%	0.213	0.208	2.42%
catalog2	0.070	0.069	0.05%	0.074	0.069	6.81%
catalog3	0.315	0.317	-0.71%	0.320	0.317	0.75%
ccard	0.119	0.132	-9.74%	0.130	0.132	-1.94%
ecommerce	0.096	0.097	-1.38%	0.098	0.097	0.76%
em	0.189	0.186	1.73%	0.189	0.186	1.66%
ml	0.119	0.112	6.74%	0.119	0.112	6.81%
msweb	0.294	0.301	-2.40%	0.310	0.301	3.10%
skills	0.178	0.165	8.01%	0.178	0.165	7.98%
Average			0.25%			3.15%

Table 3: The effect of row normalization on the recommendation quality achieved by the cosine- and conditional-probability-based recommendation algorithms.

more pronounced for the conditional probability-based scheme. The average improvement in terms of recall for all nine datasets is 0.65% for the cosine- and 2.94% for conditional probability-based scheme. Similar observations can be made by looking at the UIP results as well. Because of the consistent improvements achieved in the majority of the datasets, in the rest of our experiments we will always use row normalization.

5.2.3 Model Size Sensitivity

Recall from Section 3 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 recall and UIP for these experiments are shown in Table 4 for the cosine- and conditional probability-based algorithms. 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 *catalog2* dataset for which both the recall and the UIP tend to consistently decrease as we increase k. Overall, the average recall for the cosine-based algorithm incrementally improves by 1.6%, 0.0%, 0.0%, and 0.1% as we vary k from 10 to 50 items; whereas in the case of the conditional probability-based algorithm the average incremental improvements in recall are 0.4%, 0.0%, 0.1%, and 0.1%. Similar small improvements are achieved in terms of UIP as well. 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 is 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

	Top-10 Recall									
	Cosine					Conditional Probability				
	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$
catalog1	0.400	0.406	0.408	0.409	0.409	0.409	0.415	0.417	0.418	0.418
catalog2	0.152	0.147	0.143	0.140	0.139	0.168	0.154	0.146	0.142	0.139
catalog3	0.529	0.534	0.536	0.538	0.538	0.535	0.540	0.542	0.542	**0.543**
ccard	0.159	0.162	0.164	0.164	0.165	0.173	0.176	0.177	0.177	0.178
ecommerce	0.166	0.170	0.171	0.172	0.172	0.171	0.174	0.175	0.175	0.175
em	0.379	0.407	0.403	0.404	0.406	0.377	0.405	0.401	0.401	0.401
ml	0.266	0.271	0.272	0.270	0.269	0.275	0.272	0.269	0.268	0.267
msweb	0.515	0.520	0.522	0.522	0.523	0.523	0.526	0.526	0.527	0.527
skills	0.366	0.370	0.371	0.376	0.379	0.376	0.373	0.371	0.371	0.374

	Top-10 UIP									
	Cosine					Conditional Probability				
	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$	$k = 10$	$k = 20$	$k = 30$	$k = 40$	$k = 50$
catalog1	0.205	0.208	0.209	0.209	0.209	0.210	0.213	0.214	0.214	0.214
catalog2	0.072	0.070	0.069	0.068	0.068	0.079	0.074	0.072	0.071	0.070
catalog3	0.311	0.315	0.316	0.317	0.317	0.316	0.320	0.321	0.321	0.321
ccard	0.118	0.119	0.120	0.120	0.120	0.128	0.130	0.130	0.129	0.130
ecommerce	0.093	0.096	0.096	0.096	0.096	0.096	0.098	0.098	0.098	0.097
em	0.180	0.189	0.190	0.193	0.197	0.181	0.189	0.190	0.192	0.193
ml	0.114	0.119	0.119	0.120	0.119	0.117	0.119	0.118	0.118	0.120
msweb	0.289	0.294	0.294	0.294	0.294	0.306	0.310	0.310	0.311	0.311
skills	0.173	0.178	0.180	0.182	0.183	0.177	0.178	0.180	0.181	0.182

Table 4: The recall and UIP 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.

further increases in k will not change the order of the *top-N* items.

