A Subspace Clustering Framework for Research Group Collaboration*

Nitin Agarwal Ehtesham Haque Huan Liu Lance Parsons

Department of Computer Science Engineering

Arizona State University, Tempe, AZ 85281

Abstract

Researchers spend considerable time searching for relevant papers on the topic in which they are currently interested. Often, despite having similar interests, researchers in the same lab do not find it convenient to share results of bibliographic searches and thus conduct independent time-consuming searches. Research paper recommender systems can help the researcher avoid such time-consuming searches by allowing each researcher to automatically take advantage of previous searches performed by others in the lab. Existing recommender systems were developed for commercial domains to assist users by focussing towards products of their interests. Unlike those domains, the research paper domain has relatively few users when compared with the huge number of research papers. In this paper we present a novel system to recommend relevant research papers to a user based on the user's recent querying and browsing habits. The core of the system is a scalable subspace clustering algorithm (SCuBA[1]) that performs well on the sparse, high-dimensional data collected in this domain. Both synthetic and benchmark datasets are used to evaluate the recommendation system and to demonstrate that it performs better than the traditional collaborative filtering approaches when recommending research papers.

Keywords: Collaborative Filtering, Recommender Systems, Subspace Clustering, High-Dimensional Data, Highly Sparse Data, Research Paper Domain

1 Background and Motivation

The explosive growth of the world-wide web and the emerging popularity of e-commerce has caused the collection of data to outpace the analysis necessary to extract useful information. Recommender systems were developed to help close the gap between information collection and analysis by filtering all of the available information to present what is most valuable to the user (Resnick and Varian, 1997).

One area of the web that has seen continued growth is the online publication of research papers. The number of research papers published continues to increase, and new technology has allowed many older papers to be rapidly digitized. A typical researcher must sift through a large quantity of articles manually, relying on keyword-based searches or paper citations to guide them. The search results of researchers with similar interests can help direct a more effective search, but the process of sharing search results is often too cumbersome and time consuming to be feasible. A recommender system can help by augmenting existing

*Supported by grants from Prop 301 (No. ECR A601) and CEINT 2004

[1] SCuBA: Subspace Clustering Based Analysis

search engines by recommending papers based on the preferences of other researchers with similar interests. We assume that the proposed system is localized for a typical lab setting which augments an existing search engine. Rather than performing search in some digital library, SCuBA tries to leverage the existing search results. It tries to find similar interest groups of users based on their browsing patterns and recommends research papers which might be interesting to them.

There are two main branches of recommender systems; content based filtering and collaborative filtering. Content based filtering (CBF) approaches create relationships between items by analyzing inherent characteristics of the items. Collaborative filtering (CF) systems do not analyze an items properties, but instead take advantage of information about users' habits to recommend potentially interesting items. The analysis of user behavior patterns, allows collaborative filtering systems to consider characteristics that would be very difficult for content based systems to determine such as the reputation of the author, conference, or journal. CF approaches are also well suited to handle *semantic heterogeneity*, when different research fields use the same word to mean different things.

In many domains, there is an ever increasing number of users while number of items remains relatively stable. However, in domains such as research paper recommendation, the number of users (researchers) is much less than the number of items (articles). Collaborative filtering systems face two major challenges in the research paper domain: scalability to high dimensional data and data sparsity. In a typical recommender system there are many items. For example, Amazon.com recommends specific books of interest from a large library of available books. Item-based approaches that determine similarity measures between items do not perform well since the item space is extremely large. A user based approach allows us to leverage the relatively small number of users to create an efficient algorithm that scales well with the huge number of research papers published each year. An intuitive solution used by early collaborative filtering algorithms is to find users with similar preferences to the current user and recommend other items that group of users rated highly. Even with a relatively small number users, however, this approach is computationally complex. The use of clustering algorithms to pre-determine groups of similar users has been used to significantly increase performance (Ungar and Foster, 1998; Mobasher et al., 2000).

A particular user of the system will probably purchase a very small percentage of the available books. As a result, if we consider the data as a user-item matrix, a typical row will be extremely sparse, with only a few columns containing 1's, representing purchased books. Similarly, in the domain of research papers where a researcher will be interested in articles related only to a particular research area, the row representing the researcher will also be very sparse. Notice that the number of articles is ever increasing since there are many papers published each year which further compounds the degree of sparsity. Presence of sparsity poses a problem for user-based approaches because they often rely on nearest neighbor schemes to map a new user to the existing user groups. It has been demonstrated that that the accuracy of nearest neighbor algorithms is very poor for sparse data (Sarwar et al., 2001; Demiriz, 2004). Subspace clustering is a branch of clustering algorithm that is able to find low dimensional clusters in very high-dimensional datasets. This approach to clustering allows our system to find groups of users who share a common interest in a particular field or sub-filed regardless of differences in other fields. Searching for similar users across all of the items leads to finding users who share many common interests. By finding groups based on subsets of items, subspace clustering find groups of users who share an interest in a particular area, regardless of their differences in other areas or fields. We address the issue of high-dimensionality and sparsity of the data space by proposing a new approach to collaborative filtering utilizing subspace clustering principles. Furthermore, we propose a novel subspace clustering algorithm suited to sparse, binary data.

The remainder of the paper is organized as follows. In Section 2 we present a formal definition of the problem we propose to solve. In Section 3 we give a detailed description of our proposed algorithm. In Section 4 experimental results are presented, including description of the dataset used, the evaluation metrics and discussion. In Section 5 we discuss related work in the area of subspace clustering and recommender systems. Finally Section 6 contains concluding remarks and directions for future research.

2 Problem Definitions

The proposed algorithm utilizes a binary user-item matrix, where a "*1*" in position i, j indicates that user i selected article j during a session. Each row is generated by anonymously recording the browsing pattern of users. The algorithm then finds clusters of users that are connected by subsets of papers they selected in common. The subsets of items are stored and used to make recommendations. The subsets of papers form groups of research articles that are connected by fields of interest. When a user selects a paper, the system recommends other papers that are in subsets containing the selected paper. This allows the system to respond to current, short-term interests of users that change from visit to visit.

We define the data space in the research paper domain as an $m \times n$ matrix such that there are m researchers $R = \{ r_1, r_2, ..., r_m \}$ and n articles $A = \{ a_1, a_2, ..., a_n \}$. The row r_i represents the interests of researcher i and consists of a list of articles A_{r_i} which indicates the user's interest in those articles. In the research paper domain, this could indicate that the user has read or accessed a certain article. For a given session there is an *active researcher* ($r_{active} \in R$) for which the collaborative filtering algorithm would like to recommend new articles that may be of interest to r_{active} based on the researcher's interest in the current session and the opinion of other like-minded researchers.

