Clustering-Based Recommender System Using Principles of Voting Theory

Joydeep Das*, Partha Mukherjee†, Subhashis Majumder‡ and Prosenjit Gupta‡
*The Heritage Academy, Kolkata, WB, India
Email: joydeep.das@heritageit.edu
†College of Information Science and Technology, Pennsylvania State University, USA
Email: pom5109@ist.psu.edu
‡Dept of Computer Sc. and Engg, Heritage Institute of Technology, Kolkata, WB, India
Email: subhashis.majumder@heritageit.edu
Email: prosenjit_gupta@acm.org

***Abstract*—*Recommender Systems* (RS) are widely used for providing automatic personalized suggestions for information, products and services. *Collaborative Filtering (CF)* is one of the most popular recommendation techniques. However, with the rapid growth of the Web in terms of users and items, majority of the RS using CF technique suffer from problems like *data sparsity* and *scalability*. In this paper, we present a Recommender System based on data clustering techniques to deal with the *scalability* problem associated with the recommendation task. We use different voting systems as algorithms to combine opinions from multiple users for recommending items of interest to the new user. The proposed work use DBSCAN clustering algorithm for clustering the users, and then implement voting algorithms to recommend items to the user depending on the cluster into which it belongs. The idea is to partition the users of the RS using clustering algorithm and apply the Recommendation Algorithm separately to each partition. Our system recommends item to a user in a specific cluster only using the rating statistics of the other users of that cluster. This helps us to reduce the running time of the algorithm as we avoid computations over the entire data. Our objective is to improve the running time as well as maintain an acceptable recommendation quality. We have tested the algorithm on the Netflix prize dataset.**

Keywords*—*Recommender Systems, Clustering, Voting System, Scalability.

I. Introduction

In everyday life, we often face a situation in which we need to make choices without sufficient personal experience. Ever increasing volume of information on the web has created the need for automated filtering, refinement and personalized presentation of information to users to help decision making. There have been efforts to design information filtering systems that filter and present information according to the preferences of the individual user. Recommender Systems (RS) form a subclass of information filtering systems that helps the users in their decision making process by suggesting items that the users may prefer.

RS are being used in a number of e-commerce sites to help customers in finding suitable products [13]. Such systems should be able to identify the user preferences for items in the application domain. Typical application domains for RS include recommendations for music CDs and DVDs[1], movies[2] and books[3]. Majority of the RS use Collaborative Filtering (CF) techniques [1], [7], [15], to predict the likely preferences of a user based on the known preferences of the other similar users. However, these CF based RS require computations that are very expensive and grow polynomially with the number of users and items in the database. To address this scalability problem, we propose a recommendation method using data clustering techniques and voting algorithms. Our proposed system avoids the costly similarity computations of the CF process by applying a voting[4] based recommendation algorithm separately to each cluster. Note that though we partition the users' space into smaller clusters and applied the voting based Recommendation Algorithm individually to the clusters, it does not mean that two users in different clusters cannot have similarity in the rating patterns. It may also happen that the recommendation quality degrades as we recommend only using the data of a particular cluster. However, our goal is to reduce the overall running time without sacrificing the recommendation quality much. This ensures scalability, allowing us to tackle bigger datasets. To demonstrate the applicability of our method, we are developing a Movie Recommender System that will cater to the interests of users.

RS for movies have many dimensions. Each dimension has a set of attributes or elements. One of these dimensions may describe the type of a movie (genre) and contain elements like horror, comedy, tragedy, musical, action, etc. RS usually combine the value/rating of the attributes of every dimension according to some evaluation criteria to obtain a recommendation rating of an item. In the proposed Movie Recommender System, a classical voting method is used as the evaluation scheme. Principles of voting theory have been satisfactorily used for many years in multi-agent systems [4] regarding group decision making that maximizes social welfare. So, the use of voting theory in the proposed system promises recommendation that optimizes the user preferences. In this work, we use DBSCAN [6] clustering algorithm for clustering the users in the dataset according to their respective preference of movie genres. Experiments performed indicate that our method is effective in enhancing the scalability of the Recommender System.

