CSE601: Data Mining

Project 2
Clustering Algorithms

Group: 31

Anuj Rastogi – 5013 4324

Nalin Kumar – 5017 0479

Pranshu Pancholi – 5016 9864

1) <u>K-Means Clustering</u> –

Description –

K-Means clustering is a partition based clustering algorithm. In this algorithm we have to specify initially, the number of clusters, number of iterations and need to specify initial centroid dimensions corresponding to the given number of clusters. For this, either we specify cluster centroid dimensions in the initial configuration properties file or choose initial centroids randomly from the dataset, if the initial cluster centroid dimensions are missing. Picking centroid dimensions from the given dataset ensures that we achieve convergence in lesser number of iterations. Choice of centroids is extremely essential since it governs the number of iterations at which the centroid dimensions stop changing and the clusters formed till this point doesn't change further and become the final clusters. Henceforth, we believe that choosing random centroids from the dataset is a good strategy since choosing a random point unknowingly as a centroid might produce poor clusters and may take greater number of iterations to converge. Subsequent to this, in the initial pass, we calculate the Euclidian distance between each data point and each cluster centroid and assign the data point to the cluster with closest centroid in terms of Euclidean distance. Finally, we perform an iteration step at which we repeatedly recalculate centroid dimensions for each centroid based on the dimensions of the data sets assigned to that cluster and assign clusters to the data sets based on new centroid dimensions using Euclidean distance. This iteration step runs as many times as are the number of iterations specified initially in the k means configuration parameters. Usually, the number of iterations are chosen so as to ensure that k means clustering achieves convergence before the specified number of iterations which can only be known by running the algorithm several times for different combinations of number of clusters and number of iterations.

