Research Paper Recommender Systems: A Subspace Clustering Approach[⋆]

Nitin Agarwal, Ehtesham Haque, Huan Liu, and Lance Parsons

Arizona State University, Tempe AZ 85281, USA

Abstract. Researchers from the same lab often spend a considerable amount of time searching for published articles relevant to their current project. Despite having similar interests, they conduct independent, time consuming searches. While they may share the results afterwards, they are unable to leverage previous search results during the search process. We propose a research paper recommender system that avoids such time consuming searches by augmenting existing search engines with recommendations based on previous searches performed by others in the lab. Most existing recommender systems were developed for commercial domains with millions of users. The research paper domain has relatively few users compared to the large number of online research papers. The two major challenges with this type of data are the large number of dimensions and the sparseness of the data. The novel contribution of the paper is a scalable subspace clustering algorithm (SCuBA[1]) that tackles these problems. Both synthetic and benchmark datasets are used to evaluate the clustering algorithm and to demonstrate that it performs better than the traditional collaborative filtering approaches when recommending research papers.

1 Introduction

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 [20].

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 automatically recommending papers based on the preferences of other researchers with similar interests.

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

1 SCuBA: Subspace Clustering Based Analysis.

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.

The remainder of the paper is organized as follows. In Section 2 we discuss the common challenges in the collaborative filtering process followed by 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 Challenges, Definitions and Problem Statement

In spite of the success achieved by CF algorithms, there are limitations to such systems [10] that arise in the research paper domain. In many domains, there is an ever increasing number of users while number of items remains relatively stable. However, in research paper recommendation domain, 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 of 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 [16].

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 the accuracy of nearest neighbor algorithms is very poor for sparse data [4]. 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-field regardless of differences in other fields. Searching for similar users across all of the items leads to finding users who share many common interests. 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.

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

The access patterns of the researchers can be maintained by tracking the log of research papers they access. From these access patterns, we can generate a researcher/article data space as defined in the previous section. 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.

Our proposed subspace clustering algorithm is a two-step process staring with finding subspaces that form clusters, 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, the original vector is transformed 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 [17]. 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 [17], we chose an efficient representative subspace clustering algorithm, MAFIA [7], 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.

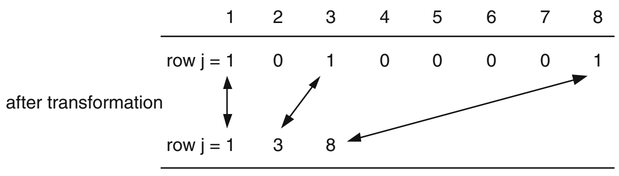

Fig. 1. Transformation: A row in the original matrix is transformed into a string containing the position of the non-zero columns

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

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.

	1	1	2	3	17	
row id	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

Fig. 2. Transformed Data D after being compressed

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. The hash table memory requirement grows slowly since the 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. 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).*

```
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
```

Fig. 3. Subspace Clustering Algorithm

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:

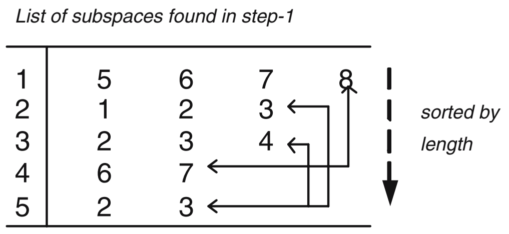

Fig. 4. Set of Subspaces S with redundancy. Subspaces marked with arrows are pairs where one subspace is subsumed by another. For example, subspace 4 is subsumed by subspace 1.

1. Sort the set S according to the size of each subspace in descending order. The result of sorting is 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 these 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 a number of 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 indicating the percentage of dimensions that match. If the overlap is above 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

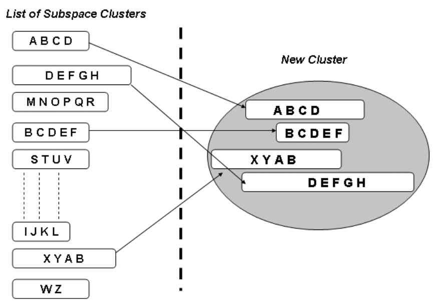

Fig. 5. The subspaces are grouped together in one cluster if there is overlap between subspaces

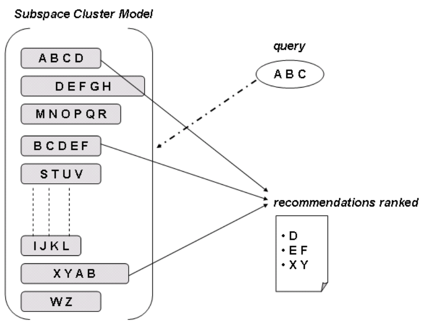

Fig. 6. Using the subspace clustering model to generate recommendations

with *A B C D*. In this example, subspaces with at least one item in 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. The user can control the threshold parameter depending on the domain and the data characteristics. 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 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 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.