[1]http://www.dvdcdnow.com/

[2]http://www.criticker.com/

[3]http://www.abebooks.com/

[4]http://en.wikipedia.org/wiki/Voting_system

The rest of the paper is organized as follows. In section II, we review prior research related to clustering based RS and voting theory. Section III outlines our proposed scheme while sections IV and V present our clustering scheme and voting algorithms respectively. Section VI details our recommendation algorithm and in section VII, we report and analyze the experimental results. We conclude discussing our future research directions in Section VIII.

II. Related Works

A. Clustering Based Recommender Systems

Clustering algorithms are used in RS to identify clusters of users having similar preferences. 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. [17] 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. [8] implemented a cluster-based collaborative filtering scheme on the basis of an iterative clustering method that exploits the inter-relationship between users and items. In this model, both users and items are first clustered using the K-means algorithm, then a predicted rating is generated over user classes and item classes. To address the scalability issue, Sarwar et al. [12] clustered the complete user set on the basis of user similarity and used the cluster as the neighborhood. In contrast, O'Conner et al. [11] used clustering algorithms to partition the item set on the basis of user rating data. Das et al. [3] in their location-aware system, used Voronoi Diagrams to tessellate the plane and applied the Recommendation Algorithm separately in each Voronoi cell. With the same perspective, George and Merugu [5] used a collaborative filtering approach on the basis of a weighted co-clustering algorithm that involves simultaneous clustering of users and items.

B. Voting Algorithms

Voting algorithms can be used in selecting choices from several conflicting alternatives. Implementation of voting theory can be found in many applications. Lang [9] introduced the notion of combinatorial vote, where a group of agents (or voters) express preferences and come to a common decision concerning a set of non-independent variables to assign. He studied two key issues pertaining to combinatorial vote, namely preference representation and automated choice of an optimal decision. Webber et al. [16] presented a mechanism for diagnosing and modelling learner's conceptions on the basis of a theoretical model of conceptions. They applied techniques from voting theory for group-decision-making. Mukherjee et al. [10] developed an online movie recommendation system using principles of voting theory. Their system provide multiple query modalities by which the user can pose unconstrained, constrained, or instance-based queries.

In this work, we try to address the scalability problem associated with Recommendation Algorithms by implementing data clustering and voting techniques.

III. Proposed Scheme

In this work, we use the Netflix Prize dataset[5] for testing our Recommendation System. The dataset contains 17770 movie rating files, 480189 users who have rated the movies and 29 genres. Ratings are on a five star (integral) scale from 1 to 5. The proposed work recommend movies to new users according to their preferred genres. Our system has a provision for the users to specify their preferences for genres. We have considered a maximum of four genres per user in which he can express his choices. Our approach learns the preferences of the individual users based on the genres and builds a user database. Clustering algorithm is applied to cluster the users in the database so that the users in the same cluster exhibit similar tastes. We use DBSCAN clustering algorithm to cluster the users with respect to their preferences of movie genres. Once the clusters are created, predictions for an individual can be made by averaging the opinions of the other users in that cluster. Voting based Recommendation Algorithm is then used in the individual clusters for selecting the recommended items for a new user.

Our Recommender System provides the following major functionalities to its users.

- The system stores user preferences in the form of user provided weights of different attributes like comedy, drama, action, etc. of the genre dimension. The system also stores the relative importance of each of these attributes as specified by the users (e.g., whether the user rates the comedy above drama).
- The system recommends a few most probably likeable movies to the users. Principles of voting theory are applied to recommend movies on the basis of the cluster to which the user belongs.

IV. Cluster Formation

Clustering techniques are used to identify groups of users having similar characteristics. The proposed work use a DBSCAN based clustering process to cluster the users of the dataset. Before implementing our clustering algorithm, we need to perform the following preprocessing on the dataset.