In order to predict which articles will be of high interest, we must have models of like-minded researchers. Given the $m \times n$ matrix, we can find like-minded researchers by finding groups of researchers $r \subseteq R$ who have expressed interest in similar articles $a \subseteq A$. The problem of finding such groups can be transformed into a problem of *subspace clustering* using the previously described binary matrix as input. The result of subspace clustering would be clusters, of researchers in corresponding subspaces of articles. Here, the underlying assumption is if a group of researchers have similar interests then they usually access similar sets of articles or vice-versa.

Problem Statement Given a binary $m \times n$ matrix, where rows represent m researchers ($R = \{r_1, r_2, ..., r_m$ $\}$) and columns represent n articles ($A = \{ a_1, a_2, ..., a_n \}$), find subspace clusters of researchers, $r \subseteq R$, defined in subspaces of articles, $a \subseteq A$.

3 Proposed Algorithm

Let us consider a large research lab setting with several researchers working on various topics. The access patterns of these researchers can be maintained by tracking the log of research papers they access. In order to preserve privacy, researchers could be assigned an encrypted form of identification for tracking purposes. From these access patterns, we can generate a researcher/article data space as defined in the previous section.

Each area of research has a group of people who are highly involved in the field, which we define as *experts* in the particular field. We believe that inferences about such experts can be drawn by analyzing the researcher article data space. More specifically, we infer that people accessing a similar set of articles are interested in the topics represented by the articles. Such a group is represented by a subspace cluster in the researcher/article data space. Finding these experts will ultimately help us achieve our goal of finding the groups of articles that form fields of interest. We adopt a subspace clustering approach to find groups of these experts.

Our proposed subspace clustering algorithm is a two-step process starting with finding subspaces that

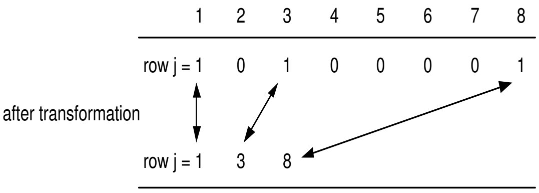


Figure 1: Transformation: A row in the original matrix is transformed into a string containing the position of the non-zero columns.

row id					
1	1	2	3	17	
2	1	2	3		
3	2	3	4	50	
4	1	2	3		
5	2	3	4		
6	2	3			
7	5	6	7	8	21
8	6	7	40		
9	5	6	7	8	26

Figure 2: Transformed Data D after being compressed

form clusters and then post-processing to remove redundancy. The output of these steps are sub-matrices of the original data matrix. Before discussing the above steps in detail, we will show how the proposed algorithm addresses the challenges associated with sparse, high-dimensional, binary-valued data in the research paper domain.

3.1 Challenges of the Domain - Addressed

Sparsity and Binary data. High sparsity means that for a given r_i, which is a vector in n dimensional space, most of the entries in the vector are zeros. Since we are only interested in the values which are not zeroes, we transform the original vector into a string containing positions of non-zero values. An example is shown in Figure-1. The result of the transformation is compact representation resulting in reduced memory requirements and less processing overhead.

High-dimensional data. In high dimensional data, the number of possible subspaces is huge. For example, if there are N dimensions in the data, the number of possible subspaces is 2^N. Hence, subspace clustering algorithms must devise an efficient subspace search strategy (Parsons et al., 2004). Most existing algorithms will work well when the number of dimensions is relatively small. In the research paper domain there are thousands of dimensions and such approaches may become impractical. For example, based on our work in (Parsons et al., 2004), we chose an efficient representative subspace clustering algorithm, MAFIA (Goil et al., 1999), to run on a data set with 1000 dimensions. The program ran out of memory because the algorithm was designed to be very efficient for datasets with many rows but comparatively few dimensions. In the research paper domain, we have a unique property that the number of rows is significantly less than the number of dimensions. We overcome this challenge of high-dimensional data by devising an algorithm that exploits the row-enumeration space rather than the larger dimension space.

By combining our exploitation of the row-enumeration space and compact representation of binary sparse data, we convert the challenges into advantages in our subspace clustering algorithm. The next two sections discuss the major steps of our algorithm in more detail.

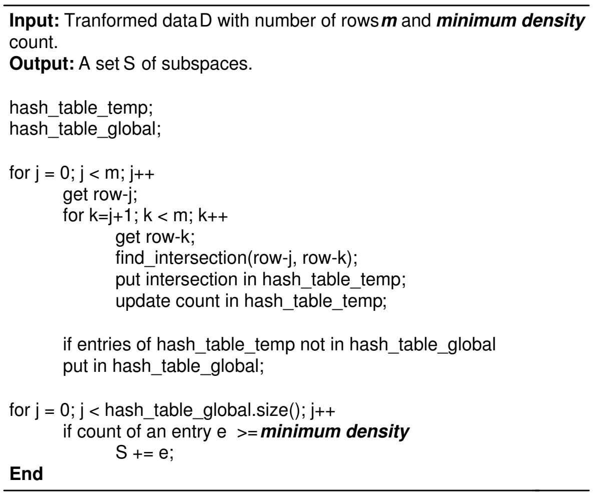
3.2 Find subspaces that form clusters.

The most straightforward way to find subspaces is to project the instance in all possible subspaces and find the ones in which clusters are formed. Such an approach is not feasible for data with a large number of dimensions since number of possible subspaces is exponential, $O(2^N)$. Therefore, we propose a search strategy which explores the smaller row-enumeration space.

Subspace Search: The result of the transformation (as shown in Figure-1) of the high-dimensional data is a list of strings representing rows in the original data. We call the new, transformed dataset *D.* An example is shown in Figure-2.

The subspace search proceeds by comparing row_i with each successive row $(row_{i+1}, row_{i+2}, row_{i+3}, ..., row_m)$. For example, if we start at row_1 in Figure-2, we first find the intersection between row_1 and row_2. The result of the intersection is *1 2 3* which represents a subspace in dimension *1, 2 and 3* with row_1 and row_2 as cluster members. *1 2 3* is stored as a key in a temporary hash table and the number of cluster members is stored as the value in the table. In addition, we also store the row-id as the values in the table in order to keep track of the cluster members for a given subspace. Next, row_1 is compared with row_3 and the intersection *2 3* is placed in the hash table. The intersection of row_1 and row_4, *1 2 3*, is already present in the table, so the count value is updated. At the end of the pass for row_1, the hash table will have two entries *1 2 3* and *2 3.* At this point, the entries in the temporary hash table are put in a global hash table which only accepts entries which are not already present in the global table and the temporary hash table is flushed. The rationale for having this rule is the following. When the search commences at row_2, it will find the intersection *1 2 3* and eventually it will update the global hash table with its local one. Notice here that the subspace *1 2 3* has already been found during the search commencing from row_1. Therefore, there is no requirement to update the global table. At the end of the search, the global hash table will have five entries, *5 6 7 8*, *1 2 3*, *2 3*, *2 3 4* and *6 7.* Notice that the subspace *2 3* is subsumed by the subspace *1 2 3* or *2 3 4.* This redundant subspace is removed in the next step. The formal description of the algorithm is shown in Figure-3.

