DATA MINING

Assignment I

C Rajmohan 040400104113110164

Question 1

---

*The solution to this problem is submitted as **hardcopy**.*

The solution is applied in movielens dataset and results are given here. This dataset contains 100,000 ratings (1-5) from 943 users on 1682 movies. Each user has rated at least 20 movies. There are many entries missing in this ratings table. The task is to predict missing entries in the ratings table (matrix).

Non-negative matrix factorization technique is used to factorize the given ratings Matrix X has MH^T. Here M is d × k matrix and H is n × k matrix given X is a d × n matrix.

This factorization captures the important features of the data. That is the 'k' value is the number of features that we want to extract in data. Matrix M represents relationship between users and these features and H represents relationship between movies and features. Our main goal is to approximate matrix X into M and H in such a way that the mean squared error is less. **Gradient descent** optimization technique which is an **iterative** technique is used to find minimum value of this mean squared error function. Initializations of parameters are given in the .m file attached with this report. Updates rules for gradient descent method are derived in the write-up submitted with this report.

The algorithm implementation is attached with this report. Only the results are discussed here.

Algorithm is first trained by using training dataset 'u1.data' which has 80% split of 'u.data'. Training here means tuning the values of

Rank of approximation (K),
Regularization parameter (lambda),
Step size of gradient descent method (alpha) and
Number of iterations required

so that we get good approximation i.e. Mean squared error is minimized.

Once the above parameters are assigned a stable value, run the algorithm on test data 'u1.test' which has only very few filled entries. It contains 20% of 'u.data'. The results are as follows.

Remember that the initial values chosen for M and H matrix are random. Hence run the algorithm for few times before deciding on a parameter value.

Input dataset: u1.test

Output: A complete approximated ratings matrix with all missing entries filled in by prediction.

How to run:

Add 'setup_data.m' file and **'predict_ratings.m'** files to the matlab workspace. Run these two matlab files one after the other in order.

"setup_data.m" prepares movielens dataset into required format in matlab.

"predict_ratings.m" file contatins code to perform NMF and error calculations.

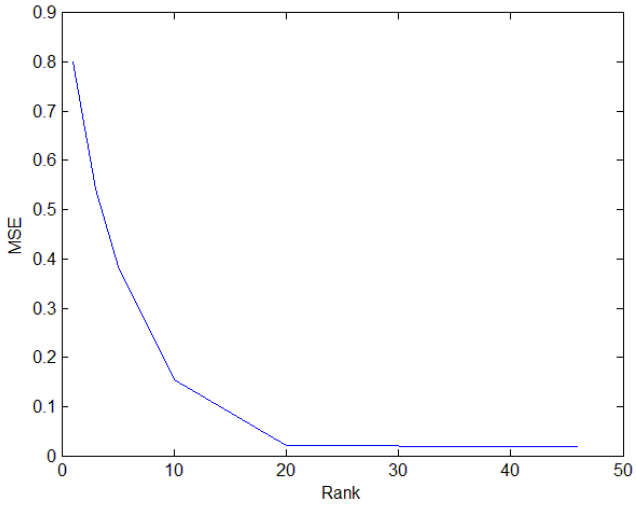
Analysis Results:

The Mean Squared Error(MSE) is given by,

$$E_k = \frac{1}{|\Omega_k^c|} \sum_{(i,j)\in\Omega_k^c} \left[X_{ij} - \psi\left(m_i^T h_j\right) \right]^2$$

Rank vs. Mean Squared Error:

As rank of approximation increases, mean squared error decreases. The following 2D diagram depicts that pictorially.

