An Improved Recommender System Based on Clustering using Representatives

Joydeep Das[1], Harsh Gupta[2], Shreya Dugar[2], Subhashis Majumder[3] and Prosenjit Gupta[4]

[1]The Heritage Academy, Kolkata, WB, India
Joydeep.das@heritageit.edu
[2]Department of Computer Science and Engineering, Institute of Engineering and Management, Kolkata, WB, India
harshguptahs@gmail.com, shreyadugar789@gmail.com
[3]Department of Computer Science and Engineering, Heritage Institute of Technology, Kolkata, WB, India
subhashis.majumder@heritageit.edu
[4]NIIT University, Neemrana, Rajasthan, India
prosenjit_gupta@acm.org

ABSTRACT

Recommender systems have proven to be valuable means for online users to cope up with the information overload and have become one of the most powerful and popular tools in electronic commerce. Collaborative Filtering (CF) is one of the most successful recommendation techniques that recommends by using the opinions of a community of users. However, the similarity computations associated with CF algorithms are very expensive and grow polynomially with the number of users and items in a database. To address this *scalability* problem, we propose a clustering based recommendation approach. Our proposed work partitions the users of the CF system using a CURE (Clustering using representatives) based data clustering algorithm and use the clusters to select the similar users of a target user. In this work, we further try to find the optimal number of clusters by using a binary search based technique. The cluster-based approach reduces the runtime of the system as we avoid similarity computations over the entire database. Experiments performed on MovieLens-1M dataset indicate that our method is efficient in reducing the runtime as well as maintains an acceptable recommendation quality.

Keywords: Recommender Systems, Collaborative Filtering, Data Clustering, Scalability.

I. INTRODUCTION

Recommender Systems (RS) are widely used for automatic personalization of information on web sites and information retrieval systems. RS help users in their decision making process by suggesting items that the user may prefer. Today the problem is not about how to get correct information to make a decision, rather, how to make a right decision out of enormous information. RS use the opinions of a group of uses to help individuals in that group more effectively identify content of interest from a potentially large search space.

Collaborative Filtering (CF) [6] is a recommendation technique that identifies similarities between users, based on their ratings in order to select neighbors and compute predictions for the active users. Unlike content-based approaches [1], which use the content of items previously rated by a user *u*, CF approaches rely on the ratings of *u* as well as those of other similar users in the system. The key idea is that the rating of *u* for a new item *i* is likely to be similar to that of another user *v*, if *u* and *v* have rated other items in a similar way.

Despite the success of RS and CF in many application areas, there are still two potential problems with RS. One is the *scalability*, which is how quickly a RS can generate recommendation, and the second is to ameliorate the quality of the recommendation for a customer. Pure CF recommender systems produce high quality recommendation than those of pure content-based and demographic recommender systems [11]. However, due to the *sparsity*, they cannot find similar items or users using rating correlation, resulting in poor prediction quality and reduced *coverage* (The percentage of the items that can be recommended from all available items in the system).

In this paper, we present a novel framework for CF which combines the strengths of memory-based and model-based CF approaches [1] in order to enable recommendation by groups of closely related individuals. We propose a clustering based CF algorithm that is perfectly applicable for large datasets which are nowadays common in e-commerce applications. Our method is to partition the users of the CF system using a clustering algorithm and use the clusters (partitions) as neighborhood. In this work, we cluster the users of the system using a CURE [5] based clustering algorithm. The use of clusters permits the integration of the advantages from both the memory-based and model-based approaches. By using the rating information from a group of closely related users, unrated items of the individual user in a group can be predicted; this allows the missing values to be filled in. Moreover, the proposed approach is likely to make the recommendations faster, because the size of the group that must be analyzed to find neighbors is much smaller. We also propose an adaptive technique to find the optimal number of clusters. Once clustering is done, we apply any traditional CF algorithm to the clusters individually. Our primary objective is to reduce the overall running time without sacrificing the recommendation quality much. This ensures scalability, allowing us to tackle bigger datasets using limited computational resources. The results of the experiments performed indicate that our approach is effective in reducing the runtime of the algorithm while maintaining an acceptable recommendation quality.

The rest of the paper is organized as follows. In section II, we provide background information about CF and also discuss some of the past works related to clustering CF models. Section III outlines our proposed clustering based approach while section IV describes our experimental framework, experimental results, and evaluations. We conclude discussing our future research directions in Section V.

