CSCE 625 : Final Project Portfolio
Movie Rating Estimation
Based on K-Means and Biclustering Algorithms

I. Abstract

Rating prediction and recommendation, that help match user preferences to items or products, are increasingly being used by online retail websites. This project focuses on analyzing the performance of biclustering and mini-batch k-means algorithms as underlying algorithms for a movie rating prediction system. Biclustering is a technique that allows simultaneous grouping of both user and products and works very well for consumer data that is represented as a bipartite graph. K-Means algorithm is a well-known clustering algorithm, and its variant Mini-Batch K-means works well for large data. Both these algorithms give a high RMSE accuracy on the MovieLens 100K dataset, and results also demonstrate that K-means run in batches can be used interchangeably with biclustering for large graph-based data.

II. Introduction

With the advent of online consumer services in areas of music, books, clothing, electronics and others, there has been a growing demand for recommendation and user prediction systems. Recommendation and rating prediction systems are useful in two ways : they provide personalized recommendations to users which increases consumer satisfaction and they help the online service predict user needs as per their past history and increase sales. Currently, these artificial intelligence systems are an integral part of retail applications such as Amazon, Netflix, YouTube and Google News. A rating estimation system works by analyzing the characteristics of items and users based on their past "shopping characteristics". This historical data is then used to make a prediction or judgement on what rating will be given to an item not yet evaluated by a user. This prediction is helpful, as higher the predicted rating implies more likely that the user will purchase or view the item.

Movie rating estimation and recommendation is an interesting problem area in this domain. An example of it is the "The Netflix Prize". As mentioned on its website, it seeks to substantially improve the accuracy of predictions about how much someone is going to enjoy a movie based on their movie preferences [1]. NetFlix Prize seeks to predict the ratings on the test data such that the RMSE (root mean squared error) improves 10 percent over Netflix's current system.Websites such as NetFlix and YouTube allow users to rate movies, TV shows and videos [13].A good rating prediction system for movie websites is helpful in providing a more enriching experience to users and predict future recommendations and demands.

For my final project portfolio, I worked on the MovieLens 100k dataset. This dataset captures information about movie ratings by users from the MovieLens website. The project involved partitioning the dataset into training and test sets. With three such disjoint sets of training and test data, similar users in the training dataset were clustered together using two algorithms : Mini-Batch k-means Clustering and Biclustering(Spectral Co-Clustering). With Mini-Batch K-Means, movie ratings were estimated for users and movies in the test dataset using average and weighted average movie rating of similar users. For Biclustering, along with the average and weighted average methods, I also identified similar movie biclusters and made rating predictions based on this and user biclusters. Root mean square error(RMSE) method and a simple heuristic measurement were used to measure and compare the accuracy of these two algorithms on cluster sizes of 10 and 50.

III. Background

Clustering algorithms attempt to find groups of similar objects from a given set of objects. These algorithms are well applicable to the the field of rating prediction as we need to find similarity between users or products as the first task in prediction. Generally, the n objects to be clustered are represented as a p-dimensional feature vector[14].Hence it becomes possible to represent all objects using a matrix which has a size number of objects * number of features. The two algorithms that I have used are described in detail below.

A. Mini Batch K-Means

K-Means is one of the most popular clustering algorithms. Given n objects, it tries to find k clusters such that each object belongs to a cluster. Each of these objects is assigned to that cluster such that mean value of that cluster is closest to the value of the object. Also, in K-Means we try to minimize the sum of squares within each cluster. Hence the objective function for k-means is specified as [18] :

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

The essential steps of the algorithm are [16] :

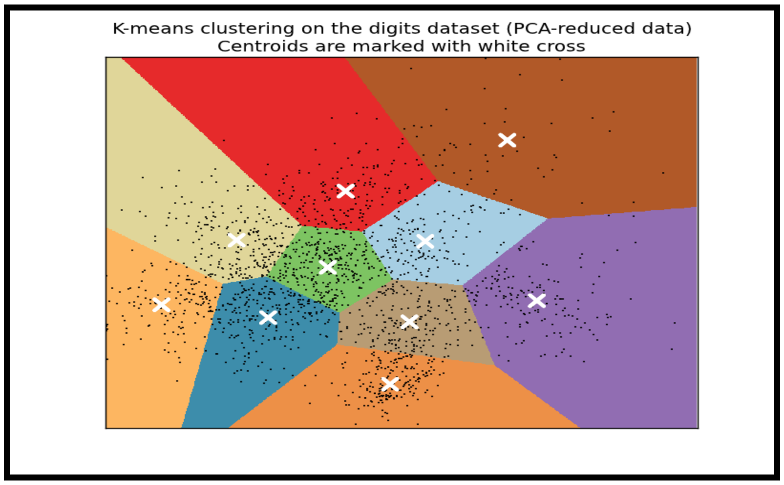
a. Pick up k points/samples from the objects given. These represent initial centroids.

b. From the remaining objects, assign each object to that cluster whose centroid is closest to the value of object. For this distance/closeness calculation, simple Euclidean distance is used.

c. After the assignments in step b, recompute the means of the k clusters(centroids) again.

d. Perform steps b and c again until centroid values converge (i.e. no longer change). This helps in achieving the objective function of k-means.