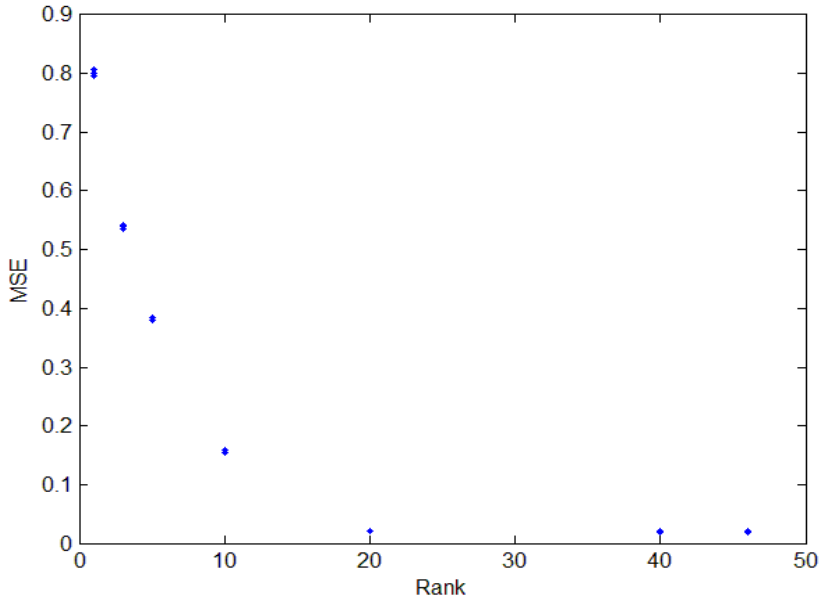


(No. of Iterations = 1000, alpha(eeta) = 0.003, lambda = 0.01)

The following 2D plot shows 3 iterations for each rank of approximation. As error variation is very less, its minute in details in the picture.

Results of 3 independent run for each rank ‘K’:

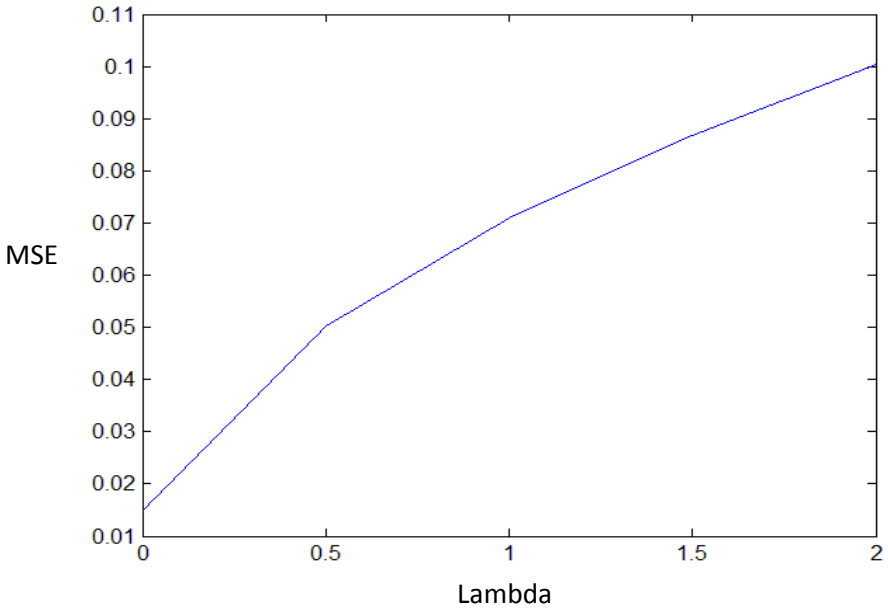
Iter\K	1	3	5	10	20	40	46
1	0.8006	0.5425	0.3850	0.1550	0.0200	0.0196	0.0192
2	0.7956	0.5364	0.3795	0.1513	0.0207	0.0200	0.0199
3	0.8075	0.5395	0.3844	0.1526	0.0216	0.0194	0.0197



(No. of Iterations = 1000, alpha(eeta) = 0.003, lambda = 0.01)

Parameter Lambda vs. MSE:

As we can see in below figure, as lambda increases, mean squared error increases.