5.2.4 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 extend to which the similarity to frequently purchased/occurring 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. Table 5 shows the recall and UIP achieved on the various datasets for the different values of α. For each dataset and recommendation algorithm combination, the highest value of recall and UIP that were achieved over the different values of α is shown using a bold-faced font. Also, to facilitate comparisons, the last row of each sub-table shows the recall and UIP achieved by the cosine-based scheme. Note that these results were obtained using the first-order item-based model and $k = 20$.

A number of interesting observations can be made by looking at the results shown in Table 5. 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 recall and UIP. 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 recall and UIP are fairly smooth, and follow a $\cap$-shaped curve. This suggests that the optimal value of α can to 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 recall or UIP. Moreover, since the values of α that lead to both the highest recall or UIP values are consistent for most of the datasets, we can learn 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 recall and UIP values and the properties of the various datasets used in our experiment reveal another interesting trend. If we compare the highest recall value to the recall value achieved for $\alpha = 0.0$, we see that for *catalog1*, *ccard*, *ecommerce*, *msweb*, and *catalog3*, the highest value is usually less than 3.2% better than that for $\alpha = 0.0$. On the other hand, the improvement for *skills*, *em*, *catalog2*, and *ml* ranges from 16% to 91%. Similar trends can be observed by focusing on the UIP measure. Thus, there is a well-

	Top-10 Recall								
α	catalog1	catalog2	catalog3	ccard	ecommerce	em	ml	msweb	skills
0.0	0.408	0.081	0.540	0.192	0.186	0.355	0.219	0.512	0.299
0.1	0.414	0.098	0.546	0.197	0.190	0.364	0.227	0.517	0.312
0.2	0.419	0.114	**0.549**	**0.198**	**0.191**	0.373	0.237	0.523	0.329
0.3	**0.421**	0.130	**0.549**	0.196	0.187	0.383	0.252	0.527	0.347
0.4	0.420	0.144	0.546	0.187	0.182	0.395	0.262	**0.528**	0.365
0.5	0.415	0.154	0.540	0.176	0.174	0.405	0.272	0.526	**0.373**
0.6	0.405	**0.155**	0.528	0.164	0.165	0.410	**0.276**	0.515	0.362
0.7	0.386	0.141	0.511	0.150	0.155	**0.412**	0.255	0.480	0.327
0.8	0.354	0.104	0.490	0.136	0.145	0.406	0.163	0.397	0.236
0.9	0.299	0.038	0.463	0.121	0.138	0.379	0.030	0.274	0.091
1.0	0.215	0.017	0.424	0.061	0.129	0.030	0.004	0.092	0.034
Cosine	0.406	0.147	0.534	0.162	0.170	0.407	0.271	0.520	0.370

	Top-10 UIP								
α	catalog1	catalog2	catalog3	ccard	ecommerce	em	ml	msweb	skills
0.0	0.202	0.036	0.317	0.119	0.100	0.173	0.098	0.303	0.139
0.1	0.206	0.044	0.321	0.131	0.104	0.176	0.100	0.310	0.146
0.2	0.209	0.052	0.323	0.139	**0.105**	0.178	0.106	0.316	0.154
0.3	0.212	0.061	**0.324**	**0.140**	0.104	0.181	0.112	**0.321**	0.163
0.4	**0.214**	0.069	0.323	0.136	0.102	0.185	0.116	0.318	0.173
0.5	0.213	**0.074**	0.320	0.129	0.098	0.189	**0.119**	0.310	**0.178**
0.6	0.209	**0.074**	0.312	0.121	0.092	**0.191**	0.116	0.294	0.172
0.7	0.196	0.065	0.300	0.112	0.085	**0.191**	0.102	0.267	0.149
0.8	0.171	0.044	0.281	0.101	0.078	0.188	0.059	0.204	0.097
0.9	0.125	0.014	0.253	0.088	0.073	0.163	0.009	0.106	0.034
1.0	0.061	0.007	0.210	0.028	0.067	0.005	0.001	0.034	0.012
Cosine	0.208	0.070	0.315	0.119	0.096	0.189	0.119	0.294	0.178

Table 5: The recall and UIP as a function of the item-frequency-based scaling achieved by the α parameter for conditional-probability-based recommendation algorithms.

defined 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 recall (or UIP) improvements are those datasets that according to the statistics shown in Table 1 have some of the highest densities and the largest number of items per user.