The following image taken from [17] depicts how the plot of K-Means looks like after clustering. The different colors represent different clusters. The × symbols represents the centroids or means of the clusters.



A major limitation of K-means is that its performance is poor with large data (especially the data that I used in this project, which has ratings from 942 users) and with a large number of features for an object.

So we use Mini Batch KMeans that divides data into mini-batches. Each mini-batch consists of few samples, and K-means is iteratively run over these subsets. It involves following steps :
a. First pick up k centroids defining the clusters. Then we pick up b samples, these form one mini-batch. We assign these samples to the nearest centroid.
b. We recalculate the centroids after the assignment in the previous step. This is done by taking the average on this sample and previous samples assigned to this centroid.
c. Repeat steps a and b for other mini-batches.

These steps are run for some iterations until convergence. The performance is improved greatly over K-Means with very slight deterioration in accuracy. It gives a good tradeoff of computation time versus accuracy.

B. Biclustering (Spectral Co-clustering)
Biclustering is a type of clustering algorithm that simultaneously clusters both rows and columns of a matrix[12]. A very good example is suppose I wish to invite guests to our house. This example is inspired from [14].
Suppose each of these guests has different food tastes and preferences such as guests prefer Italian, French, Mexican or Asian cusines and even within these they have a liking for a particular dish or dessert. Also, suppose not all guests can fit in 1 room, so we cannot accommodate all dishes in 1 single room. Suppose there are several rooms in the house. What I

would like to do is distribute guests and food items in a manner such that guests are in the room which has their favourite food.
What we are essentially trying here is to maximize a objective function such that biclusters of equal size (consisting of guests and dishes) are formed and there is no overlapping among biclusters.Also, guests who have a strong preference for some dishes fall in biclusters with high values).

The ultimate result is that though not all guests will be assigned a room with their favourite food item, but on average most guests will get the room with the food of their liking.

If we consider the guest*dish matrix, this is how it can be represented visually.

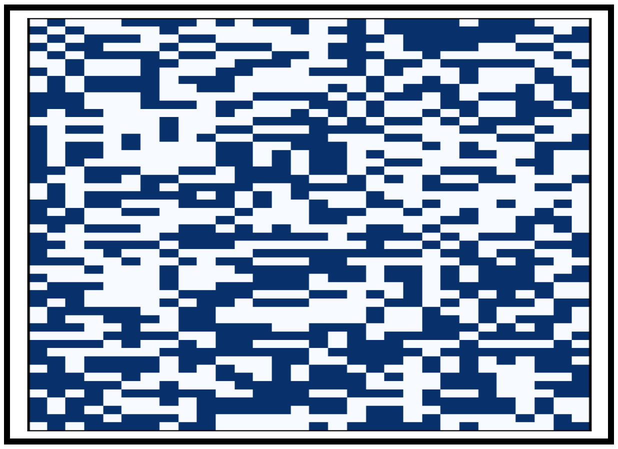


Fig 1.

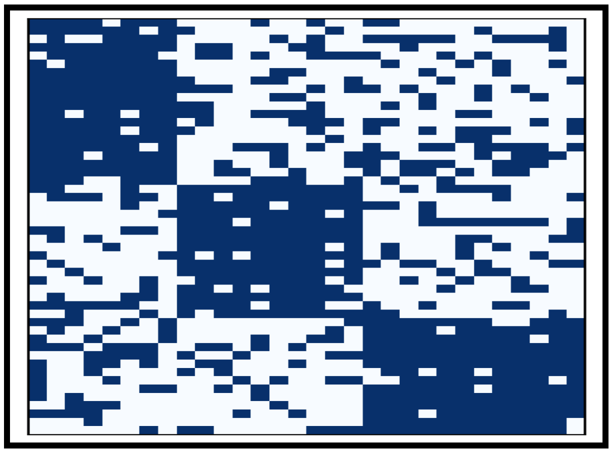


Fig 2.

Given the initial guest*food matrix on the left hand side, biclustering produces the matrix on the right hand side with biclusters with high values along diagonal, quite distinctly observed.