Memory usage: The main sources of memory consumption are the temporary and global hash tables, and the transformed dataset. It should be noted that this dataset is a compressed version of the original data and thus uses significantly less memory. The hash table memory requirement grows slowly since the



```
Input: Tranformed data D with number of rows m and minimum density
count.
Output: A set S of subspaces.

hash_table_temp;
hash_table_global;

for j = 0; j < m; j++
    get row-j;
    for k=j+1; k < m; k++
        get row-k;
        find_intersection(row-j, row-k);
        put intersection in hash_table_temp;
        update count in hash_table_temp;

    if entries of hash_table_temp not in hash_table_global
    put in hash_table_global;

for j = 0; j < hash_table_global.size(); j++
    if count of an entry e  >= minimum density
        S += e;
End
```

Figure 3: Subspace Clustering Algorithm

temporary hash table is flushed every time a new search commences and the global hash table only contains previously unfound entries. Although use of a hash table may lead to some overhead of space due to unmapped entries, the advantage of constant time lookup greatly outweighs such overhead. In our algorithm we make heavy use of lookups in the hash table to check whether a subspace has already been found and to maintain counts and cluster member-ids. Generally, it was noticed that the memory requirements grew linearly and were stable during our experiments.

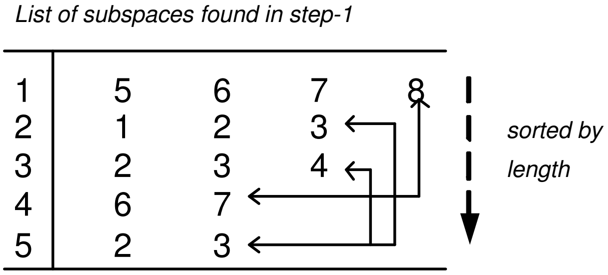
Time Complexity: The algorithm takes advantage of the fact that the number of rows, m, is relatively small. As a result, the subspace search is performed on the row enumeration space which is O(m^2). It should be noted that in our case, it is actually less than m^2 because if we are at row_i, we only look at $row_{i+1}, row_{i+2}, ..., row_m$. The algorithm also requires finding intersection of two strings which is performed in *O(k)* time where the k is the length of the strings. Notice that k is usually very small due to the high sparsity of the data. In summary, the total complexity is *O(m^2k).*

3.3 Post-processing to remove redundancy.

A larger subspace which contains several smaller subspaces covers more articles more articles within the same field of interest. Removing smaller subspaces subsumed by larger ones helps in making recommendation process faster in the absence of redundant subspace clusters.

The result from the previous step is a collection of subspaces, S. An example of such a collection is shown in Figure-4. The subspaces connected with arrows indicate two subspaces, one of which subsumes another. We must remove the fourth subspace which is *6 7* since it is subsumed by subspace *5 6 7 8.* In general, to remove the redundant/subsumed subspaces in S, we perform the following steps:

1. Sort the set S according to the size of each subspace in descending order. The result of sorting is



1	5	6	7	8
2	1	2	3	
3	2	3	4	
4	6	7		
5	2	3		

Figure 4: Set of Subspaces S with redundancy. Subspaces marked with arrows are pairs where one subspace is subsumed by another. For example, subspace 4 is subsumed by subspace 1.

shown in Figure-4.

2. Take an element s_i from the set, S, and pass through $s_{i+1}, s_{i+2}, ..., s_{|S|}$ removing any element if it is a subset of s_i .

By performing step one, we place the largest subspace as the first element in *S*. Then performing step 2, starting with the first element of the set, we will remove all subsets of s_1, in the first pass. Since s_1 is the largest subspace, without loss of generality, we can assume that there will be a large number of subsets of s_1 in *S*. As a result, |S| will shrink considerably and the remaining passes through S will be shorter, resulting in reduced running time.

The time complexity is $O(|S|^2\ p)$ where p is the size of the subspaces when computing subsumption between two subspaces. Notice that p is quite small due to sparsity and |S| is shrinking with each iteration. The time complexity for sorting is $O(|S|\lg|S|)$, so the overall complexity is still $O(|S|^2\ p)$.

3.4 Finding Overlapping Subspaces

In real world data or data which is highly sparse, it might be possible that subspace clusters in the form of sub-matrices will not be significant in number and size. In that case, we relax our subspace search so that we can find clusters of irregular shape as opposed to strict sub-matrices. These irregularly shaped clusters are larger in size and cover more of the data. Overlapping of subspace clusters can represent extended or implicit relationships between both items and users which might be more interesting to the user.

An example is shown in Figure-5. Four subspace clusters are grouped together because they share common subspaces. We apply a simple clustering algorithm to cluster the subspaces. For each element in the list of subspaces, the overlap between the given element and the other subspace clusters are found. The degree of overlap is controlled by threshold parameter. If the overlap exceeds the given threshold, the original subspace cluster is selected as a member of the cluster and is considered the *seed* of the cluster. For example in Figure-5, the seed of the new cluster found is the subspace *A B C D*. The other members of this cluster have some degree of overlap with *A B C D*. In this example, subspaces with at least one item in

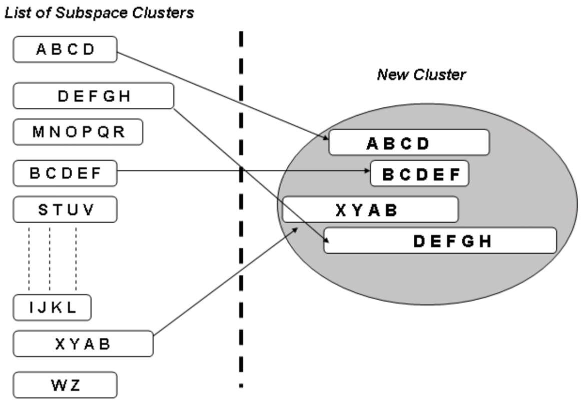


Figure 5: The subspaces are grouped together in one cluster if there is overlap between subspaces.

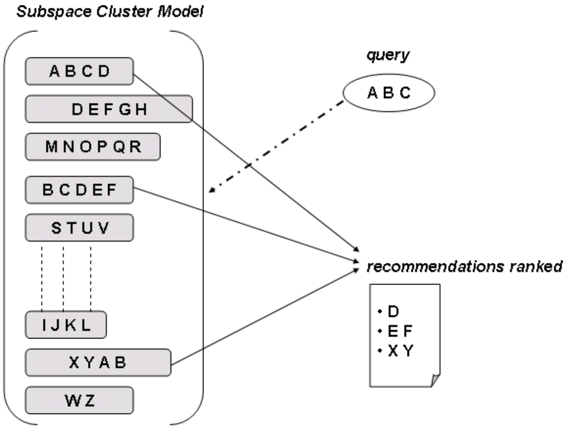


Figure 6: Using the subspace clustering model to generate recommendations.