(No. of Iterations = 1000, Rank = 40, alpha = 0.003)

Predict.m

```
%  R = Ratings Matrix containing users as rows and movies as columns.
%  Each entry in this matrix is the rating given by the user for the
%  particular movie. If an entry is zero, it means the user has not
%  rated the movie.

R = RatingsTable;

% Minimum and maximum ratings
a = 1;
b = 5;

% [M,N] => Dimensions of Ratings matrix

[m,n] = size(R);

% Rank K Approximation (K ~ Features)
K = 46;

% Decompose matrix R into M and H such that R = MH^t

% Assume random values initially for matrices P and Q
M = rand(m,K);
H = rand(n,K);

% Number of Iterations
max_iter=1000;

% Learning rate of gradient descent optimization technique
alpha=0.003;

% Regularization factor
lambda=0.01;
omega = 0;
% for each non-zero entry in ratings matrix do
for i = 1:m
      for j = 1:n
            if R(i,j) > 0
                omega = omega + 1; % Known entries
            end
      end
end

% transpose the matrix
H = transpose(H);

for iter = 1:max_iter
  for i = 1:m                    % for each entry in ratings matrix do
      for j = 1:n
            if R(i,j) > 0
                % calculate the error between estimated rating and real
                % rating
                eij = R(i,j) -(M(i,:) * H(:,j)); % error term

                for k = 1: K
                    % Update rules
```

```
                    M(i,k) = M(i,k) + alpha * (2 * eij * H(k,j) -
                                                lambda * M(i,k));
                    H(k,j) = H(k,j) + alpha * (2 * eij * M(i,k) -
                                                lambda * H(k,j));
                end
            end
        end
    end

    % Calculate the Squared Error
    e = 0;
    for i = 1:m
      for j = 1:n
          if R(i,j) > 0
              % sum up the error terms
              e = e + ((R(i,j) - (M(i,:)* H(:,j)))^ 2);

              % add regularization point calculations
              % for k = 1: K
              %  e = e + (lambda/2) * (((M(i,k)) ^ 2) + ((H(k,j)) ^ 2));
              % end
          end
      end
    end
    mse = (1/omega) * e;

    % If Squared error is very less, stop iterating.
    if mse < 0.02
       break;
    end
end

% Resultant approximated ratings matrix
tR = M * H;

% round off floating point values to nearest integer
% aR is the approximated R matrix which is the result
aR = round(tR);

% Make sure boundary overflow cases if any are handled */
for i = 1:m
    for j = 1:n
        if aR(i,j) > b
            aR(i,j) = b;
        elseif aR(i,j) < a
            aR(i,j) = a;
        end
    end
end
```

Question 2: K-Means Clustering

Task Definition:

The main task is the Study and analysis of K-Means clustering algorithm. It involves applying K-means clustering on MovieLens-100k Dataset and studying the behaviour of the algorithm in clustering the dataset for various parameters.

Description:

K-means algorithm is the most popular partitional clustering algorithm. It generates a K-partition of the dataset and the clusters are represented by their respective centroids.

Input: Dataset X and number of Clusters K

Output: A K-partition of X

Steps:

K-Means clustering [MacQueen]

1. Randomly select K instances as cluster centers.
2. Label every data point with its nearest cluster centers.
3. Re-compute the cluster centers.
4. Repeat the above two steps until no instances change clusters or certain iterations have gone by

Movie lens dataset contains 100,000 ratings (1-5) from 943 users on 1682 movies. Each user has rated at least 20 movies. Our job is to find some **useful grouping among users or movies** by applying k-means algorithm on the available data which can then be used for recommendations.

I have used k-means algorithm and tried two different things on this dataset. Firstly, I have taken the data files containing movie details. According to given dataset, genres for a movie can be one or more among 19 different types. I have applied **K-means clustering algorithm on this movies and tried to group movies**. My ultimate goal here is to group movies in such a way that movies within a cluster are similar i.e. they are of the same genre. Note that some movies are in more than one genre in the dataset. But in my clustering each movie belongs to exactly one cluster. That is clusters don't share data points.

I have used **Euclidean distance, cosine similarity** as distance measures and tried clustering the data separately.

Tools used:

MATLAB R2012a

Cluster Movies:

First I have used Euclidean distance as measure and obtained the following results. Run the k-means algorithm repeatedly **5 times, each time creating 19 clusters**. This is required because the **initial centroids chosen randomly** may have an effect on the result of clustering. I will take the clustering result of the iteration which produces minimum error ('sumd value' which is the sum of distance from each point within a cluster to its cluster centroid) among them. Though I have specified 500 as maximum number of iterations, the algorithm was converging to a result typically in 10 to 13 iterations.