A. Preprocessing

The training dataset of Netflix contains 17770 rating files one per movie. Each movie rating file contains the ratings given by the customers to that movie. We first find the average rating of each movie given by all the customers. Next, we create a movie database having the following fields: movie_id, year_of_release, title, avg_rating, genre. Here avg_rating is the average rating of the movie. Our system recommend movies from this movie database according to the user's preference of movie genres. Since we recommend according to the movie genres, we build a training set of user preferences for movie genres along with genre weights. Genre weights belongs to the set [0,1}. Weights are assigned to each genre according to their order of preference and then they are normalized. In this work, we consider four genres per user from the available genres present in the movie database. In order to build the training

[5]http://en.wikipedia.org/wiki/Netflix_Prize

set, we randomly select four genres per user from the available 29 genres, and the corresponding weights of the genres are also assigned randomly. Clustering technique is applied to this training set to move the users to the corresponding clusters according to their preferred genres. Table I shows a sample of the training set of users' preferences for movie genres. The normalized genre weights are calculated as follows.

Consider the first row of Table I. Here the genre weights are $Short = 0.93$, $Comedy = 0.71$, $Action = 0.53$ and $Romance = 0.07$. Normalized weight for $Short = 0.93/(0.93+0.71+0.53+0.07) = 0.41$. Similarly normalized weight for $Comedy = 0.71/(0.93+0.71+0.53+0.07) = 0.31$. In this way, we find the normalized weights from the corresponding genre weights.

Table I. SAMPLE TRAINING SET OF USERS PREFERENCE FOR GENRES

Genre	Genre Weight	Normalized Genre Weight
Short Comedy Action Romance	0.93 0.71 0.53 0.07	0.41 0.31 0.23 0.03
Adventure Horror Drama Crime	0.89 0.70 0.45 0.04	0.42 0.33 0.21 0.01
Sci-fi Sports Animation Musical	0.82 0.68 0.49 0.09	0.39 0.32 0.23 0.04
Musical Mystery Thriller Western	0.99 0.75 0.45 0.17	0.41 0.31 0.19 0.07
Biography Adventure Fantasy Horror	0.95 0.67 0.48 0.13	0.42 0.30 0.21 0.05

B. Clustering Algorithm

The proposed work partitions the users in the dataset according to their preference of movie genres. Let U represent the entire set of users. We partition the set U into p partitions $U_1, U_2, \cdots, U_p$, where $U_i \cap U_j = \phi$ for $1 \leq i, j \leq p$; and $U_1 \cup U_2, \cdots \cup U_p = U$. For any user u, if $u \in U_i$ then the recommendation algorithm uses the entire cluster U_i as the neighborhood.

DBSCAN [6] algorithm requires two parameters: (i) ε-neighborhood (eps), which defines the radius of the neighborhood of a given object and (ii) *MinPts*, the minimum number of points required to form a cluster. The algorithm starts with an arbitrary starting point that has not been visited. The ε-neighborhood of this point is retrieved, and if the number of points present in it $\geq$ *MinPts*, a cluster is started. Otherwise, the point is labelled as noise. This point might later be found in a different cluster. If a point is found to be part of a cluster, its ε-neighborhood is also part of that cluster. Thus, all points that are found within the ε-neighborhood are added to that cluster. This process continues until the cluster is completely found. Then a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

In order to cluster the users of the training set, we represent the preferences of each user as a vector in k dimensions, where k is the total number of genres. The values of these dimensions are the normalized weights of the corresponding genre. A user can express his movie preferences in 4 genres (maximum). Suppose we have a user A with the preferred set of genres $\{Short, Comedy, Action, Romance\}$, and the corresponding genre weights being $\{0.93, 0.71, 0.53, 0.07\}$. Then the normalized weights of the genres are $\{0.41, 0.31, 0.23, 0.03\}$. So the preference vector for user A will have the following values: $Short = 0.41, Comedy = 0.31, Action = 0.23, Romance = 0.03$. The values of the remaining $(n-4)$ genres are set to 0.0. In this way, we represent each user of the training set as a vector. We have implemented our user clustering algorithm using these user vectors. Euclidean distance measure is used to find the distance between two user vectors. Clustering process produces clusters as well as noise points. Noise points represent the users that do not belong to any cluster. Users belonging to the clusters actively participate in the voting process, the noise points remain inactive.