Another important aspect of biclustering is that it treats the input matrix as a bipartite graph. That is, the rows and columns can be considered to be two different set of vertices and all edges exists only between a row and a column. Hence a value in the matrix corresponds to an edge between the corresponding row and a column. Such a bipartite graph is useful for clustering products according to customers or customers according to products. Biclustering tries to find a normalized cut of the graph to find heavy subgraphs[15]. The algorithm attempts to shuffle the input matrix so that biclusters are contiguous.

Different algorithms exists for biclsutering. They essentially differ in what they are looking for in the matrix : strikingly high or low values, coherent values on row and columns or constant values in row and column of matrix(after normalization).

For my particular implementation of biclustering, I considered using Spectral Co-clusetring, which looks for rows and columns with unusually high values in rows and columns (consider rating of 4 or 5).

The underlying formulation of Spectral Co-clustering is as follows [15] :
1. It forms non-overlapping biclusters
2. Finds biclusters with values higher than corresponding rows and columns.
3. It first attempts to find the normalized cut of bipartite graph. For this the Laplacian of the input matrix should be used. But spectral co-clustering works with the input user*product matrix directly.
4. If R is a diagonal matrix with entry i = $\sum_j A_{ij}$ and C is the diagonal matrix with entry j= $\sum_i A_{ij}$, a preprocessed matrix A is formed as :

$$A_n = R^{-1/2} A C^{-1/2}$$

5. A new matrix Z is then formed such that :

$$Z = \begin{bmatrix} R^{-1/2} U \\ \\ C^{-1/2} V \end{bmatrix}$$

5. We then cluster rows of Z using K-means. This gives us biclusters along the diagonal of the matrix.

IV. Related Work

Previous work in the area of movie-rating prediction system involves analyzing several machine learning algorithms – baseline predictor, k-nearest neighbour, Stochastic Gradient Descent, Singular Value Decomposition and its variants, integrated model and NMTF on the basis of their accuracy of rating prediction for unrated movies by users[5]. Another work analyzes how k-means clustering fares in movie recommendations for users. Accuracy is measured by checking the proportion of movies that were recommended, which were also present in the test data for the user[2]. [4] develops a unique method that overcomes the shortcomings of collaborative-filtering, a widely used technique used in rating prediction and recommendation. It uses bicluster networks and also utilises nearest biclusters for rating prediction and recommendation.

V. Dataset and Pre-processing

The MovieLens website asks users to rate movies. This helps in creation of a custom taste profile, which it then uses to recommends movies[8]. MovieLens 100K dataset [7] consists of 100,000 ratings, consisting of 942 users and 1680 movies. These ratings are in the range 1-5, with 5 as the highest rating. Also, each user has rated at least 20 movies, which makes this dataset rich. I made use of u1.base/test , u2.base/test and u3.base/test as the training and test datasets respectively, which are disjoint sets. For weighted average runs, I split test sets into 20 sets of 1000 for faster computations. We have length of training dataset = 80000 and length of test dataset=20000.

Pre-processing is performed on the training dataset as follows : Each line of the training dataset, say u1.base, is read and input into a user*movie matrix. The input data is in the following format:

userid movieid rating timestamp

The output matrix is a numpy matrix implemented in python.The value for a particular user and movie index in the matrix is the rating of that movie by the user. Ratings are normalized for each user/row by the following formula given in [9]:

$$Normalized\,(e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}}$$

e_i in our case is the actual rating, Emin is 1, Emax is 5. The above obtained normalized rating was multiplied by 5 to get a rating between 1 and 5. Thus a user's highest rating for is scaled to 5 and lowest to 1. If a movie has not been rated by a user, the corresponding value in the matrix is 0. This normalization is applied to improve the accuracy of results and take into account the rating pattern of different users (some users will rate a movie they really liked as 3 and a movie they did not like as 1).

VI. Implementation

The code for the project is implemented in python and python scikit. The corresponding code files are :
biclustering.py
biclustering_weighted_average.py
biclustering_with_movie.py
kmeans.py
kmeans_weighted_average.py

and these can be run as :

```
python <codefilename>
```

[20] helped me implement the code for mini batch k-means.

A. Mini-Batch K-Means

The pre-processed matrix from section V was given as input to the Mini-Batch K-Means function implemented in scikit[10]. The package sklearn.cluster.MiniBatchKMeans contains the implementation of this algorithm. The MiniBatchKMeans function initializes the algorithm and takes parameters such as number of clusters, batch size, initial size etcetera. I set the number of clusters as 10 and 50 to see the effect on rating prediction.
The fit function was called on the user*movie matrix. It computes the centroids by splitting the matrix into batches of 100.
The predict function invoked on the matrix data computes the closest centres for each user. It returns index of the cluster for each user in the user*movie matrix.