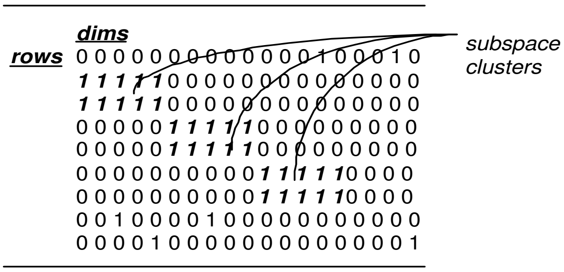
common with the seed, are put into the new cluster. This process is repeated for each element from the list of subspaces, resulting in large clusters of subspaces. A feature of this clustering process is that we allow a subspace to be member of several clusters as opposed to forming discrete partitions. This allows us to find all of the relationships that a subspace may participate in, instead of restricting it to one cluster membership.

3.5 Generating Recommendations

The process of generating recommendations illustrated in Figure-6 involves mapping a user's query to the subspaces and making recommendations based on the subspaces. The query represents the current selection of the active user. All of the subspaces containing the query item are collected and the matching subspaces are ranked based on the *coverage* of the query. Coverage is defined as the number of query items present in the subspace. The subspace with the highest coverage is ranked first and the ranked order of the subspaces determines the ranking of the recommendations. The recommendations are the elements in the subspace that are not part of the query. For example, in Figure-6 for the query $A\ B\ C$, the subspace $A\ B\ C\ D$ has the highest coverage so the recommendation from that subspace, D, is ranked first. In the case of a tie while ranking, the subspace with the higher *strength* is picked first where the strength is defined as the number of cluster members present in the subspace. In Figure-6, subspaces $B\ C\ D\ E\ F$ and $X\ Y\ A\ B$ have equal coverage. In this case, their ranking is determined by their cluster strengths.