V. VOTING SCHEME

Clustering algorithm partitions the entire users' space into smaller clusters. In order to select the most popular items in a cluster, we apply a voting based algorithm individually to the clusters. Now we explain our voting algorithm in details.

Suppose there is a list of alternatives $A = \{a_1, a_2, a_3, \ldots, a_i\}$, a set of voters $V = \{v_1, v_2, v_3, \ldots, v_j\}$ and a preference function P, that returns the rank ordering of the alternatives given by a voter. A voting scheme will produce a ranking of the alternatives. A ranked voting method allows each voter to rank the candidates in order of preference. Some ranked methods also allow voters to give multiple candidates the same ranking. Straffin has listed several criteria to rate the desirability of the outcome of different voting rules[14].

In our work, we use the Borda count voting rule. The Borda count is a ranked voting method. It works as follows: it assigns points to an alternative, on the basis of its position in a voter's preference list. The last place alternative gets 1 point, the second to last gets 2 points, and so on until the first place, which gets n points. If a voter is indifferent to 2 or more alternatives, then each one is assigned the average of the alternatives. For example, if a voter has the preference list of $P = \{a, \{b, c, d\}, e, f\}$, then the points are awarded as follows: f gets 1, e gets 2, d, c, and b each get 4, and finally a gets 6. The alternative that receives the highest number of votes from all the voters is the Borda count winner.

We use the genre dimension to provide a Borda count for each of the elements of that dimension. Each user has a preference genre list like $G = \{Action, Drama, Comedy, Horror\}$ and say the corresponding set of normalized weights $W = \{0.35, 0.31, 0.23, 0.09\}$. Our work assigns points to the different genres present in the preference list as follows. The last genre option (4th) gets 1, the 3rd one gets 2, the 2nd one gets 3 and the 1st alternative gets 4 points. For example Horror = 1, Comedy = 2, Drama = 3 and Action = 4.

We assign points to genre alternatives as above for all the users in each cluster. Next, we define a Rating Vector (RV) for each of the clusters. A Rating Vector is a vector having dimension equal to the total number of genres. Every element of this vector corresponds to a specific movie genre. RV contains the total vote received by the different genres from all the users in that cluster. The genre receiving the highest vote is the winner in that cluster. The scheme is delineated in its algorithmic form as follows:

In Algorithm 1, n is the total number of clusters, cl is an array of n clusters and rv is the rating vector one per each cluster. Procedure CalculateRating() calculates the total vote received by the genres from all the users in a cluster. Here s is the size of the cluster c, i.e., total number of users in the cluster and r is the rating vector of that cluster. $v[i]$ is the preference vector of the ith user. SumRating() function

Algorithm 1: Voting(n, cl[ ])

```
1: $Vector\ rv = new\ Vector[n]$ {rv : Rating Vector}
2: for $i = 0$ to $n$ do
3:     $rv[i] = calculateRating(cl[i])$
4: end for
5: for $i = 0$ to $n$ do
6:     $display\ rv[i]$
7: end for
```

Algorithm 2: CalculateRating(c)

```
1: $s = c.size()$
2: $Vector\ r = new\ Vector()$
3: for $i = 0$ to $s$ do
4:     $r = sumRating(r, v[i])$ {$v[i]$ : ith user vector}
5: end for
6: return $r$
```

Algorithm 3: SumRating(r1,v1)

```
1: $s1 = r1.size()$
2: $Vector\ r2 = new\ Vector()$
3: for $i = 0$ to $s1$ do
4:     $sum = r1[i] + v1[i])$
5:     $r2[i] = sum$
6: end for
7: return $r2$
```

calculates the cumulative vote received by the different genres from the users of the cluster. Here $s1$ is the total number of genres present in the rating vector, and $r2$ contains the final rating vector of the respective cluster. The dimension of the vectors used in the above algorithms is equal to the total number of genres.