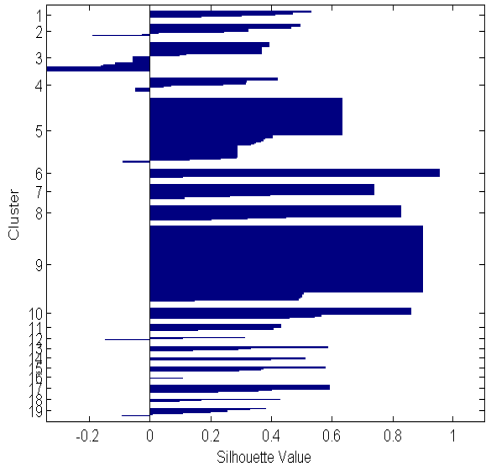
Outline steps:'

1. Import dataset from the file **'u.item'** into matlab
2. Set the distance measure to be used by the algorithm
3. Input number of clusters required.
4. Set the maximum number of iterations allowed and number of times to repeat the algorithm before producing output.
5. Run the K-Means algorithm which works as briefed earlier.

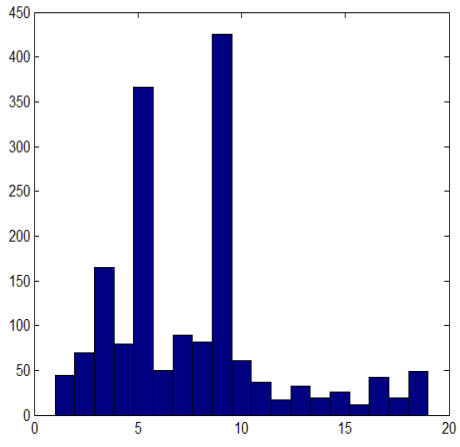
The 19 clusters are then **compared with the actual genre data** available and it was evident clustered data reflected genre based grouping correctly bearing in mind those cases where there were movies with one more than one genres in original data set.

To get an idea of how well-separated the resulting clusters are, I have used the **silhouette plot** using the cluster indices output from kmeans. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighbouring clusters. This measure ranges from +1, indicating points that are very distant from neighbouring clusters, through 0, indicating points that are not distinctly in one cluster or another, to -1, indicating points that are probably assigned to the wrong cluster.

Mean(silhouette value) = 0.5260



(Fig 1)



(Fig 2)

Fig 1 is the silhouette diagram of the clustering

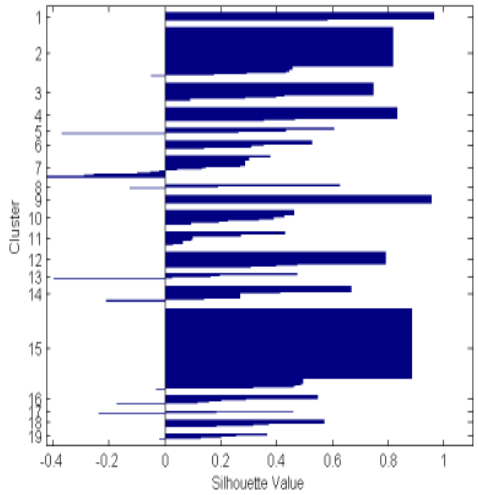
Fig 2 shows that there are 19 clusters and the number of data points assigned to each cluster as a histogram.

