An Adaptive Approach To Collaborative Filtering Using Attribute Autocorrelation

Joydeep Das*, Shreya Dugar†, Harsh Gupta†, Subhashis Majumder‡ and Prosenjit Gupta§
*The Heritage Academy, Kolkata, WB, India
Email: joydeep.das@heritageit.edu
†Dept of Computer Sc. and Engg., Institute of Engineering and Management, Kolkata, WB, India
Email: shreyadugar789@gmail.com
Email: harshguptahs@gmail.com
‡Dept of Computer Sc. and Engg., Heritage Institute of Technology, Kolkata, WB, India
Email: subhashis.majumder@heritageit.edu
§NIIT University, Neemrana, Rajasthan, India
Email: prosenjit_gupta@acm.org

***Abstract*—Recommender Systems (RS) provide a rich collection of tools for enabling users to filter through large amount of information available on the Web. *Collaborative Filtering* (CF) is one of the most widely used and successful techniques behind the development of RS. CF based RS recommend items by computing similarities between users and/or items. The items recommended to a user are those preferred by similar users. However, with the tremendous growth in users and items on the Web, CF algorithms suffer from serious *scalability* problems because similarities between every pair of users and/or items need to be computed during the training phase. In this paper, we propose a scalable CF method by using data clustering techniques. The proposed work partitions the users of the CF system using an *adaptive K-means* clustering algorithm and then use those partitions (clusters) to select the similar users (neighborhood) of a *target user*. In this work, we also try to determine the optimal value of K (number of clusters). Once a target cluster is determined, the neighborhood of the *target user* is selected by looking into the similarity score between the target user and all other users in that cluster. The basic idea is to partition the users of the RS and apply the CF based recommendation algorithm separately to the partitions. The cluster-based approach reduces the runtime of the system as we avoid similarity computations over the entire rating data. Experiments performed on MovieLens-1M dataset indicate that our method is efficient in reducing the runtime as well as maintaining an acceptable recommendation quality.**

***Keywords*—Collaborative Filtering, Recommender Systems, Clustering, Scalability.**

I. Introduction

The exponential increase in the volume of available digital information, electronic resources and on-line services has led to the problem of information overloading - where people face difficulties in finding their required information from an overwhelming set of choices. Today the problem is not about how to get correct information to make a decision, rather, how to make a right decision out of the enormous amount of information. This has initiated the development of information filtering systems like Recommender Systems (RS). 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 [17]. Some of the application domains for RS include recommendations for music CDs and DVDs[1], movies[2] and books[3].

Collaborative Filtering (CF) [1], [10] is one of the most widely used and successful technologies for RS. CF 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, to predict the likely preferences of a user based on the known preferences of the other similar users. CF algorithms can be classified into two categories: memory-based and model-based [19]. Memory-based algorithms perform similarity computations on the entire database to identify the K most similar users to the active user, and then predicts rating for the active user using the ratings of the K similar users. Examples include user-based/item-based CF algorithms with Pearson/Vector cosine correlation based similarity measures [2]. The two fundamental limitations associated with memory-based approaches are data sparsity and scalability. Since similarity computations are based on the co-rated items and therefore it is unreliable when the data are very sparse and the co-rated items are therefore few. With regards to scalability, memory-based algorithms often cannot scale well with the large number of users and items. Model-based approaches use the pure rating data to estimate or learn a model to make predictions. Examples of some well-known model-based CF techniques include Bayesian network approach [14], clustering CF approach [3] and the aspect models [11]. Model-based approaches are more scalable than memory-based approaches, but are often associated with expensive model building and updating.

Pennock et al. [16] proposed a hybrid memory- and model-based approach. Given a user's preferences for some items, they compute the probability that a user belongs to the same personality diagnosis by assigning the missing rating as a uniform distribution over all possible ratings [16]. Previous empirical studies have shown that the method is able to outperform several other approaches for collaborative filter-

978-1-4799-6908-1/15/$31.00

[1]http://www.dvdcdnow.com/
[2]http://www.criticker.com/
[3]http://www.abebooks.com/

ing [16], including the Pearson correlation coefficient method, the Vector space similarity method and the Bayesian network approach. However, the method neither takes the whole aggregated information of the training database into account nor considers the diversity among users when rating the non-rated items. From our point of view, the clustering-based smoothing could provide more representative information for the ratings.