VI. Movie Recommendation Algorithm

The Recommendation Algorithm recommends a list of items (movies) to the users in the cluster (or partition) with the idea that the recommended items will be liked by the users. In this work, we recommend movies to the user according to his most preferred or liked movie genres. As already mentioned in section I, the Recommendation Algorithm is applied separately to the clusters with the aim of reducing the complexity of the system. However, it may sometimes degrade the recommendation quality. Our aim is to optimize the algorithm such that it generates faster recommendation without compromising the recommendation quality much. We now present our Recommendation Algorithm in detail.

For recommending movies to users, first we build a test set of user preferences for movie genres along with normalized genre weights. Weights belong to the set [0,1}. We consider a maximum of four genres (randomly selected) per user from the available genres present in the movie database. Then we randomly select one of the users from this test set and represent the preferences of this naive user as a vector in k dimensions, where k is the total number of genres. Weights are assigned to the genres according to their order of preference. Our algorithm extracts the highest weight genre (most preferred) of this user from the preference vector, and then recommend movies according to this genre so that the recommended movies might interest him the most. Let us take an example of this scheme.

Suppose a user's preference vector is $PV = \{Adventure, Action, Drama, Comedy\}$ and the corresponding normalized genre weight $W = \{0.42, 0.33, 0.21, 0.01\}$. As *adventure* is the most preferred genre, the system recommends movies belonging only to that genre to a new user. To accomplish our recommendation task, we need to assign this target user to one of the clusters found by the clustering algorithm. This is done as follows. We extract the highest weight genre from the preference vector of this target user, and compare it with the corresponding genre of the Rating Vector (RV) of all the clusters. The cluster having the maximum vote for that genre is the winning cluster and the new user is assigned to this cluster. Assuming the fact that all the users of the winning cluster have seen all the movies of this genre, we recommend the new user a list of *top-10* movies of this genre highly rated by the users of that cluster. The recommendation process can be summarized as follows:

Algorithm Recommend_Movies
Step 1: Select one user for recommendation.
Step 2: Represent the user's genre preferences as a vector (preference vector).
Step 3: Assign weights to different genres of the vector.
Step 3: Extract the highest weight genre of this vector.
Step 4: Assign the user to one of the clusters according to his most preferred genre.
Step 5: Recommend *top-10* highly rated movies on the basis of the most preferred genre.

VII. Experiments and Results

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 Netflix 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 Metric Discussion

In this work, we use Mean Absolute Error (MAE) [2] to evaluate the prediction accuracy of the Recommendation Algorithm.

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

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

and

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

Table II. POSSIBLE RECOMMENDATIONS

	Customer Likes (rating = 4 or 5)	Customer Dislikes (rating = 1, 2 or 3)
Recommend	True positives	False positives
Do not recommend	False negatives	True negatives

Any prediction algorithm tries to minimize the MAE.

We have depicted the different combinations of recommendation that can be generated in a typical recommendation problem in Table II. Note that a customer likes an item if he has given a rating of 4 or 5 to that item (in a scale of 1 to 5), otherwise dislikes it, i.e., his rating is 1, 2 or 3. A recommendation is positive if the recommended rating coincides with the actual rating given by the customer.

In this work, we use *Precision* and *Recall* metrics to measure the quality of the *top-10* recommended list. *Precision* and *Recall* are widely used by RS to evaluate their recommendation quality. We formally define them below.

Precision: Precision measures the degree of accuracy of the recommendations produced by the algorithm. In our system, Precision measures what fraction of the recommended items are liked by the customers.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall: The Recall metric is also known as the hit rate, which is widely used for evaluating *top-K* Recommender Systems. In our Recommender System, Recall measures what fraction of the items liked by the customers, has been recommended by the algorithm.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

B. Experimentation with Clustering Algorithm

Our work partitions the users in the dataset according to their movie genres. As already discussed in section IV, we represent the preferences of each user as a vector (preference vector) in k dimensions, where k is total number of genres. The values of these dimensions are the normalized weights of the corresponding genre. We have then implemented the DBSCAN clustering algorithm to cluster the users. Table III depicts the results of the clustering algorithm for various values of ε-neighborhood (*eps*) and *MinPts*. Here *MinPts* is the minimum number of users required to form a cluster. If the distance between two user's preference vector is less than or equal to *eps*, then the users belong to the same cluster.