3.6 Fall-Back Model

Since the process of generating recommendations is dependent upon the successful mapping of a query to the subspace clustering model, we consider the scenario where a query is not covered by the subspace clustering



```
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Figure 7: Example of synthetic data containing 3 subspace clusters.

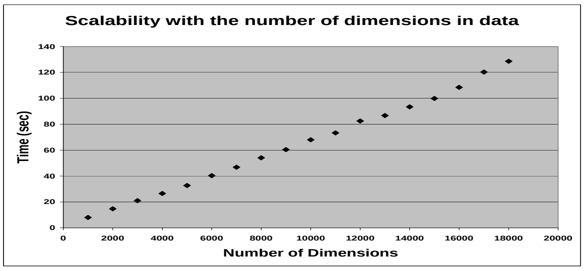


Figure 8: Scalability with the number of dimensions in the data set.

model. Although this is a rare case (see Section 4.4), there must be a mechanism to handle such a scenario if it arises.

The subspace clustering approach finds relationships between articles in the researcher/article space. There will be a number of articles which do not form any significant relationship with other articles and hence will not form subspace clusters. If a query contains such articles, the subspace clustering model will not be able to cover the query. In this case, a *fall-back model* is provided which can be used to make recommendations for the query. Although the fall-back model provides no assurance on the quality of the recommendations, it does guarantee that a given query will be covered with minimal cost for maintaining and using the model.

The fall-back model utilizes the researcher/aricle matrix directly. For a given query, the articles in the query are indexed in the matrix and the corresponding rows(researchers) are found. The articles in the rows are ranked according to their global frequency and the recommendations are made in the ranked order. This approach is similar to the user-based approach where the items in the nearest user-vector are recommended to the active user. Notice that the computational requirements of the fall-back approach are minimal, consisting mainly of indexing the query articles and ranking according to the article frequency. Article frequency information can be maintained in a table enabling inexpensive look up cost.

Finally, recommender systems generally work under the principle that a queried item exists in the user/item matrix. An interesting scenario is, when a researcher selects an article that does not exist in the researcher/article matrix. The consequence will be that both the fall-back model and the subspace clustering model will not be able to cover the query. In this case, the top-N frequent (or popular) articles are returned to the researcher. Here, the quality of the recommendation will be poorer than the fall-back model but the goal of covering the query will be satisfied.

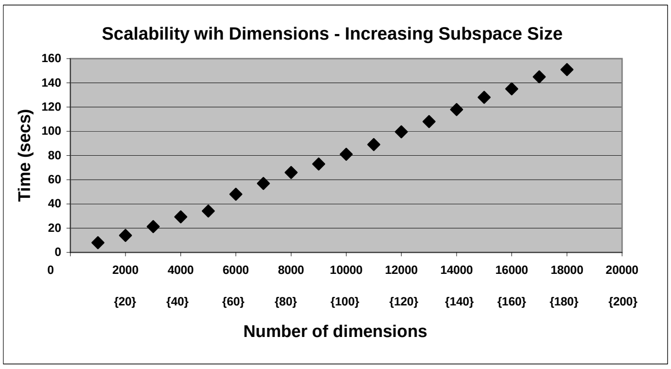


Figure 9: Scalability with dimensionality of the embedded clusters. Size of the subspace (number of dimensions) is increased linearly with the number of dimensions in the data set. Subspace size, indicated in curly braces, is set to be 1% of dimension size.

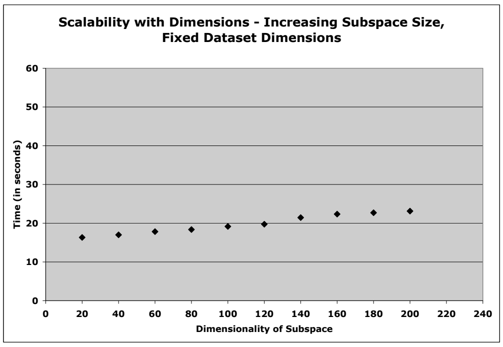


Figure 10: Scalability with dimensionality of the embedded clusters. Size of the subspace(number of dimensions) is increased linearly keeping the number of dimensions in the data set fixed.

4 Experiments

We first evaluate our subspace clustering algorithm using synthetic data and then evaluate the recommendation approach with the benchmark data. With synthetic data, we can embed clusters in specified subspaces. Since we know these locations, we can check whether the clustering algorithm is able to recover the clusters. For evaluating the quality of recommendations, *MovieLens* (Herlocker et al., 2004) benchmark dataset has been used. We compare our approach using SCuBA with the baseline approach defined in Section 4.2.

4.1 Clustering Evaluation on Synthetic Data

We have developed a synthetic data generator that allows us to embed clusters in subspaces. We can control the number and the size of such clusters and also size of the dataset. Apart from the clusters, the data space is filled with noise. For each row we place x noise values at random positions where $x = \alpha \times$ the number of dimensions. We set $\alpha = 1\%$. An example of a synthetic data with three clusters is shown in Figure-7.

For scalability, we measure the running time as we increase the number of dimensions in increments of 1000. The number of rows is kept fixed at 1000. We embed 5 subspace clusters. Each subspace cluster has 20 dimensions and 10 instances. Figure-8 shows the scalability of our algorithm as we increase the dimensions in the data. Notice that the curve is linear since our subspace search strategy is not dependent on the number of dimensions but rather it searches in the row-enumeration space.

In the second scalability graph, shown in Figure-9, size of the subspaces is linearly increased with the number of dimensions in the data. The size of the subspace is set to 1 percent of the number of dimensions. For example, when the number of dimensions is 5000, the size of the subspace is 50. Here the running time is negligibly higher than the previous case but the curve is still linear. The higher running time is due to the computation of intersection, k, between two strings to check for redundancy as mentioned in section 3.3. As discussed previously, the size of k is generally very small due to the high sparsity of the data.

In the third scalability experiment as shown in Figure-10, we check the behavior of SCuBA as the density of the dataset increases. Dimensionality of the dataset is fixed to 2000 and the number of instances to 1000. We embed 5 subspace clusters each with 10 instances. The subspace size is increased from 20 to 200 in steps of 20. It can be observed that running time increases with the density of the dataset but the curve remains linear.

Data Dimensions	1000	2000	3000	4000	5000
Recovery Accuracy	5/5 100 %	5/5 100 %	5/5 100 %	5/5 100 %	5/5 100 %

Table 1: Accuracy in recovering the embedded subspace clusters. Five datasets with increasing number of dimensions. In all cases, the 5 embedded clusters are recovered.

We present accuracy results of our subspace clustering algorithm in Table-1. Here, accuracy is defined by the number of true positives and true negatives w.r.t. the recovered clusters. In all cases, the embedded clusters were completely recovered as shown in Table-1. No extra clusters were reported even though $\alpha = 1\%$ of noise is present in the data.

Finally we present a comparison between SCuBA and an existing subspace clustering algorithm, MAFIA (Goil et al., 1999). In the first experiment, as shown in Figure-11, we compare the running times of both the algorithms with increasing number of dimensions of the dataset from 100 to 600 in steps of 100. Again 5 subspace clusters were embedded with subspace size, 10% of the dimensionality of the dataset. Number of instances in the dataset is fixed to 500 and instances in subspace cluster is fixed to 10. As expected, SCuBA scales linearly with increasing number of dimensions whereas MAFIA shows exponential behavior till 400 number of dimensions, after which it runs out of memory. The second experiment compares the strength of the both the algorithms. MAFIA is known to perform well for dataset with more number of instances as compared to the number of dimensions. SCuBA performs well for more number of dimensions as compared to the number of instances as mentioned in Section 3.1. So we transpose the high-dimensional low-instance data (for SCuBA) to get low-dimensional high-instance data (for MAFIA). Subspaces, thus generated for these

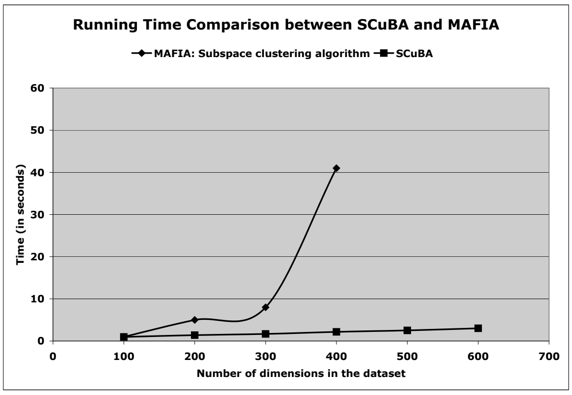


Figure 11: Running time comparison between SCuBA and MAFIA with increasing dataset dimensionality and increasing subspace size.

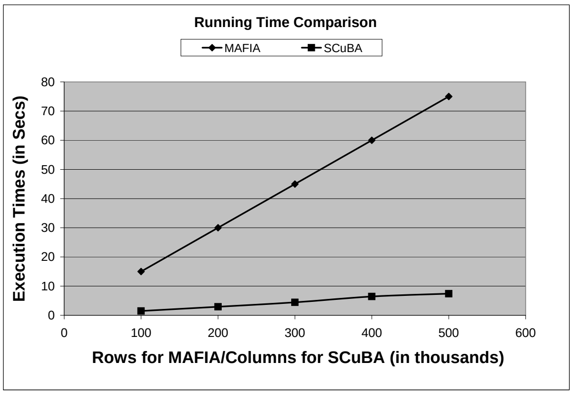


Figure 12: Running time comparison between SCuBA and MAFIA with increasing dataset dimensionality and increasing subspace size.

datasets will be exactly same. We compare the running times for both SCuBA and MAFIA. For MAFIA, the number of dimensions is fixed to 20 but number of instances change from 100K to 500K. Similarly, for SCuBA the number of dimensions are increased from 100K to 500K, keeping the number of instances fixed at 20. Although both the algorithms scale linearly with increasing dimensionality (instances) for SCuBA (MAFIA), the time difference between SCuBA and MAFIA is near ten-folds, as shown in Figure-12.