II. BACKGROUND AND RELATED WORK

A. Collaborative Filtering

Breese et al. [1] divide collaborative filtering approaches into two classes: memory-based and model-based algorithms.

Memory based algorithms maintain the original setup of the CF task. They use statistical techniques to build the neighborhood relationship for an active user, and then usually use a weighted sum of the ratings to predict missing values. A general user-based formulation of the weighted sum scheme can be [1]:

$$p_{a,j} = \overline{r_a} + k \sum_{i=1}^{n} w(a,i)\,(r_{i,j} - \overline{r_i})$$

where *n* is the size of the neighborhood, and $\overline{r_a}$ and $\overline{r_i}$ are the average ratings for the active user *a* and neighbor user *i* respectively. *w*(*a, i*) is the correlation between user *a* and user *i*. *k* is a normalizing factor. The most important part for memory-based algorithms is the similarity measurement. Two commonly used memory-based algorithms are the Pearson Correlation Coefficient (PCC) algorithm [9] and the Vector Space Similarity (VSS) algorithm [1].

Model based algorithms utilize the collection of training data to learn a model first and then use it to make predictions instead of directly manipulating the original database. The modeling process is always performed by machine learning or data mining techniques such as the Bayesian model [7], Regression-based model [10], and Clustering model [8].

B. Clustering CF models

The most related model to this paper is the clustering CF model. Clustering algorithms work by identifying groups of users who share similar preferences. Once the clusters are created, predictions for an individual can be made by averaging the opinions of the other users in that cluster. Clustering techniques usually produce less personal recommendations compared to nearest neighbor algorithms. However, we can achieve better performance (reduced runtime), since the size of the cluster that must be analyzed is much smaller compared to the entire users' space. Several CF based recommendation algorithms incorporated clustering methods in order to alleviate the sparsity and scalability problems. To overcome the sparsity problem, Xue et al. [12] proposed a CF system based on K-means clustering in order to smooth the unrated data for individual users according to the clusters. To address the scalability issue, Sarwar et al. [8] used clustering techniques to partition the data into clusters and used a memory-based CF algorithm such as a Pearson correlation-based algorithm to make predictions for users within each cluster. With the same perspective, Das et al. proposed a decomposition based recommendation algorithm using multiplicatively weighted Voronoi diagram to enhance the scalability of the system [3]. Similar decomposition based recommendation techniques can also be found in [2] and [4].

III. OUR FRAMEWORK

Our primary goal is to partition the large user-item matrix into smaller manageable clusters and apply the CF based recommendation separately to the clusters. The proposed approach should improve the scalability of the system as we avoid similarity computations over the entire user-item matrix. However, there are two main questions: (a) How to find meaningful user-item subgroups or clusters from limited information? The only information we have is the user-item matrix, such as ratings for movies. (b) How to combine user-item subgroups or clusters with existing CF methods and improve their performance?

We find the user-item subgroups (clusters) using a CURE [5] based data clustering algorithm and propose a unified strategy to combine subgroups with existing CF methods. Now, we formally present our algorithm.

Algorithm: Cluster-based Recommendation

Step 1: Preprocess: create user clusters C.
Step 2: Given an active user u_a and *i* rated items, an item *t* and an integer *K*, the number of nearest neighbors:

Step 2.1: Allocate user u_a to one of the clusters $c_a \in C$.

Step 2.2: Calculate similarity $sim(u_a, u)$ for each $u \in c_a$.

Step 2.3: Select the top-*K* most similar users as nearest neighbors of u_a.

Step 2.4: Predict the ratings of a particular item *t* for u_a by the behaviors of the *K* nearest neighbors.

Step 3: Recommend top-*N* items to the active user based on the prediction score.

In this work, we use Pearson's correlation coefficient to compute similarity between two users. Given two users' *u* and *v*, Pearson's coefficient is defined as follows.

$$Sim(u,v) = \frac{\sum_{i \in I}(r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sqrt{\sum_{i \in I}(r_{u,i} - \overline{r_u})^2}\sqrt{\sum_{i \in I}(r_{v,i} - \overline{r_v})^2}}$$

where the $i \in I$ summations are over the items that both the users *u* and *v* have rated and $\overline{r_u}$ is the average rating of the co-rated items of the *u* th user.