The results in Table 5 also show how the cosine- and conditional probability-based schemes compare with each other. From these results we can see that for all datasets and a wide range of α values the conditional probability-based algorithm leads to somewhat higher recalls than the cosine-based scheme. One the average, the conditional probability-based scheme does 1.0%, 2.2%, 2.3%, and 0.2% better in terms of recall for α equal to 0.3, 0.4, 0.5, and 0.6, respectively. Furthermore, if we compare the results obtained for the optimal values of α, we can see that the conditional probability-based algorithm does 5.7% and 5.4% better than the cosine-based scheme in terms of recall and UIP, respectively.

5.2.5 Model Order Sensitivity

Our experiments so far focused on first-order item-based *top-N* recommendation algorithms. However, as discussed in Section 4, 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 6 shows the recall and the UIP results obtained by using such higher-order interpolated models for both the cosine- and the conditional probability-based approaches. In particular, Table 6 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 parameter of higher-order models is the support threshold (σ) that is used by the frequent pattern discovery algorithm to identify the frequent itemsets to be used in the models. Depending on the density and the degree to which different items co-occur in the different datasets, we used different values of the support threshold for each dataset. These values are shown in the first column of Table 6 and were selected so that they lead to a reasonable number of frequent itemsets and that 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 of the 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 model are shown in the columns labeled F_2 and F_3, respectively. For all experiments k was set to 20, and for the conditional probability-based algorithms we used a value of $\alpha = 0.5$.

			Top-10 Recall						Top-10 UIP					
Name (σ)	F_2	F_3	Cos1	Cos2	Cos3	CPrb1	CPrb2	CPrb3	Cos1	Cos2	Cos3	CPrb1	CPrb2	CPrb3
catalog1 (0.1%)	868	486	0.406	0.405	0.406	0.415	0.414	0.416	0.208	0.208	0.208	0.213	0.213	0.214
catalog2 (5.0%)	764	835	0.147	0.147	0.147	0.154	0.154	0.154	0.070	0.070	0.070	0.074	0.074	0.074
catalog3 (0.05%)	2150	1437	0.534	0.535	0.535	0.540	0.540	0.540	0.315	0.316	0.315	0.320	0.320	0.321
ccard (0.01%)	3326	2056	0.162	0.162	0.162	0.176	0.175	0.175	0.119	0.118	0.119	0.130	0.128	0.129
ecommerce (0.01%)	255	112	0.170	0.170	0.170	0.174	0.174	0.174	0.096	0.095	0.096	0.098	0.098	0.098
em (20.0%)	4077	52434	0.407	**0.419**	0.416	0.405	**0.418**	0.415	0.189	**0.201**	0.200	0.189	**0.199**	0.197
ml (10.0%)	9921	87090	0.271	0.267	0.270	0.272	**0.279**	0.275	0.119	0.114	0.118	0.119	0.120	0.119
msweb (0.1%)	730	1052	0.520	0.520	0.521	0.526	0.527	0.527	0.294	0.292	0.293	0.310	0.309	0.311
skills (1.0%)	4485	16820	0.370	0.361	0.367	0.373	**0.380**	0.379	0.178	0.172	0.176	0.178	**0.184**	0.184

Table 6: The recommendation quality as a function of the order of the model that is used.

As we can see from these results, in general, higher-order item-based models do not lead to any significant improvements in either recall or UIP. For most datasets, the results obtained across the different schemes (*i.e.*, 1st, 2nd, and 3rd order models) are very similar or within less than 1% of each other. 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 *skills*, *em*, and *ml* datasets. These results are shown in Table 6 using underlining and a bold-faced font. In particular, the second-order model improved the recall in the above datasets by 1.8% to 3.2%, and the UIP by 3.3% to 6.3%. Also note that these three datasets are also 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.

5.2.6 Comparison with the User-based Recommendation Algorithm

Finally, to compare the performance of our item-based recommendation algorithms 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. These results are shown in Table 7 that shows the recall and UIP achieved by different algorithms. The user-based recommendations were obtained using the algorithm described in Section 2.1 with user-neighborhoods of size 50, and unit length normalized rows. Furthermore, 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 7 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-α = Opt" correspond to the conditional probability-based algorithm that uses for each dataset the value of α that achieved the highest performance in the experiments discussed in Section 5.2.4. All the item-based results were obtained using the first-order models. Finally, Table 7 also includes the *top-N* recommendation quality achieved by the naive algorithm, labeled "Frequent", that recommends the N most frequent items not already present in the active user's set of items.