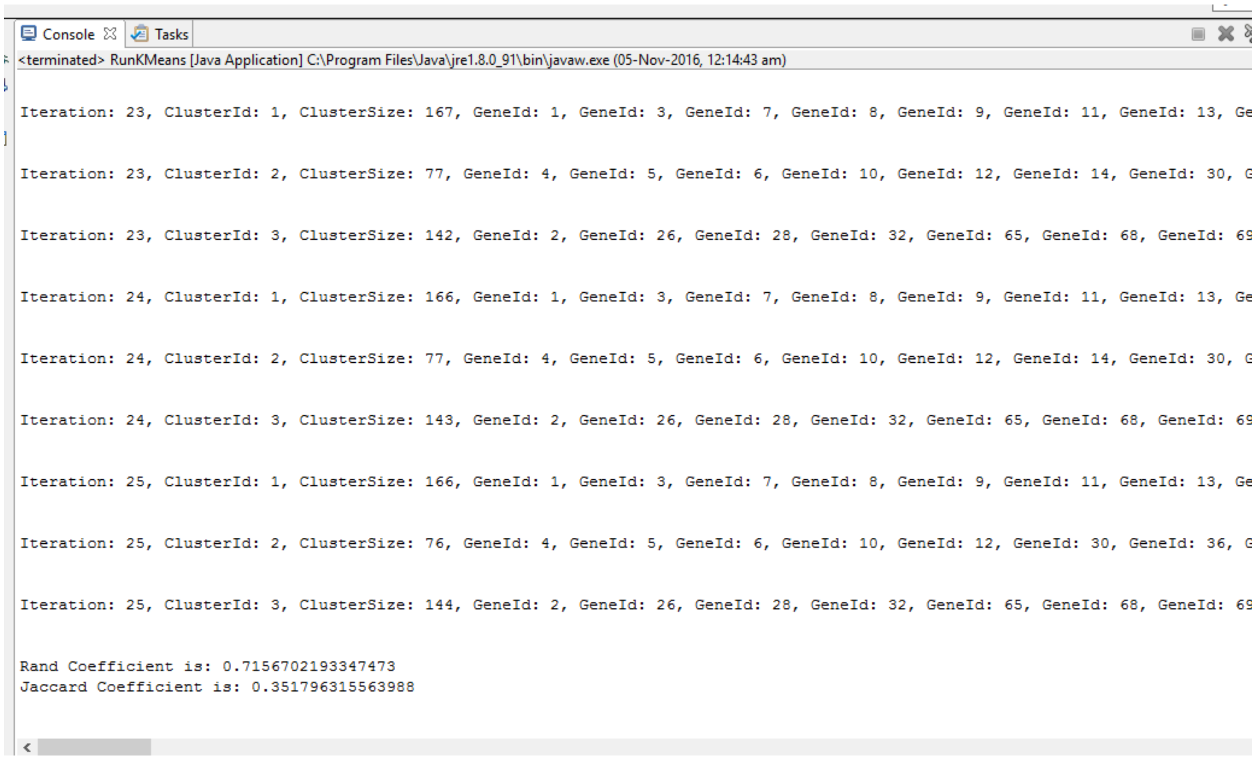
Algorithm Implementation –

1) Initial K Means configurations parameters such as number of clusters, number of iterations, input file name, initial cluster centroid dimensions etc. are all declared as static variables so that they can be reused throughout the entire program and are read as well as populated from a single configuration file in the initial step. This approach helped in removing hard coding from some parts of the program and one can easily change the initial configuration parameters very easily from a single location.
2) Input gene sample data file is read by the program in the next step and a single static array list is created to hold the gene sample data which will be reused throughout the entire program

3) In the next step, the initial cluster centroids are populated. If the cluster centroid dimensions are specified in the properties file as comma separated values, they will be populated using the values specified the file. However, if the cluster centroid dimensions values are empty in the initial configurations file which is the default case, the cluster centroid dimensions will be populated using random gene sample data set dimensions by picking up one data set randomly at a time and this is done for all the clusters. This approach helps in reducing the number of iterations required for K Means algorithm to converge since the centroid lies close to the data set initially as opposed to choosing a random unknown point which can be far away from the dataset initially.
4) We used two java objects, namely, cluster and gene sample which hold all the important information related to clusters and gene samples at every stage of the program. This object oriented approach helped tremendously in avoiding using lots of temporary variables to hold information related to clusters and genes at every iteration step. Also, we maintained a static cluster to gene hash map which stores the cluster id as key and its respective gene samples as values at every single iteration of the K Means algorithm which helped in retaining information related to every iteration in a single object and improved efficiency since hash map provided constant look up performance.
5) Subsequent to populating initial centroids, we assigned clusters to data sets based on the minimum Euclidean distance between a gene sample and a particular centroid in the initial pass of the algorithm.
6) The algorithm then proceeds by repeating the following steps as many number of times as the number of iterations specified initially in the program configuration :-
    - Iterate for all the clusters and recalculate the centroid dimensions for every single based on the median value of all the individual dimensions of all the gene samples assigned to that cluster through previous iteration
    - Clear the cluster to gene hash map to reassign gene sample data sets in accordance with the new cluster centroid dimensions
    - Assign clusters to all the data sets based on the minimum Euclidean distance between gene sample dimensions and new cluster centroid dimensions as well as populate the cluster to gene map to maintain info ration related to all the clusters assigned to their respective gene samples in a particular iteration step of the algorithm