Table III. RESULTS OF CLUSTERING

eps	MinPts	Number of clusters
0.5	100	25
0.75	150	17
1.0	200	8

C. Experimentation with Voting Algorithm

In this work, we implement the voting algorithm in the clusters found by the clustering algorithm. The voting process produces the Rating Vector (RV) of the cluster, which contains the vote received by the different genres in that cluster. Sample results of the voting algorithm applied to the different clusters are presented through the Figures 1, 2 and 3. In the Rating Vector of cluster 1 (Fig. 1), we can see that the Drama genre is the winning genre in that cluster. It signifies that the users in cluster 1 have preferred Drama movies over all other genres. Similarly in cluster 2 (Fig. 2), Horror is the winning genre and in cluster 3 (Fig. 3) Comedy movies outperform the movies of all other genres.

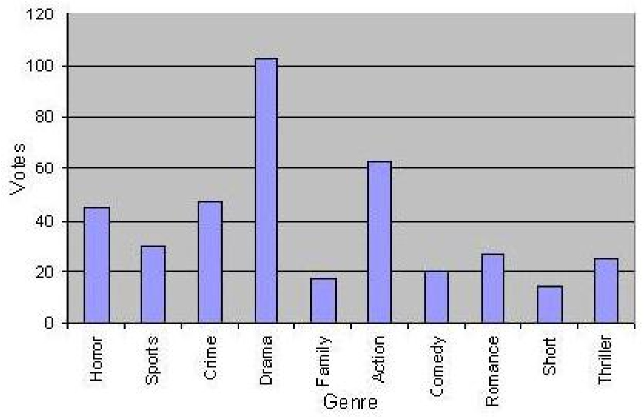

Figure 1. Sample Rating Vector of 1st Cluster

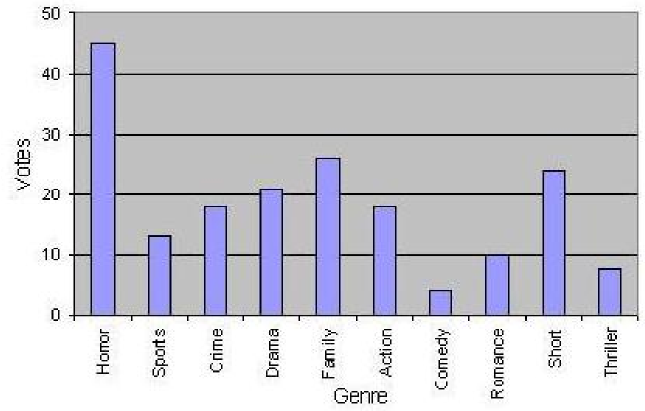

Figure 2. Sample Rating Vector of 2nd Cluster

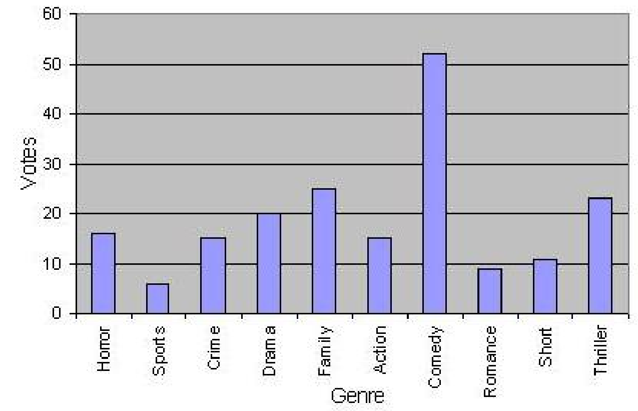

Figure 3. Sample Rating Vector of 3rd Cluster

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. We recommend movies to the user according to choices. For recommendation purpose, we select users from the test set (described in section VI) and recommend movies according to their most preferred choice. Suppose we have a user with Preference Vector $PV = \{Horror, Family, Sports, Thriller\}$ and corresponding normalized genre weight $W = \{0.42, 0.26, 0.25, 0.06\}$. Now to recommend movies based on his choices, we need to assign him to one of the clusters. Since, here Horror is the most preferred genre (normalized genre weight = 0.42), the user is assigned to cluster 2 (Fig. 2) as it has the most number of votes for the Horror genre among all the clusters. Then the *top-10* movies highly rated by the users in that cluster are recommended to this user.