From the results in Table 7 we can see that both the "Cosine" and the "CProb-$\alpha = 0.5$" algorithms outperform the user-based algorithm in four out of the nine datasets, whereas "CProb-α = Opt" outperforms the user-based scheme in six our of the nine datasets. To facilitate the relative comparisons between the various schemes we computed the relative percentage improvement achieved by the various item-based schemes over the user based schemes. These results are shown in the columns labeled "Cos/User", "CProb-$\alpha = 0.5$/User", and "CProb-α = Opt/User". From these results, we can see that on the average the cosine-based scheme does 1.18% and 4.24% worse than the user-based

	Top-10 Recall							
	User	Frequent	Cosine	CProb-$\alpha = 0.5$	CProb-α = Opt	Cos/User	CProb-$\alpha = 0.5$/User	CProb-α = Opt/User
catalog1	0.398	0.215	0.406	0.415	0.421	2.01%	4.27%	5.78%
catalog2	0.150	0.025	0.147	0.154	0.155	-2.00%	2.67%	3.33%
catalog3	0.494	0.030	0.534	0.540	0.549	8.10%	9.31%	11.13%
ccard	0.158	0.079	0.162	0.176	0.198	2.53%	11.39%	25.32%
ecommerce	0.178	0.029	0.170	0.174	0.191	-4.49%	-2.25%	7.30%
em	0.453	0.367	0.407	0.405	0.412	-10.15%	-10.60%	-9.05%
ml	0.281	0.131	0.271	0.272	0.276	-3.56%	-3.20%	-1.78%
msweb	0.517	0.628	0.520	0.526	0.528	0.58%	1.74%	2.13%
skills	0.384	0.238	0.370	0.373	0.373	-3.65%	-2.86%	-2.86%
Average						-1.18%	1.16%	4.59%

	Top-10 UIP							
	User	Frequent	Cosine	CProb-$\alpha = 0.5$	CProb-α = Opt	Cos/User	CProb-$\alpha = 0.5$/User	CProb-α = Opt/User
catalog1	0.206	0.080	0.208	0.213	0.214	1.02%	3.46%	4.00%
catalog2	0.076	0.009	0.070	0.074	0.074	-8.93%	-2.78%	-3.08%
catalog3	0.298	0.010	0.315	0.320	0.324	5.59%	7.14%	8.59%
ccard	0.119	0.066	0.119	0.130	0.140	-0.05%	8.59%	17.36%
ecommerce	0.095	0.012	0.096	0.098	0.105	0.32%	2.68%	10.05%
em	0.221	0.169	0.189	0.189	0.191	-14.36%	-14.42%	-13.57%
ml	0.128	0.046	0.119	0.119	0.119	-6.77%	-6.85%	-6.96%
msweb	0.324	0.311	0.294	0.310	0.321	-9.37%	-4.26%	-0.97%
skills	0.189	0.091	0.178	0.178	0.178	-5.61%	-5.63%	-5.59%
Average						-4.24%	-1.34%	1.09%

Table 7: The quality of the recommendations obtained by the naive, the item-based, and the user-based recommendation algorithm.

scheme in terms of recall and UIP, respectively; the conditional probability-based scheme with $\alpha = 0.5$ does 1.16% better and 1.34% worse than the user-based scheme in terms of recall and UIP, respectively; whereas the conditional probability-based scheme with the best choice for α does 4.59% and 1.09$ better than the user-based scheme in terms of recall and UIP, respectively. In general, all three item-based schemes seem to do worse than the user-based scheme for the denser datasets (*e.g.*, *skills*, *em*, and *ml*), and do better for the sparser datasets (*e.g.*, *ccard*, *ecommerce*, and *catalog3*). Also the performance of the item-based schemes relative to the user-based scheme is somewhat worse when measured in terms of UIP instead of recall. 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 (*i.e.*, recall) is smaller than the total number recommended by the item-based schemes. Finally, both the user- and item-based algorithms produce recommendations whose quality is substantially better than the recommendations produced by the naive "Frequent" algorithm.

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 8 shows the amount of time required by the two algorithms to compute the *top-N* recommendations for each one of the nine 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" shows the amount of time required to compute all the recommendations for each one of the dataset, and the columns labeled "RcmdRate" shows 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 8 are in seconds.