In this paper, we present a clustering based CF algorithm that is perfectly applicable for large datasets which are nowadays common in e-commerce applications. The idea is to partition the users of the CF system using a clustering algorithm and use the clusters (partitions) as neighborhood. The proposed approach is likely to make the recommendations faster, because the size of the group that is to be analyzed to find neighbors is much smaller. In this work, we use a variant of *K-means* [9] clustering algorithm called the *adaptive K-means* clustering algorithm. We also seek to find the optimal value of K (the number of clusters) by using a binary search based technique. We further investigate how the recommendation quality of our algorithm compares to other algorithms under different practical circumstances. 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 based RS. 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 Based Recommender Systems

Collaborative Filtering (CF) systems recommend products to a target user based on the opinions or preferences of other similar users. The basic idea behind CF is that users who agreed on the past tend to agree on the future also. These systems employ statistical techniques to find a set of users known as neighbors, that have a history of agreeing with the target user. Once a neighborhood of a user is formed, CF systems use several algorithms to produce recommendations.

In a typical CF based application, there is a list of m users $U = \{u_1, u_2, \cdots, u_m\}$ and a list of n items $I = \{i_1, i_2, \cdots, i_n\}$. Each user u_i has rated a list of items I_{u_i}. There exists a distinguished user $u_a \in U$ called the *target user*, for whom the task of a CF algorithm is to suggest a list of items that the user may like. Most of the CF algorithms typically build a neighborhood of like-minded users using some similarity measures like Pearson's correlation coefficient or vector cosine similarity. The idea is that given a *target user* u_a, compute her n similar users $\{u_1, u_2, \cdots, u_n\}$ and predict u_a's preferences based on the preferences of $\{u_1, u_2, \cdots, u_n\}$. Once the neighborhood formation is complete, CF systems generate recommendations that can be of two types:

Rating prediction: It is a numerical value, $R_{a,j}$, which predicts the likeliness (rating) of item i_j for the *target user* u_a. This predicted value should be within the same scale (e.g., from 1 to 5 rating scale) as the opinion values provided by u_a.

Recommendation: It is a list of *Top-N* items, $TI_r = \{T_{i1}, T_{i2}, \cdots, T_{pN}\}$, that the *target user* will like the most. The recommended list must not contain the items already purchased by the *target user*.

B. Clustering based Recommender Systems

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. This enables the system to be scalable. 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. [20] proposed a CF system based on K-means clustering in order to smooth the unrated data for individual users according to the clusters. Jiang et al. [12] implemented a cluster-based collaborative filtering scheme on the basis of an iterative clustering method that exploits the inter-relationship between users and items. To address the scalability issue, Sarwar et al. [18] and O'Conner et al. [15] 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, George and Merugu [8] used a collaborative filtering approach on the basis of a weighted co-clustering algorithm that involves simultaneous clustering of users and items. Das et al. [7] proposed a DBSCAN based clustering algorithm for clustering the users, and then use algorithms from voting theory to recommend items to an active user depending on the cluster into which it belongs. They also proposed a decomposition based recommendation algorithm using multiplicatively weighted Voronoi diagram to enhance the scalability of the system [6]. Similar decomposition based recommendation techniques can also be found in [4] and [5].

III. Proposed Scheme

In this work, we propose a scalable CF based recommendation algorithm using a clustering technique. The use of clusters integrate the advantages of both memory-based and model-based CF approaches. Traditional CF algorithms typically search the whole database to find the neighbors of a *target user*. This method suffers from scalability problem when more and more new users and items are added to the database. In clustering approach, the feature of the group of users in a cluster is represented by the centroid of the cluster. The centroid is represented as an average rating over all the users in the cluster. For a *target user*, we need to compute the similarity between each cluster and the *target user*, and the most similar cluster is used as

the neighborhood of the *target user*. We formally present our algorithm below.