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* [10] benchmark dataset has been used. We compare our approach using SCuBA with the baseline approach defined in Section 4.2. We present the experimental details in the following subsections.

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

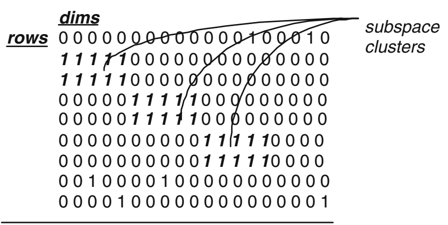

Fig. 7. Example of synthetic data used to evaluate the subspace clustering algorithm. In this example, 3 subspace clusters are embedded and noise is placed at random positions.

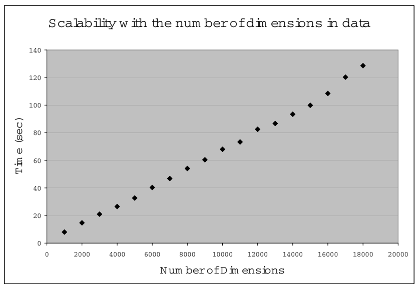

Fig. 8. Scalability with the number of dimensions in the data set

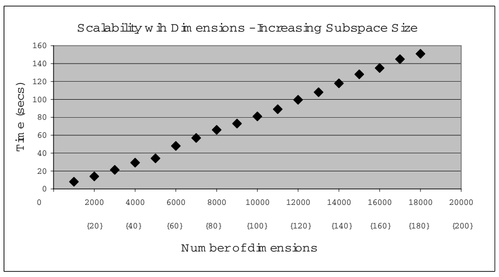

Fig. 9. Scalability with dimensionality of the embedded clusters. Size of the subspace (number of dimensions) is increased linearly with the number of dimensions in the data set. Subspace size, indicated in curly braces, is set to be 1% of dimension size.

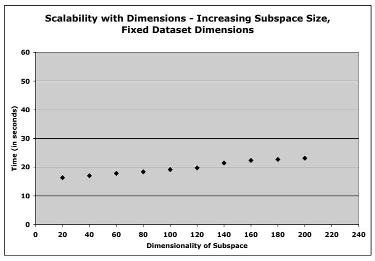

Fig. 10. Scalability with dimensionality of the embedded clusters. Size of the subspace(number of dimensions) is increased linearly keeping the number of dimensions in the data set fixed.

strategy is not dependent on the number of dimensions but rather it searches in the row-enumeration space.

In second scalability graph, shown in Figure-9, we linearly increase the size of subspaces with the number of dimensions in the data. In this case, we set the size of the subspace to be 1 percent of the number of dimensions. For example, when the number of dimensions is 5000, size of the subspace is 50. Here the running time is negligibly higher than the previous case but the curve is still linear. 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 high sparsity of 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

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.

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

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.

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.

4.2 Recommendations from Benchmark Data

Here 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 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-11 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 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 does not scale linearly with the number of dimensions as shown in Figure-14. 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

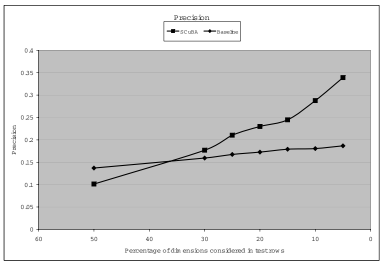

Fig. 11. Precision measurements as the percentage of query items is reduced

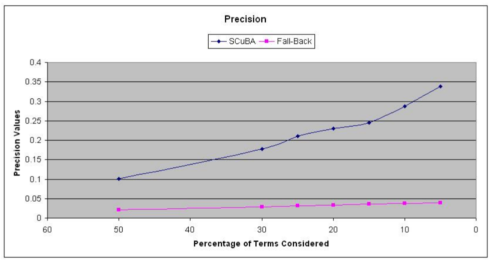

Fig. 12. Comparison of Precision values between SCuBA and Fall-Back model

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 [5]. 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 [5].

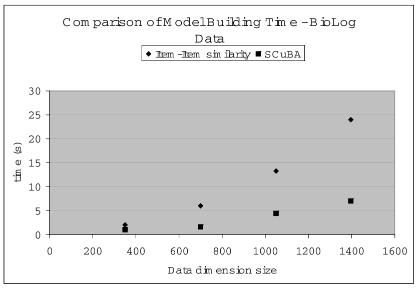

Fig. 13. Time comparison of building models with two approaches on Biolog Dataset.