Looking at the results of Table 8 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 nine 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 17μs for the item-based algorithm, versus 1075μs for the user-based algorithm. Also, as

	User-based		Item-based		
Name	RcmdTime	RcmdRate	ModelTime	RcmdTime	RcmdRate
catalog1	62.68	934	0.10	0.16	366031
catalog2	83.53	281	19.35	1.82	12901
catalog3	13.57	4315	0.69	0.78	75083
ccard	17.59	2427	0.98	0.79	53960
ecommerce	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
msweb	24.33	1344	0.04	0.08	408887
skills	1.64	2667	0.13	0.07	62485

Table 8: The computational requirements for computing the *top-N* recommendations for both the user- and item-based algorithms.

discussed in Section 3.3, 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.

5.3 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 5.2.6 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 recommendation 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 depending on the characteristics of the 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 [3], which is widely used to mimic the transactions in the retailing 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 mean with a few large *itemsets*. Similarly, the size of the user's basket is also clustered around a mean with a few user's making lots of purchases.

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

Table 9: Parameters taken by synthetic dataset generator

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 characteristic of the synthetic dataset are shown

in Table 9. 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 take 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 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 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.

As stated earlier, in our study we evaluated 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. Toward this goal, 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 are shown in the last column of Table 9. 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 the 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 of the following. As a result of the generation approach, the generator, on the average, will combine S_u/S_{is} itemsets randomly selected from the N_{is} itemsets to form a particular transaction. Thus, 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 the average, S_u/N_{is} itemsets will be included. However, because each such itemset is now larger, that affects, to some extent, the *complexity* of the purchasing decision represented by that particular itemset.

5.3.1 Results

Table 10 shows the recall and UIP achieved both by 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 5.2.6, 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 recall 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 10 provide a comprehensive comparison of the various algorithms under a wide-range of dataset characteristics. 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, the performance (either in terms of recall or UIP) 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 5.3, 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

Avg. Size of User's Basket (S_u) = 15, Sparsity = 1.4 %												
		Top-10 Recall ***(S_{is} = 4)***					**Top-10 UIP** ***(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 Recall ***(S_{is} = 6)***					**Top-10 UIP** ***(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 Recall ***(S_{is} = 8)***					**Top-10 UIP** ***(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 Recall ***(S_{is} = 4)***					**Top-10 UIP** ***(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 Recall ***(S_{is} = 6)***					**Top-10 UIP** ***(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 Recall ***(S_{is} = 8)***					**Top-10 UIP** ***(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 Recall ***(S_{is} = 4)***					**Top-10 UIP** ***(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
		Top-10 Recall ***(S_{is} = 6)***					**Top-10 UIP** ***(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 Recall ***(S_{is} = 8)***					**Top-10 UIP** ***(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

Table 10: Accuracy on different values of S_u, N_{is} and S_{is}.

of users) remains fixed, this makes the problem of learning accurate *top-N* recommendations for each user harder. Second, as the sparsity decreases, that is S_u increases from 15 to 45, the overall performance of the different schemes decreases (assuming that S_{is} and N_{is} remains fixed). 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 various algorithms increases as we increase the average size of the itemsets (S_{is}) from four to eight. This can be explain due to the decrease in the variability of the dataset as discussed earlier. However, an interesting trend arises in terms of the performance of the user- and the first-order item-based schemes. When S_{is} is small, the 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 classes of algorithms shrinks to the point at which the 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 $S_{is} = 8$). These result suggest that when there is relatively low variability and the behavior of a user is manifested via long itemsets (*e.g.*, purchasing patterns), user-based schemes are superior to first-order item-based schemes. 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.

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 5.2.4, as larger α-values tend to work better for denser datasets. Finally, the results of Table 10 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%.

6 Conclusions and Directions for Future Research

In this paper we presented and experimentally evaluated a class of model-based *top-N* recommendation algorithm that uses 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.