B. Biclustering (Spectral Co-Clustering)

Similar to above, the pre-processed matrix from section V was given as input to the SpectralCoclustering function implemented in scikit[15,19]. The package sklearn.cluster.bicluster.SpectralCoclustering contains the implementation of this algorithm. The SpectralCoclustering function initializes the algorithm and takes parameters such as number of clusters, batch size, initial size etcetera. Again I set the number of clusters as 10 and 50 to see the effect on rating prediction.
The fit function was called on the user*movie matrix. It computes the biclusters and returns the bicluster index for each user from row_labels_ method and bicluster index for each movie from column_labels_ method.

C. Rating Prediction

a. Simple Average of Similar Users Who Have Rated for The Same Movie

For both Mini-Batch KMeans and Biclustering, for predicting the rating of a movie by a particular user in the test dataset, we first find out other users in the same cluster/bicluster as the test user. For this we first get the cluster/bicluster index obtained as output of the fit function. From this index, we obtain all other users with the same cluster/bicluster index. For each of these users in the same cluster/bicluster, we see if they have rated the test movie, and compute average of the rating of all such users for that movie.
The corresponding code files are kmeans.py and biclustering.py

b. Weighted Average

For both Mini-Batch KMeans and Biclustering, instead of finding a simple average as the method above, we do the following. For predicting the rating of a movie by a particular user in the test dataset, we first find out other users in the same cluster/bicluster as the test user. For this we first get the cluster/bicluster index obtained as output of the fit function. From this index, we obtain all other users with the same cluster/bicluster index. For each of these users in the same cluster/bicluster, we see their distance from the test user. The distance is computed as the square root of the sum of squares of difference in rating by the test user and user in same cluster/bicluster. Movies rating values of 0 are not considered for this calculation . The inverse of this distance gives the weight to be assigned to the rating of such a user. We normalize this rating by the sum of weights over all the users in the same cluster/bicluster. [11] gives an idea of how to calculate weighted average.

$$\bar{x} = \frac{w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}{w_1 + w_2 + \cdots + w_n}.$$

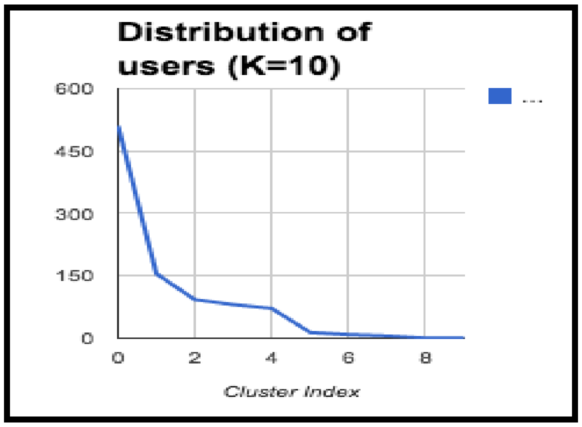
where w = 1/distance.
The corresponding code files are biclustering_weighted_average.py and kmeans_weighted_average.py.

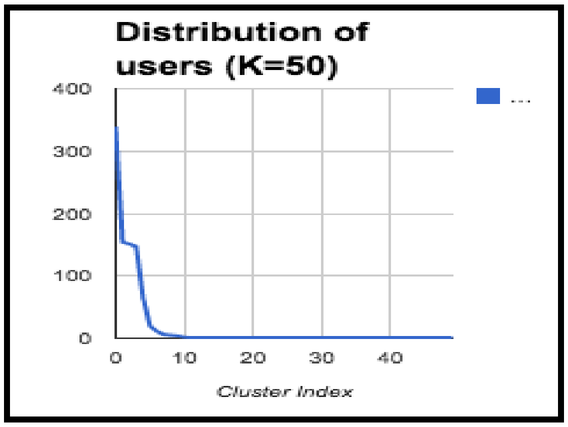
c. Rating of Similar Movies By Same Test User
Since Biclustering also clusters similar movies together, we make use of the following heuristic for predicting the movie rating. From the movie in the test set, we find out the bicluster index it belongs to. From this index, we find out other movies in same bicluster from column_labels_ method. We then see the rating given by the test user to these movies and consider the average of these ratings (ratings of 0 are ignored). We then take the mean of this rating and rating obtained from a., and compute the predicted rating for the test movie by the test user.
The corresponding code file is biclustering_with_movie.py.