The prediction score $P_{a,i}$ for a user *a*, on a certain item *i*, is calculated using the following formula.

$$P_{a,i} = \overline{r_a} + \frac{\sum_{u \in U}(r_{u,i} - \overline{r_u}) * sim(a,u)}{\sum_{u \in U}|sim(a,u)|}$$

where $\overline{r_a}$ and $\overline{r_u}$ are the average ratings for the user *a* and user *u* on all other rated items, and *sim*(*a, u*) is the correlation between user *a* and user *u*. The summations are over all the users $u \in U$ who have rated the item *i*.

A. *Clustering Algorithm*

In this work, we use a CURE [5] based clustering algorithm to cluster the users of the system. CURE is a hierarchical clustering algorithm which can identify clusters having non-spherical shapes and wide variances in size. To handle large databases, CURE employs a combination of random sampling and partitioning. A random sample drawn from the data set is first partitioned and each partition is partially clustered. The partial clusters are then clustered in a second pass to yield the desired clusters. The number *K* is input to the algorithm that specifies the desired number of clusters. CURE method first select *K* users arbitrarily as the initial centers of the *K* clusters respectively. For each of the remaining users, the distance between the user and all the cluster centers is calculated and the user is assigned to the minimum distance cluster. Next, iteratively the cluster centers are re-calculated until the centers do not change their locations any more.

In this work, we use MovieLens-1M dataset (http://www.movielens.org/) for testing our algorithm. The dataset contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users and 18 movie genres. Ratings are on a five star (integral) scale from 1 to 5. We first try to learn the preferences of the individual users on the basis of the genres of movies that they have reviewed. Assuming the fact that movie genre is sufficient to capture how users rate movies, we project the user preferences in a new, low-dimensional space, called User Genre Space (UGS). In UGS, we represent the preferences of each user as a vector, called *user vector*, in *n* dimensions, where *n* is the total number of genres present in the dataset. Each dimension in this vector is represented by a genre and the values of these dimensions are the weights assigned to the corresponding genre. For every movie that a user has rated, the corresponding genre is weighted by its rating. In case of multiple genres, all of them are assigned the same weightage as the rating, and are thus treated equally. Finally the UGS vector is normalized.

We have given an example of movie preferences for two users - User1 and User2 in Table I and Table II, while the corresponding *user vectors* are shown in Figure 1 and Figure 2 respectively. Higher weightage for genres like Comedy, Thriller and Action for User1 displays the liking for movies with these particular themes while User2 has preference for Action and Adventure movies. We calculate the weights of the different genre from Table I and Table II as follows. For the Action genre, User 1 has rated two movies 'Deep Blue Sea' and 'The 13th Warrior'. Since the ratings of the two movies are 5 and 2 respectively, therefore the Action genre is weighted by 5 + 2 = 7. Similarly for the Comedy genre, User1 has rated two movies with a rating of 4, and therefore the weight of the Comedy genre is 4 + 4 = 8. Note that, apart from the Action, the movie 'Deep Blue Sea' also belongs to Sci-Fi and Thriller genres. Therefore both the Sci-Fi and Thriller genres will also be weighted by 5. In this way, we calculate the weights of all the genres present in the *user vector*, and then the vector is normalized. If a user does not have any preference for a particular genre then the weight of that genre is considered as 0.0. Continuing the above mentioned approach, we create *user vectors* for all the remaining users. We implement our clustering algorithm using these *user vectors*. We use Euclidean distance to find the distance between two vectors. Once clustering is done, next we measure the presence of *attribute autocorrelation* in the clusters to verify whether users with similar demographic profiles such as age, gender, occupation, location, etc. are grouped in the same clusters.

TABLE I
AN EXAMPLE OF MOVIE PREFERENCES FOR USER1

Movie	Genre	Rating
Deep Blue Sea	Action, Sci-Fi, Thriller	5
Runaway Bride	Comedy, Romance	4
A Christmas Story	Comedy, Drama	4
The 13th Warrior	Action, Horror, Thriller	2

TABLE II
AN EXAMPLE OF MOVIE PREFERENCES FOR USER 2

Movie	Genre	Rating
Saturn 3	Adventure, Sci-Fi, Thriller	5
Excalibur	Action, Drama, Fantasy, Romance	5
Total Recall	Action, Adventure, Sci-Fi, Thriller	4
Tarzan the Fearless	Action, Adventure	3