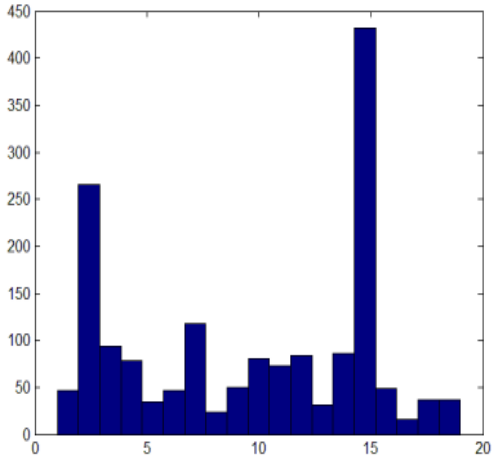
Then I have used **cosine similarity** as measure and obtained the following results. range. Given two vectors x and y, the cosine of the angle between them is the dot product x.y divided by the L2-norms of x and y (i.e., their Euclidean distances from the origin).

Cosine similarity measure considers the angle or direction of data vectors but it does not take into account the magnitude of data vectors. Run the k-means algorithm repeatedly 5 times each time creating 19 clusters.

The 19 clusters are then compared with the actual genre data available. The results were good with cosine similarity as well since our movies data had a representation of using 0/1 for indicating whether it belongs to a genre or not. Though the results from Euclidean measure and cosine similarity can't be compared one to one, the results majorly reflected the genre of movies as clusters in both cases.

Mean(silhouette value) = 0.5816





Cluster Users:

Outline steps:

1. Import dataset from the file **'u1.base'** into matlab
2. Run NMF factorization and fill missing entries using problem 1 of assignment
3. Normalized the data matrix by finding mean and standard deviation.
4. Set the distance measure to be used by the algorithm
5. Input number of clusters required.

6. Set the maximum number of iterations allowed and number of times to repeat the algorithm before producing output.
7. Run the K-Means algorithm which works as briefed earlier.
8. Calculate the sum of squared error distance.

I have tried to cluster users by analysing the ratings they have given for those 1682 movies. We can apply k-means on the dataset given directly. But the given dataset includes so many missing entries, it's better **to apply NMF and predict those missing entries first**. Then K-means algorithm can be applied on these completely filled in ratings matrix to cluster users.

Another issue is when using ratings data, it must be noted that some users may be too generous in rating who always give 5 as rating for a movie. Meanwhile there may be users who are conservative in giving ratings whose highest rating for any movie may be only 3. Hence I have decided to **normalize the data before clustering**.

Normalization of data is done as follows. We will calculate Mean rating for every user over the movies he/she has rated. Also calculate standard deviation for the user's rating. From every rating we will subtract the user's mean rating and divide it by their standard deviation. This is will give us normalized ratings matrix.

I have used **Euclidean distance, cosine similarity and city block** distance as distance measures and tried clustering the dataset.

Setup data using NMF technique as explained in answer to assignment problem 1. This essentially involves taking the 't1.base' dataset and applying NMF along with gradient descent technique to predict missing entries. Then apply k-means algorithm on the complete ratings matrix.

First I have used Euclidean distance as measure and obtained the following results. Run the k-means algorithm repeatedly 5 times each time creating K clusters. This is required because the initial centroids chosen randomly may have an effect on the result of clustering. I will take the clustering result of the iteration which produces minimum error among them. The algorithm was converging to a result typically in 40 to 50 iterations.

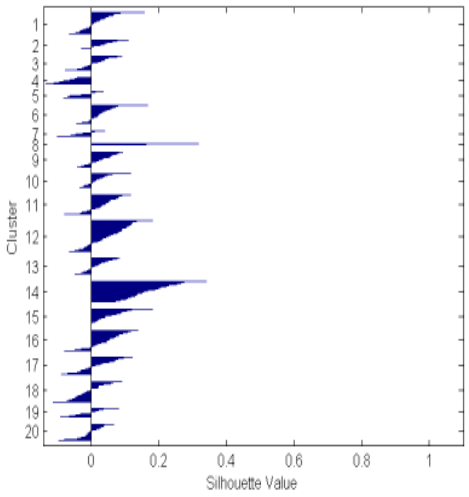
The quality of clustering is measured by **Sum of Squared Error (SSE)** i.e. we calculate the error of each data point as the Euclidean distance to the closest centroid, and then compute the total sum of the squared errors. SSE is given by,