4.2 Recommendations from Benchmark data

In this part of the experiment we evaluate the quality of the recommendations made by the SCuBA approach on the *MovieLens* dataset.

In Section 1, we reviewed two approaches in CF and pointed out that memory-based approach produce high quality recommendations by finding the nearest neighbors of a target user. We use this as our baseline approach. It is quite practical to assume that users view or rate very few articles of the thousands of available. Since we want to make recommendations based on the few articles a user looks at, we show that when the number of selected terms are few, our approach produces higher quality recommendations than the baseline approach.

Precision and *recall* are widely used measures to evaluate the quality of information retrieval systems. In our experiments, we define quality using precision which is the ratio between number of relevant results returned and the total number of returned results. We choose this measure for the following reasons. The goal of a recommender system is to present a small amount of relevant information from a vast source of information. Therefore, it is more important to return a small number of recommendations that contains relevant items rather than giving the user a large number of recommendations that may contain more relevant

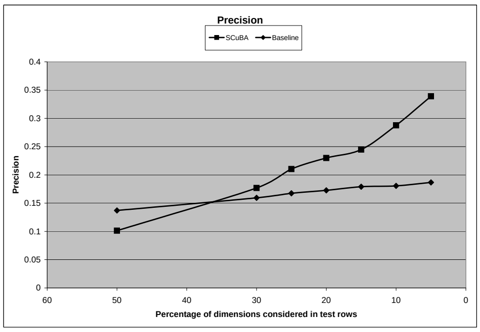


Figure 13: Precision measurements as the percentage of query items is reduced.

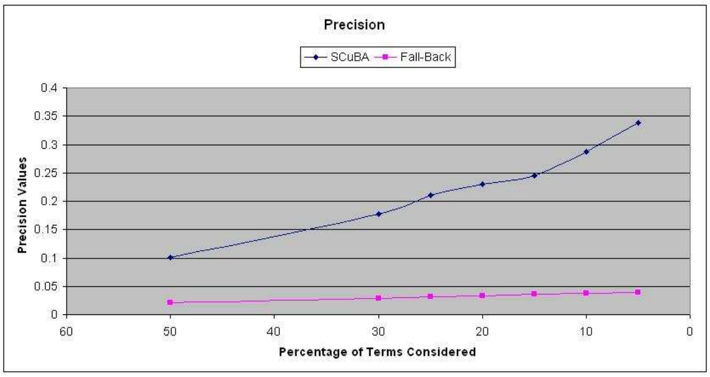


Figure 14: Comparison of Precision values between SCuBA and Fall-Back model.

recommendations but also requires the user to sift through many irrelevant results. The ratio between the number of relevant results returned and the number of true relevant results is defined as recall. Notice it is possible to have very high recall by making a lot of recommendations. In the research paper recommendation domain, a user will be more interested in reading papers that really qualify for his interests rather than going through a huge list of recommended papers and then selecting those which are of interest. Precision more accurately measures our ability to reach our goal than recall.

Experimental Setup: We divide the data into training and testing sets with 80% used for training and 20% for testing. For our subspace clustering approach, we build subspace clustering models from the training data and for the baseline approach we use the training data to find similar users and make recommendations based on those similar users. During the testing phase, for each user from the test set we take a certain percentage of the items from the test row. We call these *query items*. The rest of the items in the test row are called *hidden items*. We make recommendations (using both approaches) for the user, based on the query items. The list of recommended items are compared with the hidden items and the intersection gives the number of relevant recommendations (results) returned. This forms the basis of our precision measure.

Results and Discussion: The precision curve in Figure-13 shows that we perform better than the baseline approach as we reduce the percentage of query items. As the query items decrease, both relevant recommendations and total recommendations also decrease. In case of our approach using SCuBA, the decrease in relevant recommendations is less than the decrease in total recommendations which is not the case with the baseline approach. Therefore an increase in the precision value is observed. The results validates the discussion presented in Section 1 where it was pointed out that although the user comparison approach produces very high quality recommendations, it will not perform well in our domain where we would like to make recommendations based on very few query terms. Moreover, user comparison approach

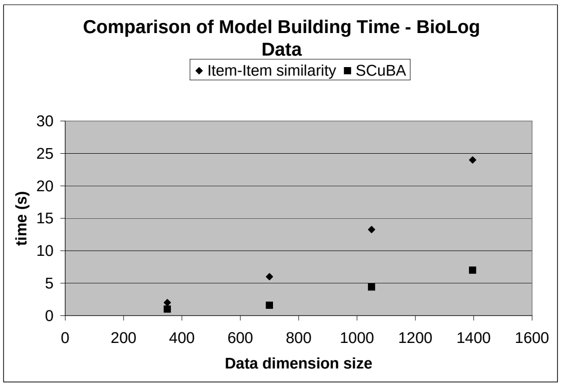


Figure 15: Time comparison of building models with two approaches on Biolog Dataset.

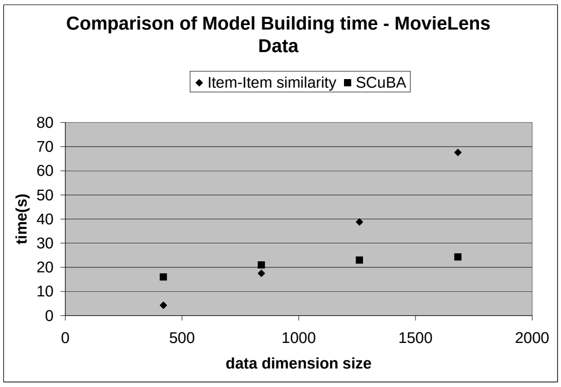


Figure 16: Time comparison of building models with two approaches on MovieLens Dataset.

does not scale linearly with the number of dimensions as shown in Figure-16. These results also verify the fact that more focussed relations are captured using our approach.

The user comparison approach treats users as vectors and computes the similarity between two vectors. The similarity calculations will be poor when there are only a few terms in the test row. In other words, this approach requires large user profiles (similar to e-commerce applications) to generate high quality recommendations which in turn warrants user-tracking and raises privacy issues. In our case we do not require user-profiles that saves the overhead of user-tracking and preserves privacy as well. At a given instant, a researcher may be interested in a new research topic(s), and if we use the researcher's previous profile or likings, we will not find relevant articles matching his/her current interest(s). With SCuBA approach we can overcome this challenge as shown in the precision curve.

4.3 Model Building Times

In model-based approaches a model is created using item similarity (Deshpande and Karypis, 2004). Since, the complexity of building the similarity table is dependent on the number of items, this approach would be unnecessarily computationally expensive in the research paper domain where we have large number of articles but much smaller number of users. Our proposed solution takes advantage of the small number of users and avoids dependence on the number of items. Hence, we would expect that the time required to build models following the subspace clustering approach would be much less than the above approach in (Deshpande and Karypis, 2004).

Our claim is validated by the results shown in Figure-15 and Figure-16. Here, we measure the time taken to build models from the two data sets used in the experiments. The subspace clustering approach clearly outperforms the item-item similarity approach.

4.4 Coverage Results

In this section we present statistics on query coverage of the subspace clustering model. As was discussed earlier, we anticipated that the subspace clustering model will not be able to provide complete coverage of queries and hence we proposed a fall-back model. The results shown in Table-2 validate our hypothesis but more importantly they show that percentage of queries not covered by the subspace clustering model is very low. This means that the fall-back model is rarely used and hence the overall quality of the recommendation will not suffer too much.

The dataset used was the same benchmark dataset, *MovieLens*, used to evaluate the quality of recommendations in Section 4.2. Here the query length denotes the number of terms considered in the test row. The total number of test rows considered is 188 which is 20% of the complete dataset, 80% of which was used to construct the subspace cluster model. For queries of length 1, there were 16 test rows out of 188 for which no recommendation was made, or the miss percentage is 8.5%. For query lengths greater than 4, miss percentage become zero or some recommendation was made.

Query Length	1	2	3	4	5
Miss Percentage %	8.5	0.5	0.5	0.5	0

Table 2: Percentage of queries that were not covered by the subspace cluster model.

4.5 Comparing Subspace Cluster Model with Fall-Back Model

In Section 3.6, fall-back model was introduced which is used when there is no mapping of query terms to the subspace cluster model. In these experiments we try to show that the fall-back model is not a replacement for subspace clustering model. The fall-back model avoids those situations when subspace cluster model fails to make recommendations and it should be used for that purpose only. The experimental setup is same as in Section 4.2. We compare *Precision* again for both the models, SCuBA as well as the fall-back. The precision curve in Figure-14 shows that SCuBA performs far better than the fall-back model for different values of query terms considered. These results also support the discussion in Section 3.6, the goal of the fall-back model, to provide coverage for query items even if the quality of recommendations is compromised.

5 Related Work