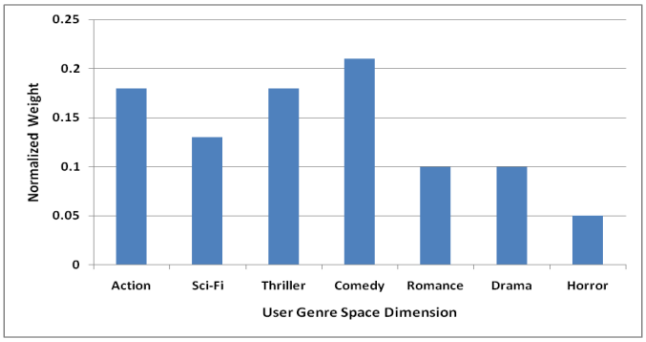

Fig. 1. A Sample User Vector for user1.

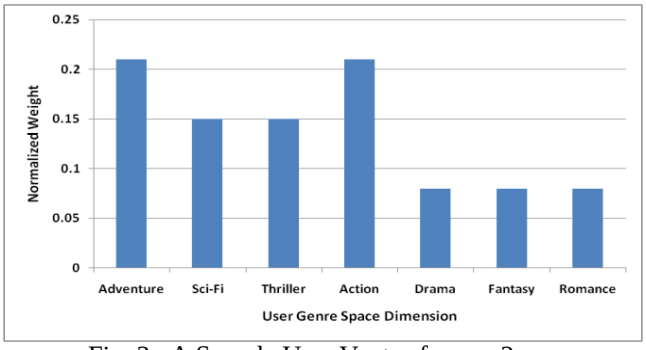

Fig. 2. A Sample User Vector for user2.

B. Attribute Autocorrelation

It measures the co-variance of an attribute of interest with other attributes. For e.g. it can measure the variation of user preferences (ratings) with the different demographic features of the users, such as age, gender, occupation, location, etc. In our work, we use MovieLens 1M dataset and it has information about the demographic attributes of the users, such as age, gender, location and occupation. We can define an autocorrelation index using the age of the users. In the MovieLens 1M data, the age of the users are divided into the following seven age groups: below 18, 18 - 24, 25 - 34, 35 - 44, 45 - 49, 50 - 55 and 56 and above. For simplicity we represent these age groups as discrete values. Age below 18 is mapped to 1, 18 - 24 to 2, 25 - 34 to 3, 35 - 44 to 4, 45 - 49 to 5, 50 - 55 to 6 and 56 and above to 7. Next, we consider one movie at a time and find how the ratings of that movie vary with respect to the different age groups of the users. This can be done as follows.

First find the average rating of the movie. Let us denote this as $\bar{r}$. Now we consider two users *i* and *j* and calculate

$$c_{i,j} = (r_i - \bar{r})(r_j - \bar{r})$$

$c_{i,j}$ is the rating similarity between user *i* and *j*. r_i and r_j are the ratings of user *i* and *j* on that movie.

Let a_i and a_j denote the age groups of user *i* and user *j*. We can define age similarity $w_{i,j}$ between the two users as follows. $w_{i,j} = 1$ if both *i* and *j* belongs to the same age group, and

$$w_{i,j} = \frac{1}{|a_i - a_j| + 1}$$

if they belongs to different age groups.

Now the autocorrelation index *c* can be defined as

$$c = \frac{\sum_i \sum_j w_{i,j}\, c_{i,j}}{\sigma^2 \sum_i \sum_j w_{i,j}}$$

Where

$$\sigma^2 = \frac{\sum_i (r_i - \bar{r})^2}{(\mathrm{n} - 1)}$$

Here σ^2 is the variance. The positive *c* value indicates the presence of attribute autocorrelation, i.e., people with similar age rated similarly while negative value implies that the ratings are dissimilar. A zero value indicates that the ratings are distributed independently of age. We measure the value of this autocorrelation index in all the clusters to verify whether people with similar age are grouped in the same clusters.