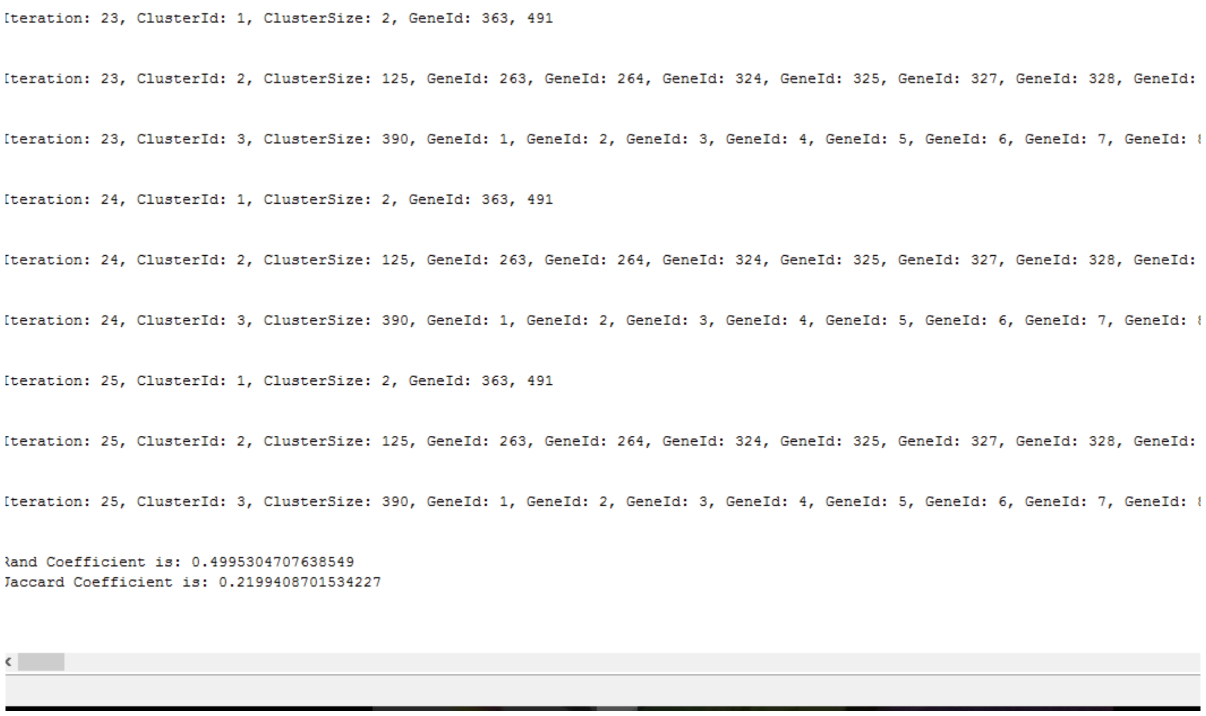
Result Evaluation –

- Although the best approach might be to know the best possible set of initial centroids which can perform very good clustering with minimal number of iterations. But since, since there is lack of such information initially, henceforth, we chose initial centroids from the given dataset randomly since it is atleast guaranteed that this data set will be closer to the dataset as compared to choosing a random unknown point which might be far away from all the data points. This is important since this might increase the number of iterations required to achieve convergence in K Means Clustering algorithm and lead to poor performance. The only point worth mentioning here is that our approach leads to convergence in different number of iterations and different values of external index coefficients every time we run the same algorithm for same number of clusters and same number of maximum iterations
- Number of iterations impact the resultant clustering. Henceforth, the best approach is to run the algorithm number of times for different number of clusters and different number of iterations and thereby choose intelligently, the maximum number of iterations such that the algorithm produces non-changing clusters before the algorithm terminate. The number of iterations within which most of the K Means runs terminated is 20 for the given datasets and for clusters of sizes 3,5,7
- K-Means algorithm chases the outlier points in the given dataset and assigns them a cluster. As we observed in the PCA visualization below, some of the clusters have been formed a bit inclined towards the outliers. Henceforth, we can say that K-Means algorithm is sensitive to outlier and noise points
- While running the algorithm several times for different number of clusters and different number of iterations, we observed that on an average, the number of iterations required to achieve non changing clusters increases as we increase the number of clusters. Although this is not guaranteed to happen since we used random initial centroids, but in most of the cases, it took more iterations as we increased the number of clusters from 3 to 5 and from 5 to 7
- We performed clustering without removing the outlier points from the initial dataset. This led to formation of cluster of small sizes even when the number of clusters was only 3. For example, as can be seen in the snapshot below, for cho.txt all the formed clusters are of significant size whereas in case of iyer.txt there is a formation of a very small cluster with only 2 data points even when the number of clusters is very low, k=3. This happened due to the presence of some very far away outlier points which can be clearly comprehended from PCA visualization



```
Console    Tasks
<terminated> RunKMeans [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (05-Nov-2016, 12:14:43 am)

Iteration: 23, ClusterId: 1, ClusterSize: 167, GeneId: 1, GeneId: 3, GeneId: 7, GeneId: 8, GeneId: 9, GeneId: 11, GeneId: 13, Ge

Iteration: 23, ClusterId: 2, ClusterSize: 77, GeneId: 4, GeneId: 5, GeneId: 6, GeneId: 10, GeneId: 12, GeneId: 14, GeneId: 30, G

Iteration: 23, ClusterId: 3, ClusterSize: 142, GeneId: 2, GeneId: 26, GeneId: 28, GeneId: 32, GeneId: 65, GeneId: 68, GeneId: 69

Iteration: 24, ClusterId: 1, ClusterSize: 166, GeneId: 1, GeneId: 3, GeneId: 7, GeneId: 8, GeneId: 9, GeneId: 11, GeneId: 13, Ge

Iteration: 24, ClusterId: 2, ClusterSize: 77, GeneId: 4, GeneId: 5, GeneId: 6, GeneId: 10, GeneId: 12, GeneId: 14, GeneId: 30, G

Iteration: 24, ClusterId: 3, ClusterSize: 143, GeneId: 2, GeneId: 26, GeneId: 28, GeneId: 32, GeneId: 65, GeneId: 68, GeneId: 69

Iteration: 25, ClusterId: 1, ClusterSize: 166, GeneId: 1, GeneId: 3, GeneId: 7, GeneId: 8, GeneId: 9, GeneId: 11, GeneId: 13, Ge

Iteration: 25, ClusterId: 2, ClusterSize: 76, GeneId: 4, GeneId: 5, GeneId: 6, GeneId: 10, GeneId: 12, GeneId: 30, GeneId: 36, G

Iteration: 25, ClusterId: 3, ClusterSize: 144, GeneId: 2, GeneId: 26, GeneId: 28, GeneId: 32, GeneId: 65, GeneId: 68, GeneId: 69

Rand Coefficient is: 0.7156702193347473
Jaccard Coefficient is: 0.351796315563988
```