Algorithm: Cluster-based Collaborative Filtering
Step 1: Apply the clustering algorithm to produce n partitions of users. Let U represents the entire set of users. We partition the set U into p partitions $U_1, U_2, \cdots, U_n$, where $U_i \cap U_j = \phi$ for $1 \leq i, j \leq n$; and $U_1 \cup U_2, \cdots \cup U_n = U$.

Step 2: Choose the neighborhood for a *target user*, u_a. If $u_a \in U_i$, then the entire cluster U_i is used as the neighborhood.

Step 3: Calculate the similarity $sim(u_a, u)$ for each user u in U_i and select the *top-K* most similar users (nearest neighbors) of u_a. The similarity between two users u and v is computed using Pearson's correlation as follows.

$$Sim(u,v) = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}$$

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

Step 4: Once the neighborhood is defined, predict the rating of a particular item for u_a by the behaviors of the K nearest neighbors. The prediction score $P_{a,i}$ for a user a, on a certain item i, is calculated using the following formula.

$$P_{a,i} = \bar{r}_a + \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u) w_{a,u}}{\sum_{u \in U} |w_{a,u}|}$$

where $\bar{r}_a$ and $\bar{r}_u$ are the average ratings for the user a and user u on all other rated items, and $w_{a,u}$ 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 *K-means* based clustering algorithm to cluster the users of the system. The number K is input to the algorithm that specifies the desired number of clusters. The *K-means* clustering method creates K clusters each of which consists of the users who have similar preferences among themselves. In this method, we first select K users arbitrarily as the initial centers of the K clusters respectively. For each of the remaining users, calculate the distance between the user and all the cluster centers and assign the user 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[4] 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. The genres present in the dataset are Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War and Western. In this work, we cluster the users of the dataset according to their preferences of movie genres. Assuming genre is sufficient to capture how users rate movies, we represent the preferences of each user as a vector (*preference 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. The mapping between the rating and its corresponding genre weight is shown in Table I.

Table I. MAPPING BETWEEN RATING AND GENRE WEIGHT

Movie rating	Genre weight
1	0.2
2	0.4
3	0.6
4	0.8
5	1.0

Table II. AN EXAMPLE OF MOVIE PREFERENCES FOR USER 1

Movie	Genre	Rating
Broken Arrow	Action, Thriller	5
Batman Forever	Action, Adventure, Comedy, Crime	4
Something to Talk About	Comedy, Drama, Romance	3
Random Hearts	Drama, Romance	2

Table III. AN EXAMPLE OF MOVIE PREFERENCES FOR USER 2

Movie	Genre	Rating
Rumble in the Bronx	Action, Adventure, Crime	5
Flesh and Bone	Drama, Mystery, Romance	4
Twice Upon a Yesterday	Comedy, Drama, Romance	4
Night of the Creeps	Comedy, Horror, Sci-Fi	2

We have given an example of movie preferences for two users in Table II and Table III, while the corresponding *preference vectors* are shown in Table IV and Table V respectively. In Table IV, the weights of the different genres are calculated as follows. For the Action genre, user 1 has rated two movies 'Broken Arrow' and 'Batman Forever' (Table II). Since the ratings of the two movies are 5 and 4 respectively, and as a result the Action genre is weighted by $1.0 + 0.8 = 1.8$. We can also notice that apart from Action, the movie 'Batman Forever' also belongs to Adventure, Comedy and Crime genres. Therefore all the three genres will also be weighted by 0.8. We similarly calculate the weights of all the genres. If a user do not have any preference for a particular genre then the weight of that genre is considered as 0.0. We follow the similar approach for creating the *preference vector* of user 2 (Table V). Continuing the above mentioned approach, we create *preference vectors* for all the remaining users. Then we implement our clustering algorithm using these *preference vectors* to cluster the users' space into K clusters. 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