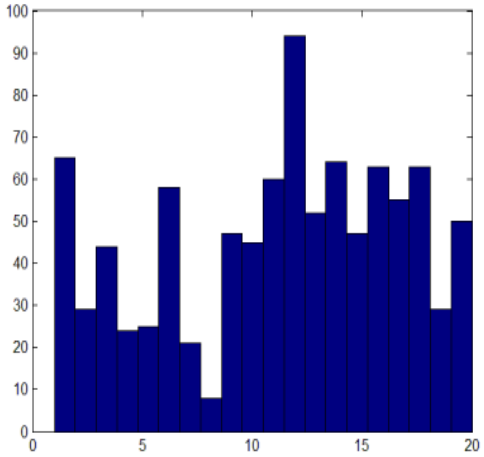
$$\sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

The users were clustered according to their preference for movies. The users within same clusters had similar kind of ratings for the movies. The results were even better when the clustering was applied on normalized data. But comparing the results with dataset for checking the effectiveness of clustering was not easy manually.

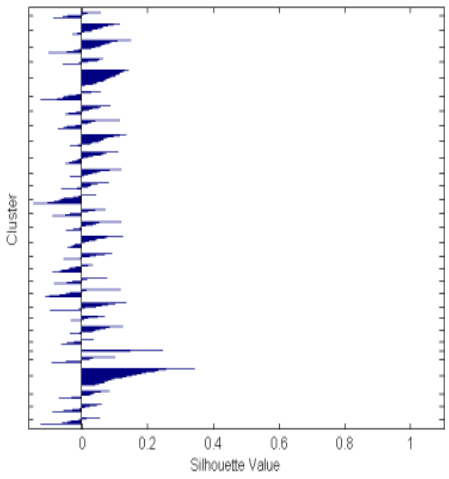
Note: **‘u1.base’** dataset after NMF is used for below illustrations.

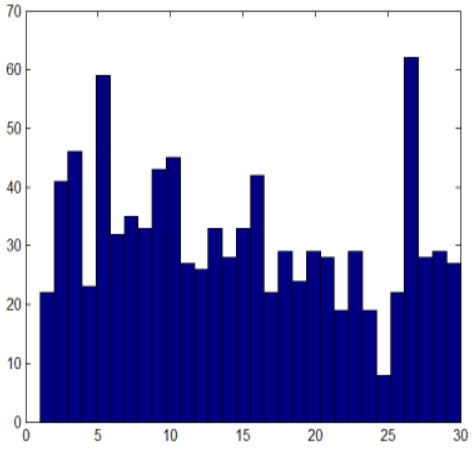
K = 20





K = 30

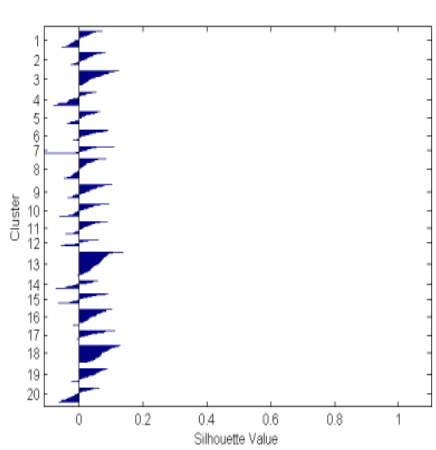


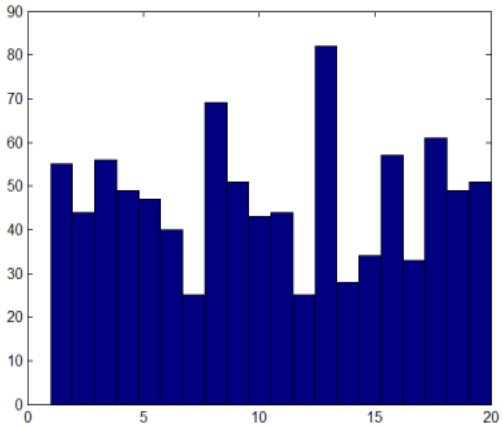


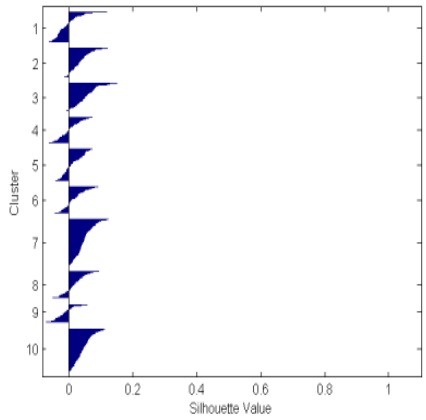
On Normalized Data:

Note: **‘u1.base’** dataset after NMF is **normalized** and used for below illustrations.