VII. Results and Analysis

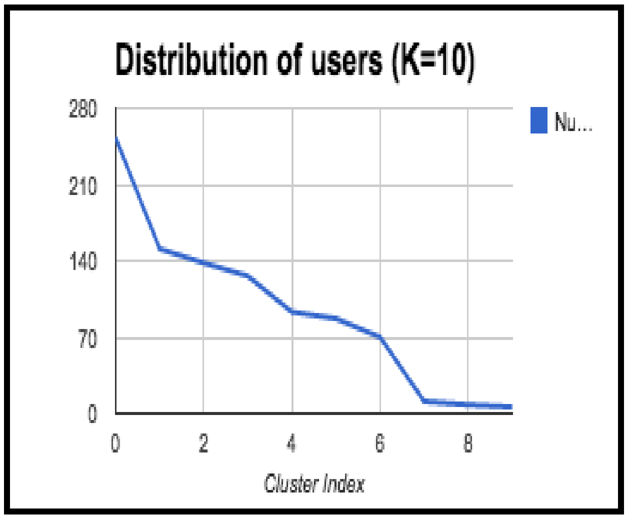
A. Power law Curve number of assigned users to each cluster.

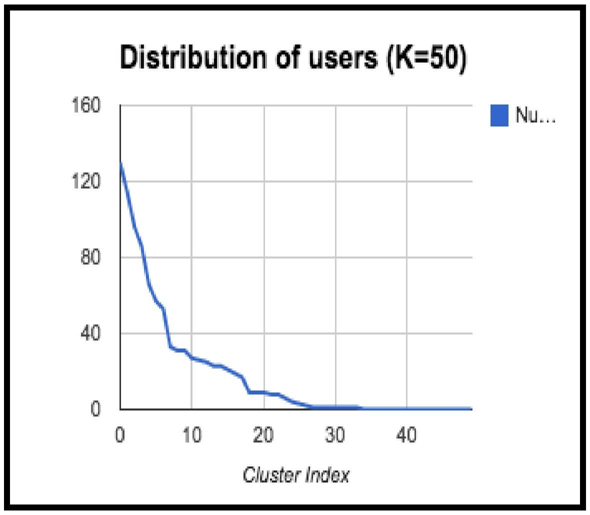
Running Mini-Batch KMeans with k=10 and k=50, produces a power law curve when we compare number of assigned users in each cluster. The maximum users in a cluster are 511 and 339 respectively. The graphs seem to obey the 80-20 rule. That is most number of users(80%) lie in few clusters(20%).





Running Biclustering with k=10 and k=50, produces a power law curve when we compare number of assigned users in each cluster. The maximum users in a cluster are 511 and 339 respectively.The maximum users in a cluster are 253 and 130 respectively. The graphs obeys the 80-20 rule.



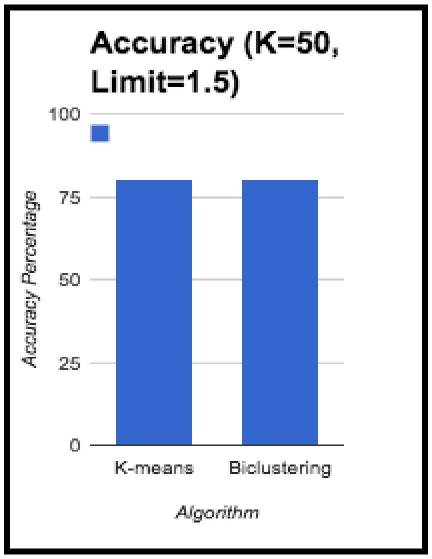


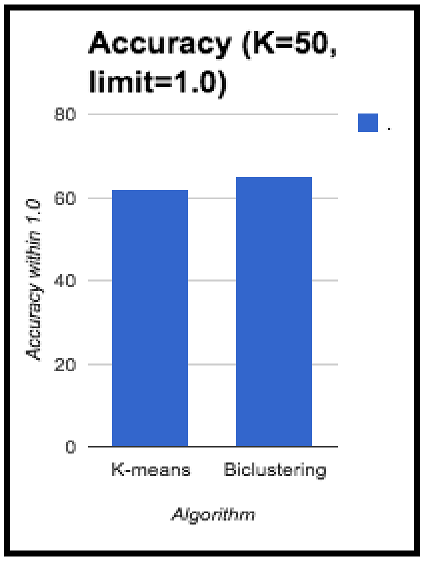
B. Accuracy in terms of difference of 1 or 1.5 rating

After computing the predicted rating in Section VI., we compare against the actual rating present in test dataset. By allowing a error of 1.5 and 1 in predicted and actual rating and K=50, we see both algorithms perform well in rating prediction when a difference of 1.5 is allowed. With 1.0, accuracy deteriorates for both. Deterioration in accuracy is more with k-means.