[4]http://www.movielens.org/

Table IV. PREFERENCE VECTOR FOR USER 1

Movie genre	Genre weight
Action	1.8
Adventure	0.8
Animation	0.0
Children's	0.0
Comedy	1.4
Crime	0.8
Documentary	0.0
Drama	0.6
Fantasy	0.0
Film-Noir	0.0
Horror	0.0
Musical	0.0
Mystery	0.0
Romance	0.6
Sci-Fi	0.0
Thriller	1.0
War	0.0
Western	0.0

Table V. PREFERENCE VECTOR FOR USER 2

Movie genre	Genre weight
Action	1.0
Adventure	1.0
Animation	0.0
Children's	0.0
Comedy	1.2
Crime	1.0
Documentary	0.0
Drama	1.6
Fantasy	0.0
Film-Noir	0.0
Horror	0.4
Musical	0.0
Mystery	0.8
Romance	1.6
Sci-Fi	0.4
Thriller	0.0
War	0.0
Western	0.0

verify whether users with similar demographic profiles such as age, gender, occupation, location, etc. are grouped in the same clusters. We use the value of *attribute autocorrelation* as a parameter for determining the optimal number of clusters.

B. Attribute Autocorrelation

Analogous to the notion of spatial autocorrelation like Geary's or Moran's index used in GIS [13], we define a measure called *attribute autocorrelation*. It measures the covariance 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 user, 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. In this work, we define an autocorrelation index using the gender of the users as follows.

We consider one movie at a time and find how the ratings of that movie varies with respect to the gender 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

$$c_{ij} = (r_i - \bar{r})(r_j - \bar{r}) \tag{1}$$

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

Let g_i and g_j denotes the respective gender user i and user j. We can define a gender similarily w_{ij} between the two users as follows.

$w_{ij} = 1$ if both i and j belongs to the same gender, otherwise $w_{ij} = 0$.

Now the autocorrelation index c can be defined as

$$c = \frac{\sum_i \sum_j w_{ij} c_{ij}}{\sigma^2 \sum_i \sum_j w_{ij}} \tag{2}$$

where

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

Here σ^2 is the variance. The positive c value indicates that the presence of attribute autocorrelation, i.e., people with similar gender rated similarly while negative value implies that the ratings are dissimilar. A zero value indicates that the ratings are distributed independently of gender. We measure the value of this autocorrelation index in all the clusters to verify whether people with same gender are grouped in the same clusters. This *attribute autocorrelation* value will be used to determine the best possible value of K (number of clusters). Next, we introduce an adaptive approach to find this optimal value of K.

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 clusters, therefore we define a metric PA, which measures the percentage of positive autocorrelation in a cluster. Next, an adaptive procedure is followed to determine the suitable value of K. Initially, we start with $K = 5$ and find the average PA value across all the five 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 $(2\log_2 K_{opt} - 1) = O(\log K_{opt})$ iterations.

D. Recommendation with Clusters

In order to improve the *scalability* of the system, we apply the recommendation algorithm separately to the clusters to avoid computations over the entire dataset. The main idea is to apply some CF algorithm in each cluster and try to merge the prediction results together. However, it may compromise the recommendation quality as users in two different clusters may have similarity in the rating pattern. Our objective is to reduce the running time without compromising the recommendation much. For a pure CF method, the only input is the user-item matrix and the output is the prediction score for each missing value in the matrix. For each cluster, we actually get a submatrix from the original big user-item matrix, and therefore we can directly apply any CF method without any modification. For a *target user*, we first extract the genre preferences of this user and then represent the preferences of this *target user* as a vector (*preference 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 the cluster mean of all the clusters is calculated and the user is the placed in the minimum distance cluster. Then we recommend items (movies) of interest to the *target user* using the preferences of the other similar users in the cluster.

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 3: Calculate the distance between the *target user* and the mean of all the clusters.
Step 4: Assign the *target user* to the minimum distance cluster.
Step 5: Recommend *top-N* movies using any CF method.

IV. Experimental Framework

We conducted several experiments to evaluate the effectiveness of the proposed method. In this section, we describe the experimental settings in detail. 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 - observed items (80%) for training and held-out items (20%) for testing. Ratings for the held-out items were to be predicted.