Recommender systems can be grouped into two broad categories, content based and collaborative filtering. Content based systems attempt to analyze inherent characteristics of the items to find relationships between them. Collaborative filtering based systems, instead examine the preferences of the systems users to determine relationships between users and items. Research paper recommendation systems are mainly content based systems, utilizing a combination of text based analysis as well as the citations of each paper to generate recommendations. Collaborative filtering algorithms have been very successful in other domains, however, and their application to research paper recommendation has not been fully explored. Most collaborative filtering systems generate models in order to scale up massive numbers of users and items. We developed SCuBA, based on the principles of subspace clustering, to generate the collaborative filtering models to recommend research papers. This section explores related work in both recommender systems and subspace clustering.

5.1 Recommender Systems

Content Based recommender systems attempt to determine relationships between items by comparing inherent characteristics of the items. In the research paper domain, the *CiteSeer* system utilizes the citation network between papers to find related papers (Bollacker et al., 1998; Giles et al., 1998; Bollacker et al., 1999). *CiteSeer* also utilizes text-based analysis, but as a separate list of recommendations. Bradshaw et. al. combined text based analysis by taking into account how the work was cited (Bradshaw et al., 2000). In *ClaiMaker*, Li et. al. propose a system that analyzes the claims made in a paper (Li et al., 2002). *ClaiMaker* utilizes ontologies to improve the accuracy of the analysis. *Quickstep* and *FoxTrot* both utilize ontologies of research topics to assist in recommending research papers (Middleton et al., 2004). McNee et. al. propose a method to combined the citation network with various existing CF algorithms (McNee et al., 2002).

Collaborative filtering approaches can be divided into user-based and model-based approaches. The nearest neighbor, user-based approaches make recommendations by examine the preferences of similar users. Model-based approaches attempt to improve performance by building models of user and item relationships and using those models to make recommendations.

Early CF systems compare the active user to all of the other users and found the k most similar users (Konstan et al., 1997; Sarwar et al., 2000; Shardanand and Maes, 1995). Weights are then assigned to items based on the preferences of the neighborhood of k users, using a cosine or correlation function to determine the similarity of users. Recommendations are made using the weighted list of items. The recommendations produced are high quality and the systems are easily able to incorporate new or updated information, but the approaches do not scale well (Schafer et al., 2001). Mobasher (Mobasher et al., 2000) and Ungar (Ungar and Foster, 1998) addressed the issue of performance by comparing the current user to pre-computed clusters of users. Performance was improved, however, the quality of the recommendations suffered.

To overcome the scalability issues, model based systems were developed. These systems pre-build a user or item based model that is used to make recommendations for an active user. There are two major categories of models, user based and item based. Billsus and Pazzani used a neural network approach to model building, which produce high quality recommendations, but was still rather slow (Billsus and Pazzani, 1998). Aggarwal et. al. introduced a graph-based approach where the nodes were users and the edges their similarity. Bayesian probability networks also proved to be useful in building models (Breese et al., 1998). Performance was further improved by using dependency networks (Heckerman et al., 2001.

Two issues of particular interest to research paper recommender systems, and addressed by SCuBA, are the sparsity of data the communities of researchers. Using a bipartite graph, Huang et. al. were able to find transitive associations and alleviate the sparsity problem found in many recommender system datasets (Huang et al., 2004). Hofmann was able to discover user communities and prototypical interest profiles using latent semantic analysis to create compact and accurate reduced-dimensionality model of a community preference space (Hofmann, 2004).

Sarwar et. al. used correlations between items to build models (Sarwar et al., 2001). Taking advantage of the categorical nature of the data, many item based models are based on association rules (Lin et al., 2000; Mobasher et al., 2000; Demiriz, 2001; Demiriz, 2004). In the most recent, Demiriz borrows from clustering approaches and uses a similarity measure to find rules, instead of an exact match (Demiriz, 2004). Deshpande and Karypis used conditional probability models, used to find higher-order item correlations (Deshpande and Karypis, 2004).

5.2 Subspace Clustering Algorithms

Subspace clustering algorithms can be broadly categorized based on their search method, top-down or bottom-up (Parsons et al., 2004). Top down approaches search in the full dimensional space and refine the search through multiple iterations. Searching in all of the dimensions first means they are not well suited for sparse data such as that found with recommender systems. Bottom-up approaches first search for interesting areas in one dimension and build subspaces. This approach is much more suited to sparse datasets where clusters are likely to be found using fewer than 1% of the dimensions.

The prototypical bottom-up algorithm is based on the downward closure property of density. This states that a area that is dense in n dimensions must also be dense in all $n-1$ dimensional projections. By combining dense areas in lower dimensional spaces, bottom-up algorithms greatly reduce the higher dimensional search space. *CLIQUE* was the first such algorithm and follows the basic approach (Agrawal et al., 1998). Adaptations to the basic method include *ENCLUS* which uses entropy instead of measuring density directly (Cheng et al., 1999) and *MAFIA* which uses a data-driven adaptive method to form bins (Goil et al., 1999). *CLTree* uses a decision tree algorithm to determine the boundaries of the bins (Liu et al., 2000). *Cell Based Clustering* also attempted to optimize the binning procedure (Chang and Jin, 2002). Another algorithm, DOC, compares randomly selected bins to find those that best define clusters (Procopiuc et al., 2002). Each of these algorithms focus on continuous valued data and do not perform well on categorical or binary data. Recently there have been subspace clustering algorithms developed for binary (Patrikainen and Manilla, 2004) and categorical data (Peters and Zaki, 2005). The few algorithms designed for both sparse and binary high-dimensional data do not cluster in subspaces of the dataset (Dhillon and Guan, 2003; Ordonez, 2003).

6 Conclusions and Future Work

In this paper, we proposed a subspace clustering approach for recommender systems aimed at the research paper domain. A useful source of information when recommending research papers is the reading habits of other researchers who are interested in similar concepts. Thus, we adopted a collaborative filtering approach which allows us to use data collected from other researchers browsing patterns, and avoids issues with the interpretation of content. Such data consists of a small number of users (researchers) and a very large number of items (research papers). Our proposed approach takes advantage of the unique characteristics of the data in this domain and provides a solution which is fast, scalable and produces high quality recommendations.

Existing approaches for recommender systems were developed primarily for e-commerce domains and thus have to deal with a large and ever increasing user base. These approaches use models based on either item/item comparisons or based on finding users groups of similar users. Creating an item/item matrix is unnecessarily expensive, since the number of users is so small in comparison to the huge number of research papers, and thus must often be done offline. User/user models address this when the number of users is small. However, existing approaches use a nearest neighbor approach comparing users across the entire dimensionality of the dataset (all items). This requires that the target user be defined by extensive profile information. Our system finds similar users in subspaces of the dataset, each representing a particular area of interest. This allows us to make recommendations based on as few as a single query item. This has the advantage of allowing us to adapt our recommendations to the immediate, short-term interests of the user.