Algorithm(K=50)	Accuracy(upto 1.5)	Accuracy(upto 1.0)
K-means	80.355	61.95

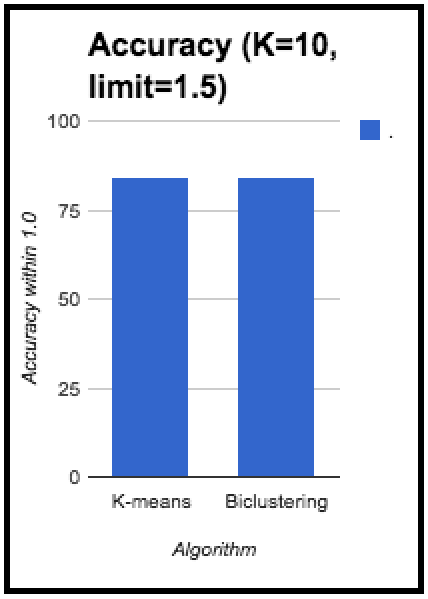
Biclustering	80.17	65.09

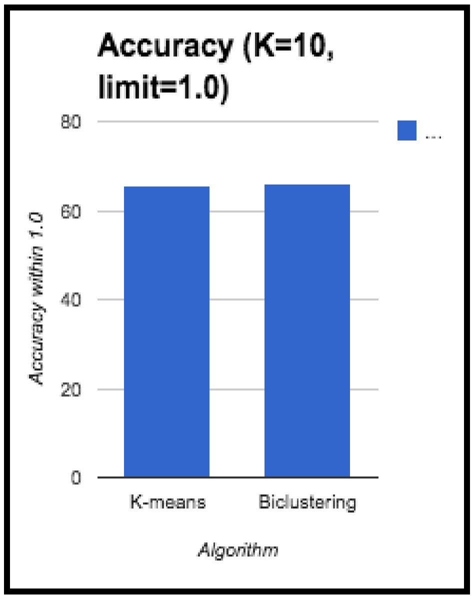




With k=10, there is improvement in accuracy of both algorithms. This means that k=10 is a good estimate of the cluster number, and increasing the number of clusters might adversely affect accuracy.

Algorithm(K=10)	Accuracy(upto 1.5)	Accuracy(upto 1.0)
K-means	84.245	65.735
Biclustering	84.51	66.25





C. Root Mean Square Error(RMSE)
Another way that I measured accuracy of the two algorithms is by computing the Root Mean Square Error which is a standard way of accuracy measurement of prediction systems. The formula I used for RMSE is :

$$\text{RMSE} = \sqrt{\sum_{test\ data\ length} (predicted\ rating\ -\ actual\ rating)^2 / test\ data\ length}$$

Here test data length = 20000.
The RMSE values found are as follows :

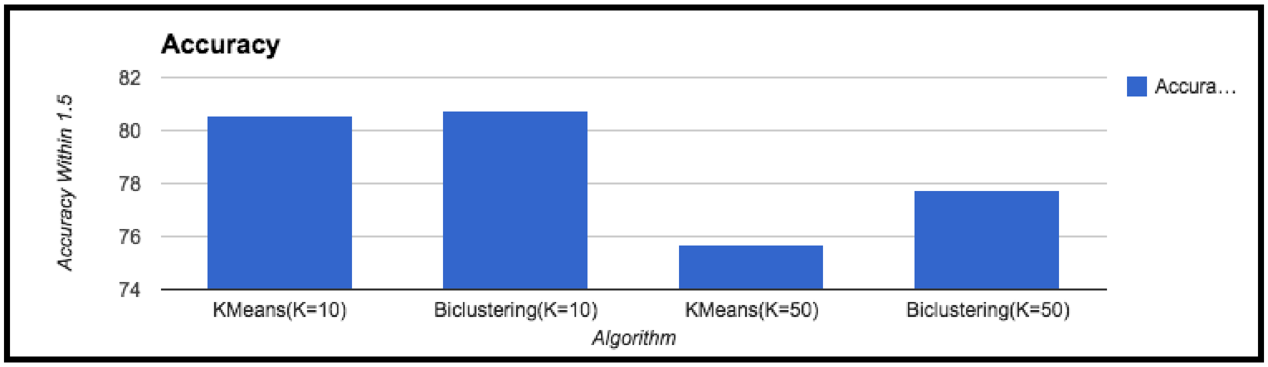
Algorithm	RMSE
K-Means(K=10)	1.061862152
Biclustering(K=10)	1.032437857
K-Means(K=50)	1.012032481
Biclustering(K=50)	1.029432892