C. Finding optimal value of K (no. of clusters): An adaptive approach

The value of the *attribute autocorrelation* defined in the previous section plays a major role in determining the suitable value of K. Since positive values of *c* (autocorrelation index) indicate the presence of attribute autocorrelation in the clusters, therefore we define a metric *PA*, which measures the percentage of positive autocorrelation in the clusters. Next, an adaptive procedure is followed to determine the suitable value of K. Initially, we start with K=2 and find the average *PA* value across the two clusters. Then in successive iteration we increase the value of K exponentially as successive powers of 2 till for some $K = K_{max}$, the average *PA* value across all the K clusters is less than the average *PA* value of the previous K clusters. A binary search is then carried out within the range $K_{max}/2$ and K_{max}, to find K_{opt}, the optimal value of K for which the average *PA* value of the K clusters is greater or equal to the average *PA* value of the preceding K clusters. Note that the above adaptive method ensures that if K_{opt} is the desired value of K, then it is found in at most $2\log 2K_{opt} - 1 = O(\log 2K_{opt})$ iterations

D. Our Recommendation Approach

The main idea of our work is to apply some CF algorithm in each cluster and try to merge the prediction results together. For a *target user*, we extract the genre preferences of the user and represent his/her preferences as a vector (*user vector*) in *n* dimensions, where *n* is the total number of genres. The weights of the genres are assigned using the method discussed in section III (A). Now we need to assign the target user to one of the clusters. The distance between the *target user* and cluster mean of all the clusters is calculated and the user is the placed in the minimum distance cluster. Then we can apply any CF algorithm to the cluster to recommend items of interest to the *target user*.

Algorithm: Recommend_Movies

Step 1: Select a *target user* for recommendation.
Step 2: Represent the genre preferences of the *target user* as a vector.
Step 3: Assign weights to different genres of the vector.
Step 4: Calculate the distance between the *target user* and mean of all the clusters.
Step 5: Assign the *target user* to the minimum distance cluster.
Step 5: Recommend *top-N* movies using any CF method.

IV. EXPERIMENTAL SETTINGS

We have tested our recommendation algorithm on the MovieLens 1M dataset to validate our scheme. The user ratings of the dataset are randomly split into two sets - training set (80%) and test set (20%) for testing. Ratings for the items in the test set were to be predicted.

A. Evaluation Metric Discussion

We use Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) [9] to evaluate the prediction accuracy of our recommendation algorithm. The objective of any recommendation algorithm is to reduce MAE and RMSE values. Since our system produces a list of Top-N recommended items, therefore MAE and RMSE are not sufficient to truly evaluate the recommendation quality. Therefore, we also use *Precision* and *Recall* metrics [6] to evaluate the quality of the ranked list. *Precision* (P) is defined as the ratio of relevant items selected to number of items selected. *Precision* represents the probability that a selected item is relevant. *Recall* (R) is defined as the ratio of relevant items selected to total number of relevant

items available. *Recall* represents the probability that a relevant item will be selected.

$$P = \frac{tp}{tp + fp} \qquad and \qquad R = \frac{tp}{tp + fn}$$

True Positive (*tp*) or a hit denotes the case where a product which is liked by a customer is recommended by the recommender system. Similarly, False Positive (*fp*) denotes the case where an item disliked by a customer has been recommended by the system. False Negative (*fn*) is the case when an item of customer's liking has not been recommended by the system.

B. Evaluation Metric Discussion

We investigate two popular CF methods - user-based and item-based and then combine these recommendation methods with our framework to verify whether their performance is improved. We use Pearson's correlation coefficient [9] as the similarity metric for finding user-user similarity while item-item similarity is captured using Cosine-Based similarity metric [9].

User-based: For user-based algorithm, we use a representative similarity metric – Pearson's correlation, to measure the user-user similarities and use the user-based model in [9].

Item-based: For item-based algorithm, we use the vector cosine similarity model to compute the item-item similarities and use the item-based model in [9].

C. Results of Clustering

In this work, we cluster the users of the dataset according to their preferences for movie genres. As already discussed in section III, we represent the preferences of each user as a vector (user vector) in *n* dimensions, where n is total number of genres. The values of these dimensions are the normalized weights of the corresponding genre. We implement a CURE based clustering algorithm to cluster the users in *K* clusters. Once clustering is done, we next find the presence of *Attribute Autocorrelation* in the clusters using the *PA* metric (discussed earlier). *PA* metric measures the percentage of positive autocorrelation in a cluster. In Table III, we have shown the results of our clustering using different values of *K*.