We believe that the *top-N* recommender algorithms presented in this paper can be improved by combining elements from both the user- and item-based approaches. User-based approaches by dynamically computing a neighborhood of similar users are better suited to provide truly personalized information. On the other hand, item-based approaches by directly computing the similarity between items appear to compute more accurate recommendations. However, one potential limitation of item-based approaches on very large user collections, is that the globally computed item-to-item similarities may not be able to provide sufficiently degree of personalization (even when combined in the context of basket-to-item similarity). In this case, an approach that first identifies a reasonably large neighborhood of similar users and then using this subset to derive the item-based recommendation model may be able to combine the best of both worlds and perform even better recommendations.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of 1993 ACM-SIGMOD Int. Conf. on Management of Data*, Washington, D.C., 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, Santiago, Chile, 1994.

[4] Ramesh Agrawal, Charu Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, August 2000.

[5] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[6] Marko Balabanovic and Yoav Shoham. FAB: Content-based collaborative recommendation. *Communications of the ACM*, 40(3), March 1997.

[7] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the 1998 Workshop on Recommender Systems*, pages 11–15. AAAI Press, 1998.

[8] Doug Beeferman and Adam Berger. Agglomerative clustering of a search engine query log. In *Proceedings of ACM SIGKDD International Conference*, pages 407–415, 2000.

[9] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proceedings of ICML*, pages 46–53, 1998.

[10] P. Chan. A non-invasive learning approach to building web user profiles. In *Proceedings of ACM SIGKDD International Conference*, 1999.

[11] MovieLens Dataset. Available at http://www.grouplens.org/data.

[12] A. L. Delcher, D. Harmon, S. Kasif, O. White, and S. L. Salzberg. Improved microbial gene identification with glimmer. *Nucleic Acid Research*, 27(23):4436–4641, 1998.

[13] Ayhan Demiriz. An association mining-based product recommender. In *NFORMS Miami 2001 Annual Meeting Cluster: Data Mining*, 2001.

[14] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[15] N. Good, J. Scafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of AAAI*, pages 439–446. AAAI Press, 1999.

[16] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX*, May 2000.

[17] D. Heckerman, D.M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[18] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithm framework for performing collaborative filtering. In *Proceedings of SIGIR*, pages 77–87, 1999.

[19] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of CHI*, 1995.

[20] Brendan Kitts, David Freed, and Martin Vrieze. Cross-sell: A fast promotion-tunable customer-item recommendation method based on conditional independent probabilities. In *Proceedings of ACM SIGKDD International Conference*, pages 437–446, 2000.

[21] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[22] Weiyang Lin, Sergio Alvarez, and Carolina Ruiz. Collaborative recommendation via adaptive association rule mining. In *International Workshop on Web Mining for E-Commerce (WEBKDD'2000)*, 2000.

[23] Paul McJones and John DeTreville. Each to each programmer's reference manual. Technical Report 1997-023, Systems Research Center, 1997. http://research.compaq.com/SRC/eachmovie/.

[24] C.J. Merz and P.M. Murphy. UCI repository of machine learning databases, 1998.

[25] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. Automatic personalization based on web usage mining. *Communications of the ACM*, 43(8):142–151, 2000.

[26] Bamshad Mobasher, Honghua Dai, Tao Luo, Miki Nakagawa, and Jim Witshire. Discovery of aggregate usage profiles for web personalization. In *Proceedings of the WebKDD Workshop*, 2000.

[27] Resnick and Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[28] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of CSCW*, 1994.

[29] John s. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th conference on Uncertaintly in Artificial Intelligence*, pages 43–52, 1998.

[30] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[31] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of ACM E-Commerce*, 2000.

[32] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender systems–a case study. In *ACM WebKDD Workshop*, 2000.

[33] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW10*, 2001.

[34] J. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *Proceedings of ACM E-Commerce*, 1999.

[35] Masakazu Seno and George Karypis. Lpminer: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *IEEE International Conference on Data Mining*, 2001. Also available as a UMN-CS technical report, TR# 01-026.

[36] Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the ACM CHI'95 Conference on Human Factors in Computing Systems*, pages 210–217, 1995.

[37] Loren Terveen, Will Hill, Brian Amento, David McDonald, and Josh Creter. PHOAKS: A system for sharing recommendations. *Communications of the ACM*, 40(3):59–62, 1997.

[38] Lyle H. Ungar and Dean P. Foster. Clustering methods for collaborative filtering. In *Workshop on Recommendation Systems at the 15th National Conference on Artificial Intelligence*, 1998.

[39] J. wolf, C. Aggarwal, K. Wu, and P. Yu. Horting hatches and egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1999.