A. Evaluation Metrics

In order to evaluate the prediction accuracy of our recommendation algorithm, we use Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) [19]. Since our system produces a list of *Top-N* recommended items, therefore MAE and RMSE are not suffcient to truly evaluate the recommendation quality. Therefore we also use $F1$ score metric [10] to evaluate the quality of the ranked list. $F1$ score requires another two metrics, *Precision* (P) which is the success in retrieving items that is of users interest, and *Recall* (R), which is the success in retrieving items that are truly of interest in relation to the number of all items that claim to be of interest. Now, we formally define the evaluation metrics.

MAE: MAE is defined as the average of the absolute error. Absolute error is the difference between the predicted rating and actual rating. Let the actual user ratings be, $\{r_1, r_2, \cdots, r_n\}$, and predicted ratings are, $\{p_1, p_2, \cdots, p_n\}$, where n is the number of items. Then Absolute error,

$$E = \{e_1, e_2, \cdots, e_n\} = \{p_1 - r_1, p_2 - r_2, \cdots, p_n - r_n\}$$

and

$$MAE = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Any prediction algorithm tries to minimize the MAE.

RMSE: RMSE is similar to MAE and is biased to provide more weights to larger errors.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} e_i^2}{n}}$$

F1 score: $F1$ score metric can be defined as

$$F1 = \frac{2 * P * R}{P + R}$$

where

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

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 a 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. Comparisons

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 [19] as the similarity metric for finding user-user similarity while item-item similarity is captured using Cosine-Based similarity metric [19].

User-based: For user-based algorithm, we use a representative similarity metric-pearson correlation, to measure the user-user similarities and use the user-based model in [19].

Item-based: For item-based algorithm, we use the vector cosine similarity measure to compute the item-item similarities and use the item-based model in [19].

C. Experimentation with Clustering Algorithm

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 (preference vector) in n dimensions, where n is total number of genres. The values of these dimensions are the weights of the corresponding genre. We implement a K-means 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 VI, we have shown the results of our clustering using different values of K.

Table VI. Results of Clustering

No. of clusters (K)	PA (Average)
5	35.22
10	52.27
20	67.81
40	74.38
80	68.21

From Table VI, we can notice that the average PA value across all the 5 clusters is 35.22 while that for 10 clusters is 52.27. Since the average PA value for 10 clusters is more than corresponding PA value for 5 clusters, so we next try with $K = 20$, 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 $K = 80$, the average PA value is 68.21, which is less than the average PA value for $K = 40$. Next, we carry out a binary search between 40 and 80 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 $K = 40$. In our work, the optimal value of K, K_{opt} is 52 having an average PA value of 78.5. We have given a comparative analysis of the presence of *Attribute Autocorrelation* (in terms of average PA value) for different values of K in Figure 1. 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 1, 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 $K = 52$, which is the optimal value of K.

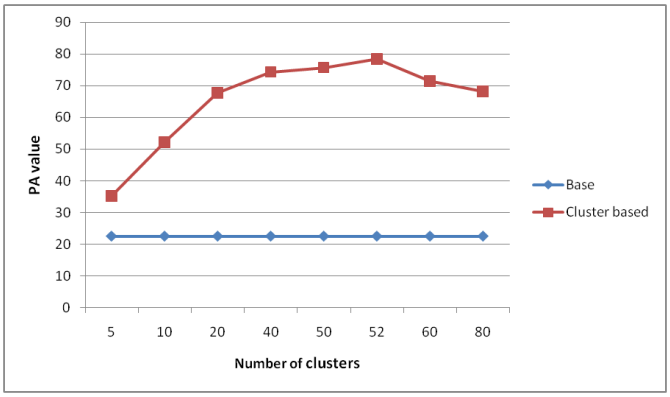

Figure 1. Comparison of Attribute Autocorrelation

Figure 2. Prediction Performance on MovieLens-1M Dataset using MAE

D. Experimentation with Recommendation Algorithm

In this work, we apply the recommendation algorithm separately to the clusters in order to reduce the running time of the system. Our objective is to execute the algorithm only using the ratings of that particular cluster, which allows us to avoid the similarity computations over the entire rating dataset. Note that, it may degrade the recommendation quality as we lose some of the ratings in this process.