K = 20







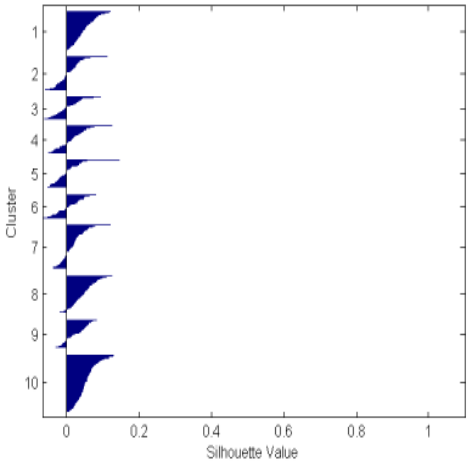
Clustering on normalized data with Euclidean distance measure gives better results as user's ratings are all normalized and recommendations can be more accurate. If two users are in same cluster, they have high commonality in watching movies.

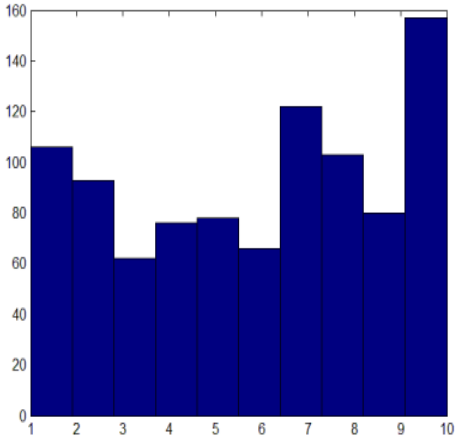
Using **cosine similarity** as measure gives the following results. Run the k-means algorithm repeatedly 5 times each time creating K clusters. Cosine similarity measure considers the angle or direction of data vectors but it does not take into account the magnitude of data vectors. The analogous quantity to the total SSE is the total cohesion which is given by

Total cohesion is given by,

$$\sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \text{Cosine}(x, c_i)$$

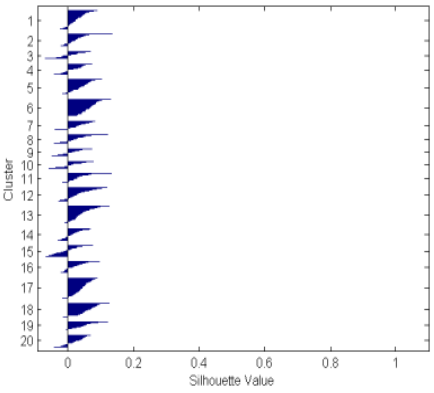
K = 10, 43 iterations, total sum of distances = 296.156.