In order to improve the perceived quality and usefulness of the recommendations, a ranking scheme could be developed as an extension to the algorithm. This scheme could rank the recommendations in order of the likelihood they would be useful. The algorithm could also be extended to include the subjective user ratings rather than treating them as binary values and categorize recommendations under strong, mediocre and weak recommendations. A lot of work has been done in mixing the two models, content based filtering and collaborative filtering, to generate a hybrid model which tries to enhance the recommendation quality and also solve the Cold-Start problem for the case where there are no user preferences at all (Schein et al., 2002). This could be one of the future directions that can be pursued from the work in this paper.

References

Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 94–105. ACM Press.

Balabanovi, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72.

Basu, C., Hirsh, H., and Cohen, W. (1998). Recommendation as classification: using social and content-based information in recommendation. In *Proceedings of the fifteenth national/tenth Conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 714–720. American Association for Artificial Intelligence.

Billsus, D. and Pazzani, M. J. (1998). Learning collaborative information filters. In *Proceedings of the*

Fifteenth International Conference on Machine Learning, pages 46–54. Morgan Kaufmann Publishers Inc.

Bollacker, K., Lawrence, S., and Giles, C. L. (1998). CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In Sycara, K. P. and Wooldridge, M., editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York. ACM Press.

Bollacker, K., Lawrence, S., and Giles, C. L. (1999). A system for automatic personalized tracking of scientific literature on the web. In *Digital Libraries 99 - The Fourth ACM Conference on Digital Libraries*, pages 105–113, New York. ACM Press.

Bradshaw, S., Scheinkman, A., and Hammond, K. J. (2000). Guiding people to information: providing an interface to a digital library using reference as a basis for indexing. In *Intelligent User Interfaces*, pages 37–43.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence.*

Chang, J.-W. and Jin, D.-S. (2002). A new cell-based clustering method for large, high-dimensional data in data mining applications. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 503–507. ACM Press.

Cheng, C.-H., Fu, A. W., and Zhang, Y. (1999). Entropy-based subspace clustering for mining numerical data. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 84–93. ACM Press.

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

Demiriz, A. (2004). Enhancing product recommender systems on sparse binary data. *Data Min. Knowl. Discov.*, 9(2):147–170.

Deshpande, M. and Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177.

Dhillon, I. S. and Guan, Y. (2003). Information theoretic clustering of sparse co-occurrence data. In *Proceedings of the third International Conference on Data mining*. IEEE Press.

Giles, C. L., Bollacker, K. D., and Lawrence, S. (1998). CiteSeer: an automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98. ACM Press.

Goil, S., Nagesh, H., and Choudhary, A. (1999). Mafia: Efficient and scalable subspace clustering for very large data sets. Technical Report CPDC-TR-9906-010, Northwestern University, 2145 Sheridan Road, Evanston IL 60208.

Heckerman, D., Chickering, D. M., Meek, C., Rounthwaite, R., and Kadie, C. (2001). Dependency networks for inference, collaborative filtering, and data visualization. *J. Mach. Learn. Res.*, 1:49–75.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53.

Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 194–201. ACM Press/Addison-Wesley Publishing Co.

Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115.

Huang, Z., Chen, H., and Zeng, D. (2004). Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):116–142.

Kamba, T., Bharat, K., and Albers, M. C. (1995). The Krakatoa Chronicle: An interactive personalized newspaper on the Web. In *Fourth International World Wide Web Conference Proceedings*, pages 159–170.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87.

Li, G., Uren, V., Motta, E., Buckingham-Shum, S., and Domingue, J. (2002). Claimaker: Weaving a semantic web of research papers. In *Proceedings of the 1st International Semantic Web Conference, Sardinia, June 2002.*

Lin, C., Alvarez, S., and Ruiz, C. (2000). Collaborative recommendation via adaptive association rule mining. In *In Proceedings of the International Workshop on Web Mining for E-Commerce (WEBKDD 2000).*

Liu, B., Xia, Y., and Yu, P. S. (2000). Clustering through decision tree construction. In *Proceedings of the ninth International Conference on Information and Knowledge Management*, pages 20–29. ACM Press.

McNee, S. M., Albert, I., Cosley, D., Gopalkrishnan, P., Lam, S. K., Rashid, A. M., Konstan, J. A., and Riedl, J. (2002). On the recommending of citations for research papers. In *Proceedings of the 2002 ACM Conference on Computer supported cooperative work*, pages 116–125. ACM Press.

Middleton, S. E., Shadbolt, N. R., and Roure, D. C. D. (2004). Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 22(1):54–88.

Mobasher, B., Cooley, R., and Srivastava, J. (2000). Automatic personalization based on web usage mining. *Commun. ACM*, 43(8):142–151.

Ordonez, C. (2003). Clustering binary data streams with k-means. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 12–19. ACM Press.

Parsons, L., Haque, E., and Liu, H. (2004). Subspace clustering for high dimensional data: a review. *SIGKDD Explorations.*, 6(1):90–105.

Patrikainen, A. and Manilla, H. (2004). Subspace clustering of high-dimensional binary data - a probabilistic approach. In *Workshop on Clustering High Dimensional Data and its Applications, SIAM International Conference on Data Mining.*

Peters, M. and Zaki, M. J. (2005). Clicks: Clustering categorical data using k-partite maximal cliques. In *IEEE International Conference on Data Engineering.* IEEE.

Procopiuc, C. M., Jones, M., Agarwal, P. K., and Murali, T. M. (2002). A monte carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 418–427. ACM Press.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer supported cooperative work*, pages 175–186. ACM Press.

Resnick, P. and Varian, H. R. (1997). Recommender systems. *Commun. ACM*, 40(3):56–58.

Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the tenth International Conference on World Wide Web*, pages 285–295. ACM Press.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000). Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic commerce*, pages 158–167. ACM Press.

Schafer, J., Konstan, J., and Reidl, J. (1999). Recommender systems in e-commerce. In *Proceedings of ACM E-Commerce*.

Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153.

Schein, A., Popescul, A., Ungar, L., and Penncok, D. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002).*

Shardanand, U. and Maes, P. (1995). Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co.

Terveen, L., Hill, W., Amento, B., McDonald, D., and Creter, J. (1997). Phoaks: a system for sharing recommendations. *Commun. ACM*, 40(3):59–62.

Torres, R. D. (2004). Combining collaborative and content-based filtering to recommend research papers. Master's thesis, Universidade Gederal Do Rio Grande Do Sul.

Ungar, L. H. and Foster, D. P. (1998). Clustering methods for collaborative filtering. In *In Workshop on Recommendation Systems at the 15th National Conference on Artificial Intelligence.*

Woodruff, A., Gossweiler, R., Pitkow, J. E., Chi, E. H., and Card, S. K. (2000). Enhancing a digital book with a reading recommender. In *ACM Conference on Human Factors in Computing Systems*, pages 153–160.