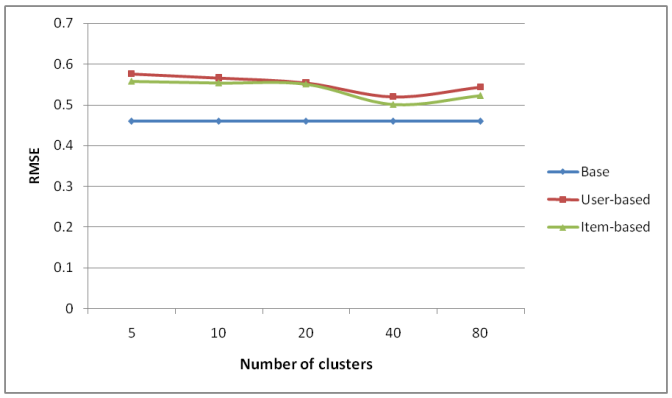

Figure 3. Prediction Performance on MovieLens-1M Dataset using RMSE

We report and compare the prediction accuracy of the recommendation algorithm in terms of MAE and RMSE in Figure 2 and Figure 3 respectively. In the Figures, Base performance indicates the prediction performance of the algorithm using the entire users' space (without clustering). The Base performance is compared with the performance of the algorithm in the clusters. As for example, in Figure 2, we can notice that the average MAE value (for the Item-based case) over all the clusters is less than the base value when the number of clusters is 40. In other cases, the MAE value is nearly equal to the base value. Similarly, in Figure 3, it can be noted that in terms of RMSE value, the performance of our clustering based approach is comparable with the Base performance. In this work, we also use $F1@K$ to evaluate the quality of the *top-K* recommended items. We report and compare the recommendation performance in Table VII. Note that, we present $F1$ ($F1@10$) score on position 10. Here base performance indicates the performance of the algorithm using the entire users' space (without clustering). We compare the overall performance of the algorithm in the clusters with the Base performance. The bold numbers indicate that its value has an improvement over the base value. In Table VII, the column $Time$ reports the overall running time of our recommendation algorithm. Our experiments are run on a computer with Core i3 - 2100 @ 3.10GHz x 4 CPU and 4 GB RAM.

Table VII. PERFORMANCE COMPARISONS ON MOVIELENS-1M DATASET IN TERMS OF F1 AND TIME

No. of clusters (K)	F1@10		Time
	User-based	Item-based	
1 (Base)	0.821	0.797	925.3
5	0.793	0.77	**523.5**
10	**0.838**	0.785	**320.3**
20	**0.842**	**0.818**	**210.6**
40	**0.849**	**0.821**	**115.32**
80	0.802	0.786	**90.5**

From Table VII, it is clear that our algorithm is working fine with high $F1$ values. As for example, for $K = 10$ (User-based case), we have an average $F1$ score of 0.838

averaged across all the 10 clusters, while the corresponding Base value is 0.821. However, for the Item-based case, the Base value is slightly better. For the other values of K, we can notice similar results. It can be noted that the runtime of the algorithm is reduced significantly when we cluster the users' space. As for example, the time required to test the recommendation algorithm using all the users of the dataset (Base) is 925.3 minutes. However, when it is partitioned into 80 clusters, we require a overall runtime of 90.5 minutes for recommending all the users of the 80 different clusters, which is significantly less (by about 90%) than the Base. For the other values of K, we have similar results. The results depicted in Table VII conclusively show that our approach is efficient in reducing the runtime. We report the average recommendation time (in seconds) per user in the entire users' space (Base) and the corresponding time in the different clusters in Figure 4. One can clearly observe that the runtime per user improves significantly when we divide the users' space into clusters, and as the number of clusters increase, recommendation time decrease.