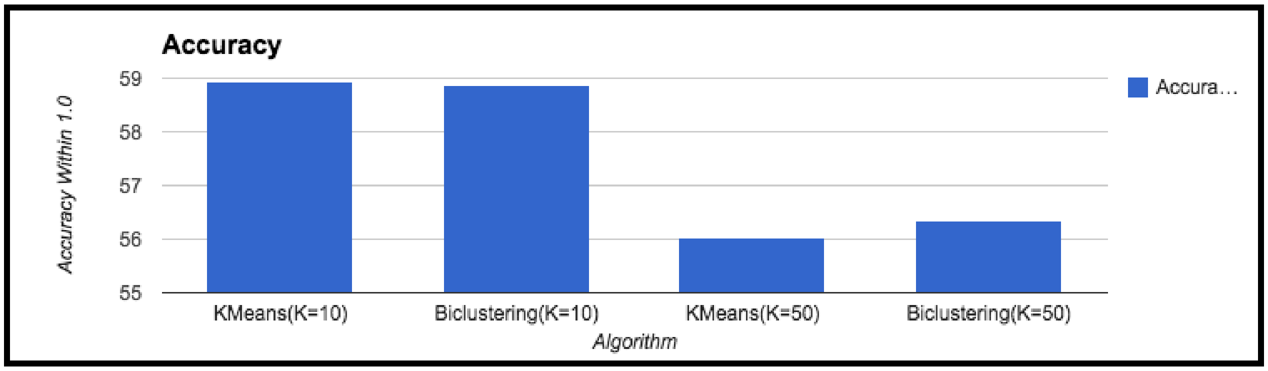
The RMSE is improved with k=50 than k=10 for both the algorithms, hence having more number of clusters gives more RMSE accuracy. Mini-batch K-Means and Biclustering perform equally well in this aspect, with slight variations.

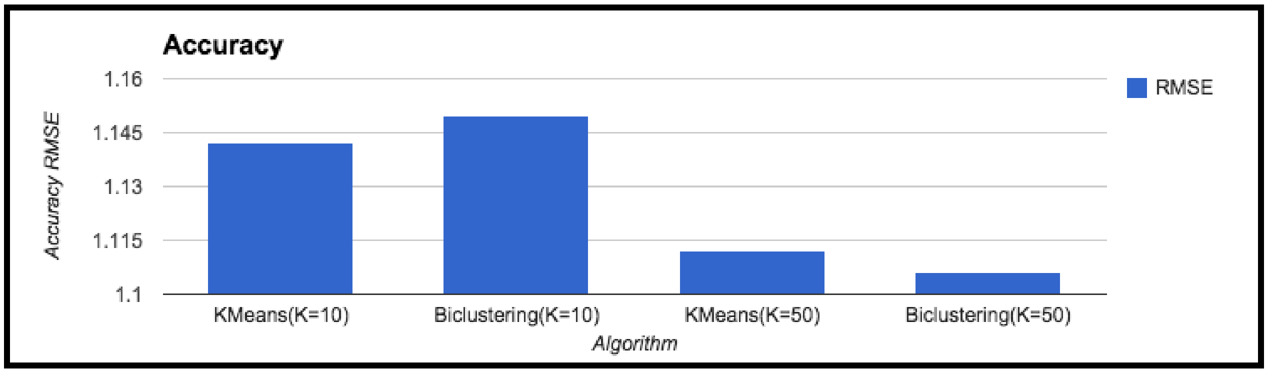
C. Weighted Average Scheme

As described in the Implementation section, Rating Prediction part b, we used weighted average for getting the rating. These are the results :

Algorithm	Accuracy(upto 1.5)	Accuracy(upto 1.0)	RMSE
KMeans(K=10)	80.58	58.935	1.142236256
Biclustering(K=10)	80.76	58.89	1.149653026
KMeans(K=50)	75.7	56.025	1.112096084
Biclustering(K=50)	77.755	56.34	1.105984043







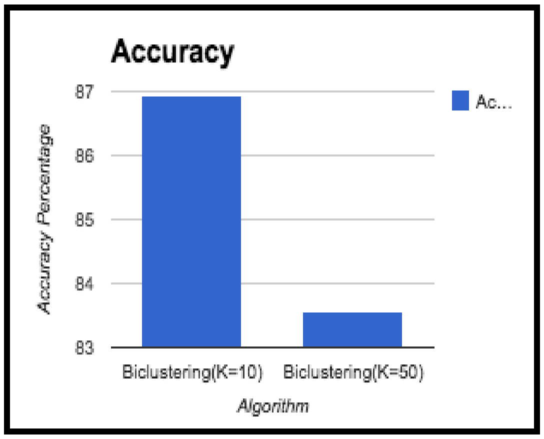
Similar to simple average, while k=10 performs better with simple heuristics of difference in 1.0 1.5, k=50 gives better RMSE accuracy(Lower error). Mini-batch K-Means and Biclustering perform good and with nearly same accuracy.