TABLE III
RESULTS OF CLUSTERING

No. of clusters (*K*)	*PA* (Average)
2	28.35
4	48.34
8	60.24
16	68.6
32	75.65
64	65.21

From Table III, we can notice that for *K*=2, the average *PA* value across the 2 clusters is 28.35 while that for 4 clusters is 48.34. Since the average *PA* value for 4 clusters is more than corresponding *PA* value for 2 clusters, therefore, as discussed in section III (c), we next set K=8, and find the average *PA* value. In this way, we increment the value of *K* until we get a *K* value, K_{max} , for which the average *PA* value is less than the average *PA* value of the previous *K*. It can be noted that for *K*=64, the average *PA* value is 65.21, which is less than the average *PA* value for *K*=32, which is 75.65. Next, we carry out a binary search between 32 and 64 to find the value of K_{opt} ,the optimal value of *K* for which the average *PA* value is greater or equal to the average *PA* value for *K*=32. We experimentally find the optimal value of *K*, K_{opt} as 48 with average *PA* value of 79.2. We have given a comparative analysis of *Attribute Autocorrelation* (in terms of average *PA* value) for different values of *K* in Figure 3. Here *base* performance represents the *PA* value of the entire users' space (without clustering). We compare the base performance with the average *PA* values of the different clusters. From Figure 3, it can be clearly noted that our cluster based approach outperforms the base results. We can also note that the maximum *PA* is achieved for *K*=48, which is the optimal value of K.

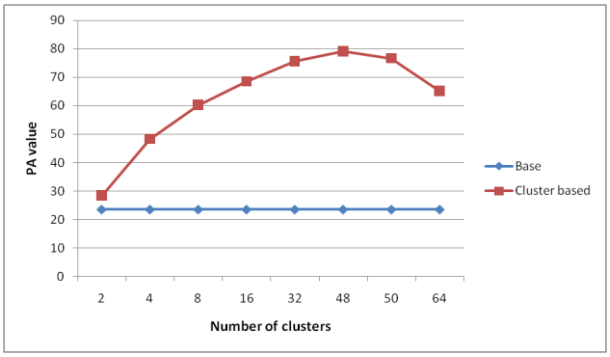

Fig. 3. Comparison of Attribute Autocorrelation.

D. Recommendation Performance on MovieLens-1M dataset

We apply the recommendation algorithm individually to each cluster to improve the scalability of the system. The clustering approach may however compromise the recommendation quality as we do not use the ratings of the entire users' space. The presence of *Attribute Autocorrelation* in the clusters (discussed in the previous section) further indicates that if we recommend only using the users (or items) present in the cluster of the target user, recommendation quality will improve.

For our experiment, MovieLens-1M ratings are randomly divided into 80% training data and 20% testing data. We report the performance of the recommendation algorithm on the MovieLens-1M dataset in Table IV. In the Table, we make a comparative analysis of the recommendation performance using different evaluation metrics. Here base performance indicates the performance of the algorithm using the entire users' space (without decomposition). We use MAE and RMSE to evaluate the prediction accuracy while Precision@10 and Recall@10 are used to evaluate the quality of the top-10 recommended items. The bold numbers indicate that its value has an obvious improvement over the corresponding values in the base. Our experiments are run on a computer with Core i3 - 2100 @ 3.10GHz x 4 CPU and 4 GB RAM.

TABLE IV
RECOMMENDATION PERFORMANCE ON MOVIELENS-1M DATASET

	P@10 (Avg)	R@10 (Avg)	MAE (Avg)	RMSE (Avg)
User-based				
Base	0.935	0.736	0.398	0.460
K = 2	0.858	**0.765**	0.419	0.571
K = 4	0.846	**0.77**	0.435	0.585
K = 8	0.841	**0.75**	**0.392**	0.589
K = 16	0.828	**0.746**	0.446	0.589
K = 32	0.817	0.724	0.486	0.619
K = 64	0.794	0.705	0.49	0.623
Item-based				
Base	0.842	0.715	0.412	0.521
K = 2	0.822	**0.724**	0.421	0.532
K = 4	0.817	**0.731**	**0.405**	**0.511**
K = 8	**0.849**	**0.753**	0.452	0.562
K = 16	**0.852**	0.704	0.456	0.589
K = 32	0.79	**0.721**	0.476	0.589
K = 64	0.782	0.692	0.481	0.602