We report and compare the results of the Recommendation Algorithm using different evaluation metrics in Table IV. In this Table, the column $Time$ reports the overall running time of our Recommendation Algorithm. Here base represents the entire dataset without decomposition. We compare the overall performance in the clusters formed by the DBSCAN algorithm with the base performance. Note that, we present $Precision@10$ and $Recall@10$ 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 2 GB RAM.

Table IV. PERFORMANCE COMPARISONS ON NETFLIX DATASET IN TERMS OF MAE, PRECISION, RECALL AND TIME

No. of clusters	MAE	Precision@10	Recall@10	Time (minutes)
1 (base)	0.483	0.846	0.782	843.56
25	0.482	**0.857**	0.763	**170.54**
17	**0.441**	0.833	**0.853**	**210.65**
8	0.498	**0.878**	**0.848**	**314.32**

From Table IV, it is clear that our algorithm is working fine with high precision and recall values as well as lower MAE values. We can also notice 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 843.56 minutes. However, when it is partitioned into 25 clusters, we require a overall runtime of 170.54 minutes for recommending all the users of the 25 different clusters which is significantly less (by about 80%) than the base. Similarly for the case of 17 and 8 clusters, runtime reduces by about 75% and 63% respectively.

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

VIII. CONCLUSION

In this paper, we have presented a scalable clustering based Recommender System. We have incorporated voting based movie selection technique that uses stored user preferences for genre. Our proposed approach deals with the *Scalability* problem of the recommendation task by applying the recommendation algorithm separately to the clusters. Our DBSCAN based clustering algorithm successfully identified the clusters as well as noise points. We have discarded the noise points (users not belonging to any cluster) so that they do not contribute to the voting procedure. Incorporating other movie dimensions such as Actors, Actresses, etc. for the recommendation purpose will be the focus of our future work. We also have a plan to implement other data clustering and voting algorithms with the aim of optimizing the clustering technique and hence the Recommendation Algorithm.

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*, pp. 734–749, 2005.

[2] B. Bhasker and K. Srikumar, *Recommender Systems in e-Commerce*. McGraw-Hill Education, 2010.

[3] J. Das, S. Majumder, and P. Gupta, "Voronoi based location aware collaborative filtering," in *Proceedings of the 3rd IEEE Conference on Emerging Trends and Applications in Computer Science (NCETACS)*, 2012, pp. 179–183.

[4] E. Ephrati and J. Rosenschein, "Multi-agent planning as a dynamic search for social consensus," in *Proceedings of the 13th International Joint Conference on Artificial Intelligent*, 1993, pp. 423–429.

[5] 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.

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

[7] 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.

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

[9] J. Lang, "Logical preference representation and combinatorial vote," *Annals of Mathematics and Artificial Intelligence*, vol. 42, pp. 37–71, 2004.

[10] R. Mukherjee, P. Dutta, and S. Sen, "Movies2go-a new approach to online movie recommendation," in *Proceedings of the IJCAI Workshop on Intelligent Techniques for Web Personalization*, 2001.

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

[12] 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.

[13] J. Schafer, J. Konstan, and J. Riedl, "Recommender systems in e-commerce," in *Proceedings of the 1st ACM conference on Electronic Commerce (EC-99)*, 1999, pp. 158–166.

[14] P. Straffin, *Topics in the theory of voting*. The UMAP expository monograph series, Birkhauser, Boston, MA, 1980.

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

[16] C. Webber, S. Pesty, and N. Balacheff, "A multi-agent and emergent approach to learner modelling," in *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, 2002, pp. 98–102.

[17] 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.