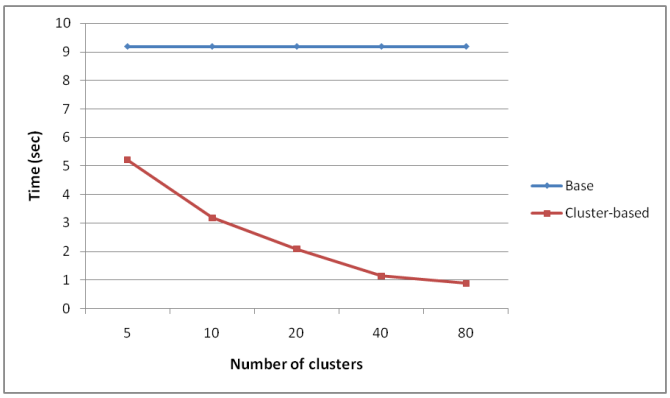

Figure 4. Recommendation Time Per User

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 much. This establishes that our method is scalable and it can be used to deal with even bigger datasets with the same resources.

V. Conclusion

In this paper, we have presented a scalable clustering based Recommender System. The increasing volume of customers on the Web along with the massive already existing customer data poses serious *scalability* issues for Collaborative Filtering based recommender systems. Our proposed approach deals with the *scalability* problem of the recommendation task by applying the recommendation algorithm separately to the clusters. The results obtained suggest that our recommendation approach provides satisfactory recommendation quality compared to the basic CF approach and at the same time improves the runtime significantly. The focus of our future work is to use other clustering techniques in order to generate much faster recommendations as well as provide better recommendation quality. We also have a plan to implement *Attribute Autocorrelation* using other demographic attributes of the user, such as age, occupation, and location. How this can be leveraged is a matter of future research.

References

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[2] J. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 43–55.

[3] S. Chee, J. Han, and K. Wang, "Rectree: an efficient collaborative filtering method," in *Proceedings of the 3rd International Conference on DataWarehousing and Knowledge Discovery*, 2001, pp. 141–151.

[4] A. Dalmia, J. Das, P. Gupta, S. Majumder, and D. Dutta, "Scalable hierarchical recommendations using spatial autocorrelation," in *Proceedings of the 3rd ASE International Conference on Big Data Science and Computing (BigDataScience, 2014)*, 2014.

[5] J. Das, A. K. Aman, P. Gupta, A. Haider, S. Majumder, and S. Mitra, "Scalable hierarchical collaborative filtering using bsp trees," in *Proceedings of the International Conference on Computational Advancement in Communication Circuits and Systems (ICCACCS 2014)*, 2014, pp. 269–278.

[6] J. Das, S. Majumder, D. Dutta, and P. Gupta, "Iterative use of weighted voronoi diagrams to improve scalability in recommender systems," in *Proceedings of the 19th pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2015)*, vol. LNAI 9077, 2015, pp. 605–617.

[7] J. Das, P. Mukherjee, S. Majumder, and P. Gupta, "Clustering-based recommender system using principles of voting theory," in *Proceedings of the 2014 International Conference on Contemporary Computing and Informatics (IC3I)*, 2014, pp. 230–235.

[8] T. George and S. Merugu, "A scalable collaborative filtering framework based on co-clustering," in *Proceedings of the Fifth IEEE International Conference on Data Mining*, 2005, pp. 625–628.

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

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

[11] T. Hofmann, "Latent semantic models for collaborative filtering," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 89–115, 2004.

[12] X. Jiang, W. Song, and W. Feng, "Optimizing collaborative filtering by interpolating the individual and group behaviors," in *APWeb*, 2006, pp. 568–578.

[13] C. P. Lo and A. K. W. Yeung, *Concepts and Techniques of Geographic Information Systems*. Prentice Hall, 2007.

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

[15] M. O'Connor and J. Herlocker, "Clustering items for collaborative filtering," in *Proceedings of the ACM SIGIR Workshop on Recommender Systems*, 1999.

[16] D. Pennock, E. Horvitz, S. Lawrence, and C. Giles, "Collaborative filtering by personality diagnosis: a hybrid memory- and model-based approach," in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.

[17] P. Resnick and H. Varian, "Recommender systems," *Communications of the ACM*, vol. 40, no. 3, pp. 56–58, 1997.

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

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

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