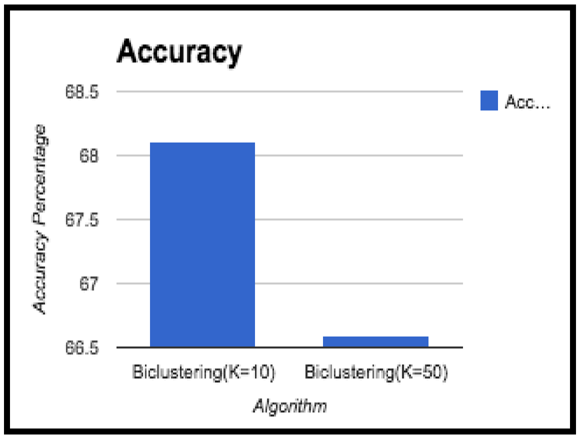
However there are no particular improvements in using the weighted average scheme over simple average. In fact it performs more poorly. This is probably because each user has 1680 features, because of which distance and hence weight calculation deteriorates.

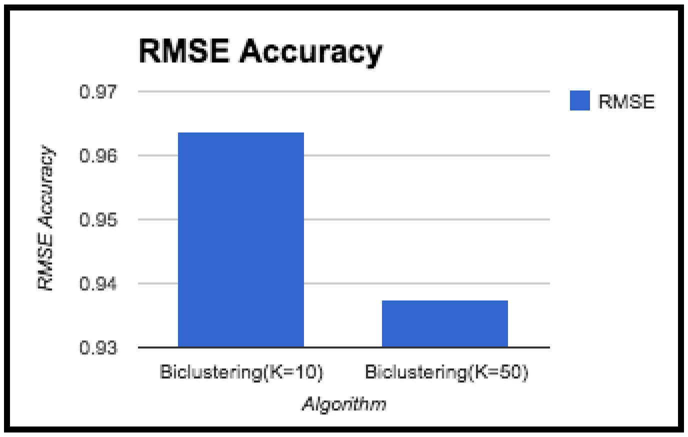
D. Incorporating movie clusters for calculation

Incorporating movie clusters and ratings of similar movies by same user in Biclustering(method c of Rating Prediction in Implementation section) definitely boosts performance. This is because movie clusters give more information about the movie in the test data and improves prediction. For k=10, there is an improvement of 2.5% and for k=50, there is an improvement of 3%.
In terms of RMSE, error is reduced by 0.13, which is a significant reduction(12 %).

Algorithm(K=10)	Accuracy(upto 1.5)	Accuracy(upto 1.0)	RMSE
Biclustering(K=10)	86.925	68.115	0.9636454835
Biclustering(K=50)	83.55	66.595	0.9375826978







VIII. Conclusion

Overall, as part of this project, I was successfully able to implement movie rating prediction with biclustering and mini-batch k-means algorithm. The results demonstrate that both algorithms show good accuracy with RMSE in a simple weighted scheme. Also, though biclustering performs slightly better when we allow errors of 1.5 or 1.0 in predicted rating, RMSE values are nearly same for both algorithms. This means that apart from being good algorithms for rating prediction, k-means run in batches can be plugged in places where large data is attempted to be biclustered and gives nearly accuracy with reduced computational time compared to k-means.

IX. References

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

[2] http://cs229.stanford.edu/proj2013/Bystrom-MovieRecommendationsFromUserRatings.pdf

[3] http://www.rowequality.com/sites/default/files/Market_Segmentation_Using_Kmeans.pdf

[4] http://www.iaeng.org/publication/IMECS2013/IMECS2013_pp85-88.pdf

[5]http://cs229.stanford.edu/proj2012/BaoXia-MovieRatingEstimationAndRecommendation_FinalWriteup.pdf

[6]http://grouplens.org/datasets/movielens/

[7] http://files.grouplens.org/datasets/movielens/ml-100k-README.txt

[8] https://movielens.org/

[9] http://stn.spotfire.com/spotfire_client_help/norm/norm_scale_between_0_and_1.htm

[10]http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html

[11] http://en.wikipedia.org/wiki/Weighted_arithmetic_mean

[12]http://en.wikipedia.org/wiki/Biclustering

[13] http://www.cs.utexas.edu/users/inderjit/courses/dm2007/projects.htm

[14] http://www.kemaleren.com/an-introduction-to-biclustering.html

[15] http://scikit-learn.org/stable/modules/biclustering.html

[16] http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html

[17] http://scikit-learn.org/0.11/_images/plot_kmeans_digits_1.png

[18] http://en.wikipedia.org/wiki/K-means_clustering

[19]http://scikit-learn.org/stable/modules/generated/sklearn.cluster.bicluster.SpectralCoclustering.html#sklearn.cluster.bicluster.SpectralCoclustering

[20]http://codereview.stackexchange.com/questions/52029/k-means-clustering-in-python