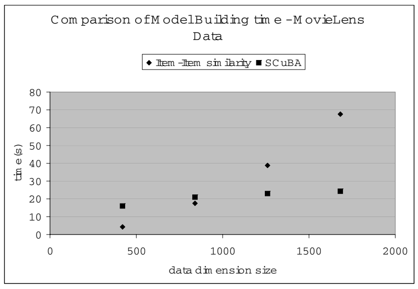

Fig. 14. Time comparison of building models with two approaches on MovieLens Dataset.

Our claim is validated by the results shown in Figure-13 and Figure-14. 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

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

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

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

Experiment was performed with the *MovieLens* dataset. 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.

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 helps to avoid those situations when subspace cluster model fails to make recommendations and it should be used for that purpose only. The experimental setup is kept the same as in Section 4.2. Again, we compare *Precision* for both the models, SCuBA as well as the fall-back. The precision curve in Figure-12 clearly shows 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 is to provide coverage for query items even if the quality of recommendations is compromised.

5 Related Work

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 items. In research paper domain, the *CiteSeer* system utilizes the citation network between papers to find related papers [8]. *CiteSeer* also utilizes text-based analysis, but as a separate list of recommendations. *Quickstep* and *FoxTrot* both utilize ontologies of research topics to assist in recommendation [15]. McNee et. al. propose a method to combined the citation network with various existing CF algorithms [14].

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 [22]. 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.

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. 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 [2]. Performance was further improved by using dependency networks [9]. Using a bipartite graph, Huang et. al. were able to find transitive associations and alleviate the sparsity problem found in many recommender system datasets [12]. 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 [11]. Sarwar et. al. used correlations between items to build models [21]. Recently, Demiriz borrows from clustering approaches and uses a similarity measure to find rules, instead of an exact match [4].

5.2 Subspace Clustering Algorithms

Subspace clustering algorithms can be broadly categorized based on their search method, top-down or bottom-up [17]. 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.

CLIQUE was the first bottom-up algorithm and follows the basic approach [1]. Adaptations to the basic method include *ENCLUS* which uses entropy instead of measuring density directly [3] and *MAFIA* which uses a data-driven adaptive method to form bins [7]. *CLTree* uses a decision tree algorithm to determine the boundaries of the bins [13]. 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 [18] and categorical data [19]. The few algorithms designed for both sparse and binary high-dimensional data do not cluster in subspaces of the dataset [6].

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.

In order to improve the perceived quality and usefulness of the recommendations, a more sophisticated ranking scheme could be developed as an extension to the algorithm. The algorithm could also be extended to include the subjective user ratings rather than treating them as binary values and categorize recommendations as strong, mediocre and weak. 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.

References

1. Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 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, 1998.
2. John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence.*, 1998.
3. Chun-Hung Cheng, Ada Waichee Fu, and Yi Zhang. 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, 1999.
4. Ayhan Demiriz. Enhancing product recommender systems on sparse binary data. *Data Min. Knowl. Discov.*, 9(2):147–170, 2004.
5. Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.
6. Inderjit S. Dhillon and Yuqiang Guan. Information theoretic clustering of sparse co-occurrence data. In *Proceedings of the third International Conference on Data mining*. IEEE Press, 2003.
7. Sanjay Goil, Harsha Nagesh, and Alok Choudhary. 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, June 1999.
8. Abby Goodrum, Katherine W. McCain, Steve Lawrence, and C. Lee Giles. Scholarly publishing in the internet age: a citation analysis of computer science literature. *Information Processing and Management*, 37(5):661–675, 2001.
9. David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *J. Mach. Learn. Res.*, 1:49–75, 2001.
10. Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

11. Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, 2004.
12. Zan Huang, Hsinchun Chen, and Daniel Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):116–142, 2004.
13. Bing Liu, Yiyuan Xia, and Philip S. Yu. Clustering through decision tree construction. In *Proceedings of the ninth International Conference on Information and Knowledge Management*, pages 20–29. ACM Press, 2000.
14. Sean M. McNee, Istvan Albert, Dan Cosley, Prateep Gopalkrishnan, Shyong K. Lam, Al Mamunur Rashid, Joseph A. Konstan, and John Riedl. 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, 2002.
15. Stuart E. Middleton, Nigel R. Shadbolt, and David C. De Roure. Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 22(1):54–88, 2004.
16. Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. Automatic personalization based on web usage mining. *Commun. ACM*, 43(8):142–151, 2000.
17. Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations.*, 6(1):90–105, 2004.
18. Anne Patrikainen and Heikki Manilla. 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.*, 2004.
19. Markus Peters and Mohammed J. Zaki. Clicks: Clustering categorical data using k-partite maximal cliques. In *IEEE International Conference on Data Engineering.* IEEE, 2005.
20. Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, 1997.
21. Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the tenth International Conference on World Wide Web*, pages 285–295. ACM Press, 2001.
22. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic commerce*, pages 158–167. ACM Press, 2000.