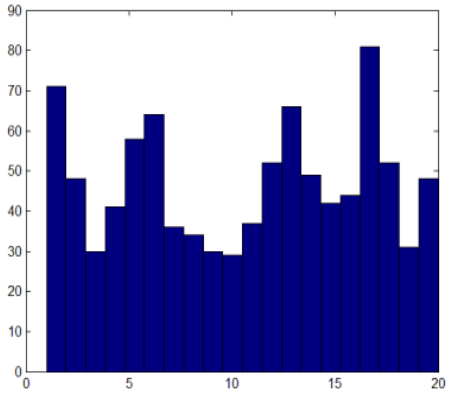




K = 20

33 iterations, total sum of distances = 278.991

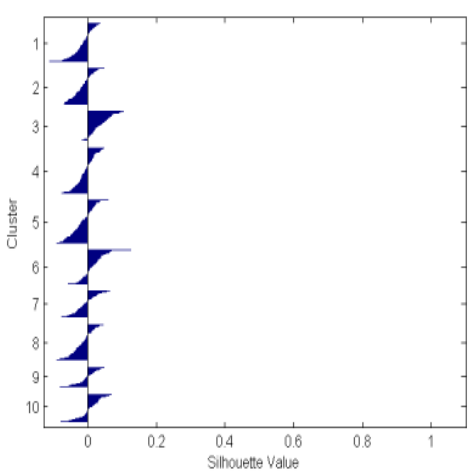


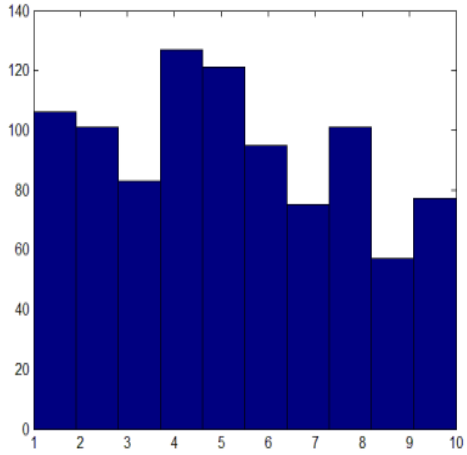


The clusters created by cosine similarity based k-means clustering are compared by manual checking with dataset. The clustering has grouped users who have given ratings relatively same into same clusters than those who rated it differently.

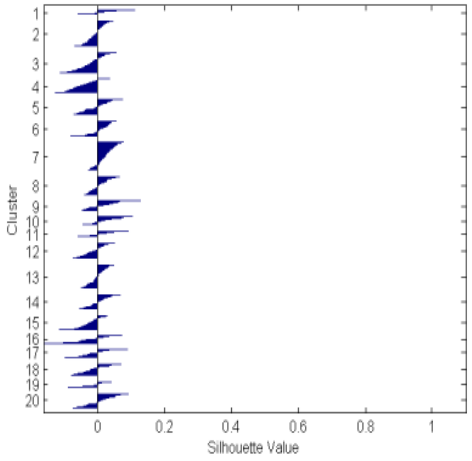
City block distance measure is also used for clustering. This measure is a form of geometry in which the usual distance function or metric of Euclidean geometry is replaced by a new metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. The objective is to minimize the L_1 distance (Manhattan distance) of the data points to its cluster centroids.

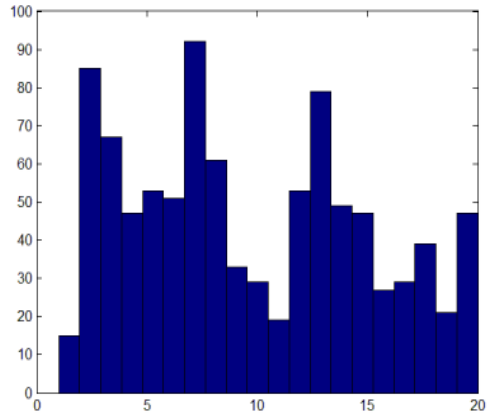
K = 10





K = 20





The city block distance measure with K = 10 is performing badly compared to Euclidean distance and cosine similarity measures for this dataset. K = 20 produces improved clustering but when comparing the cluster results with dataset, the clustering produced by other two approaches are better.

K-means++:

Finally I have tried **k-means++** algorithm for choosing initial clusters along with Euclidean distance measure. This algorithm only has a variation in choosing the initial centroids from traditional k-means. So the implementation is similar to k-means except the extra computation of cluster centroids at the beginning. These centroids are then given to k-means as initial centroids. The first cluster centroid is chosen uniformly at random from the

data points that are being clustered, after which each subsequent cluster centroid is chosen from the remaining data points with a chance of choosing it proportional to its squared distance from the point's closest existing cluster centroid.

The initial selection in the algorithm takes extra time due to additional computations but, the k-means part itself converges very quickly thus actually lowers the running time of the algorithm. It produced little improvement in the final error of k-means.