From Table IV, it is clear that our algorithm is working fine with high precision and recall values as well as lower MAE and RMSE values. As for example, in Table IV, for *K*=8 (User-based case), we have an average Precision, Recall, MAE and RMSE of 0.841, 0.75, 0.392 and 0.589 respectively averaged across all the 8 clusters. Thus the algorithm performs better in terms of Recall and MAE values while in terms of Precision and RMSE, the base performance is slightly better. It can be observed that for the other values of *K*, the values of the evaluation metrics are quite comparable with the base. We report the average recommendation time (in seconds) per user in Figure 4. It can be clearly observed that the recommendation time reduces significantly when we cluster the users' space. As for example, the average recommendation time per user in the entire users' space (base) is 10.16 seconds. However, when it is partitioned into 64 clusters, the corresponding time is 0.74 seconds, which is significantly less (by about 92%) than the base. From Figure 4, one can easily find that our clustering based approach outperforms the base performance irrespective of the number of clusters. Analyzing the results of the experiments performed, we can conclude that our approach is efficient in reducing the running time without sacrificing the recommendation quality in most cases.

CONCLUSION

In this paper, we propose a Collaborative Filtering based recommendation technique which is made scalable by clustering the users' space. Our proposed approach deals with the *Scalability* problem of the CF process by applying the recommendation algorithm separately to the clusters. Experimental analysis using real datasets show that our model is efficient, scalable as well as maintain acceptable recommendation quality. In future, we have a plan to implement *Attribute Autocorrelation* on the basis of other demographic attributes of the user, such as gender, occupation and location with the aim of optimizing the clustering technique and the recommendation algorithm.

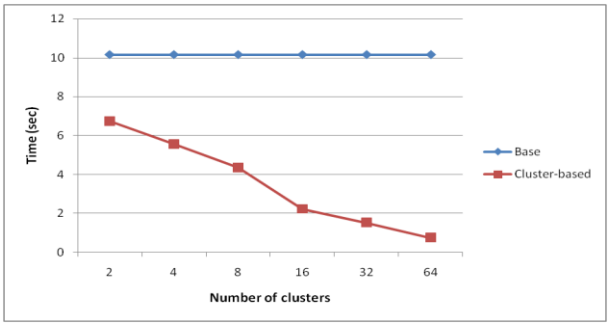

Fig. 4. Recommendation Time Per User.

REFERENCES

[1] J. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in Proc. of the 14th Conference on Uncertainty in Artificial Intelligence, 1998, pp. 43-52.

[2] A. Dalmia, J. Das, P. Gupta, S. Majumder, and D. Dutta. "Scalable hierarchical recommendations using spatial autocorrelation," in Proc. of the 3rd ASE International Conference on BigData Science and Computing, 2014.

[3] J. Das, S. Majumder, D. Dutta, and P. Gupta, "Iterative use of weighted voronoi diagrams to improve scalability in recommender systems," in Proc. of the 19th pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2015), 2015, pp. 605–617.

[4] J. Das, S. Majumder, and P. Gupta, "Spatially aware recommendations using k-d trees," in Proc. of the 3rd International Conference on Computational Intelligence and Information Technology (CIIT 2013), 2013, pp. 209–217.

[5] J. Han and M. Kamber, Data Mining Concepts and Techniques. Morgan Kaufmann Publishers, 2006.

[6] J. L. Herlocker, J. A. Konstan, and J. Riedl, "Explaining collaborative filtering recommendations," in Proc. of the 2000 ACM Conference on Computer Supported Cooperative Work, ser. CSCW '00, 2000, pp. 241–250.

[7] K. Miyahara and M. Pazzani, "Collaborative filtering with the simple bayesian classifier," in Proc. of the 6th Pacific Rim International Conference on Artificial Intelligence, 2000, pp. 679–689.

[8] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Recommender systems for large-scale e.commerce: Scalable neighborhood formation using clustering," in Proc. of the Fifth International Conference on Computer and Information Technology, 2002, pp. 158–167.

[9] X. Su and T. Khoshgoftaar, "A survey of collaborative filtering techniques," Advances in Artificial Intelligence, vol. 2009, 2009.

[10] S. Vucetic and Z. Obradovic, "Collaborative filtering using a regression based approach," Knowledge and Information Systems, vol. 7, no. 1, pp. 1–22, 2005.

[11] Y. Wang, S. Chan, and G. Ngai, "Applicability of demographic recommender system to tourist attractions: a case study on trip advisor," in Proc. of 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, 2012, pp. 97–101.

[12] G. Xue, C. Lin, Q. Yang, G. Xi, H. Zeng, Y. Yu, and Z. Chen, "Scalable collaborative filtering using cluster-based smoothing," in Proc. of the 28th annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2005, pp. 114–121.