For k=3 and file=cho.txt, all the clusters are of significant sizes



```
Iteration: 23, ClusterId: 1, ClusterSize: 2, GeneId: 363, 491

Iteration: 23, ClusterId: 2, ClusterSize: 125, GeneId: 263, GeneId: 264, GeneId: 324, GeneId: 325, GeneId: 327, GeneId: 328, GeneId:

Iteration: 23, ClusterId: 3, ClusterSize: 390, GeneId: 1, GeneId: 2, GeneId: 3, GeneId: 4, GeneId: 5, GeneId: 6, GeneId: 7, GeneId:

Iteration: 24, ClusterId: 1, ClusterSize: 2, GeneId: 363, 491

Iteration: 24, ClusterId: 2, ClusterSize: 125, GeneId: 263, GeneId: 264, GeneId: 324, GeneId: 325, GeneId: 327, GeneId: 328, GeneId:

Iteration: 24, ClusterId: 3, ClusterSize: 390, GeneId: 1, GeneId: 2, GeneId: 3, GeneId: 4, GeneId: 5, GeneId: 6, GeneId: 7, GeneId:

Iteration: 25, ClusterId: 1, ClusterSize: 2, GeneId: 363, 491

Iteration: 25, ClusterId: 2, ClusterSize: 125, GeneId: 263, GeneId: 264, GeneId: 324, GeneId: 325, GeneId: 327, GeneId: 328, GeneId:

Iteration: 25, ClusterId: 3, ClusterSize: 390, GeneId: 1, GeneId: 2, GeneId: 3, GeneId: 4, GeneId: 5, GeneId: 6, GeneId: 7, GeneId:

Rand Coefficient is: 0.4995304707638549
Jaccard Coefficient is: 0.2199408701534227
```

For k=3 and file=iyer.txt, there is cluster of size 2 and one cluster with very huge size

Pros and Cons of the Algorithm –

Pros-

1) K-Means algorithm is easy to understand and implement.
2) It has fairly low running time of O(K*N*I) and is suitable for large datasets, here
   K = number of initial clusters
   N = number of data points
   I = number of iterations
3) It gives best results when the clusters are distinct and far apart
4) It doesn't produce overlapping clusters since it is a hard clustering technique

Cons-

1) Value of K i.e. number of clusters need to be specified in the beginning
2) Sensitive to outliers and noise points
3) Cannot handle irregular shaped data
4) Random choice of initial centroids may not yield the best results
5) Due to the presence of outliers and noise points, clustering algorithm may produce almost empty clusters of very small sizes. This is not considered to be a very good cluster

Experiment Values –

Values of Jaccard Coefficient and Rand Index are mentioned below by choosing random initial centroid points from the dataset for different number of clusters and different number of iterations

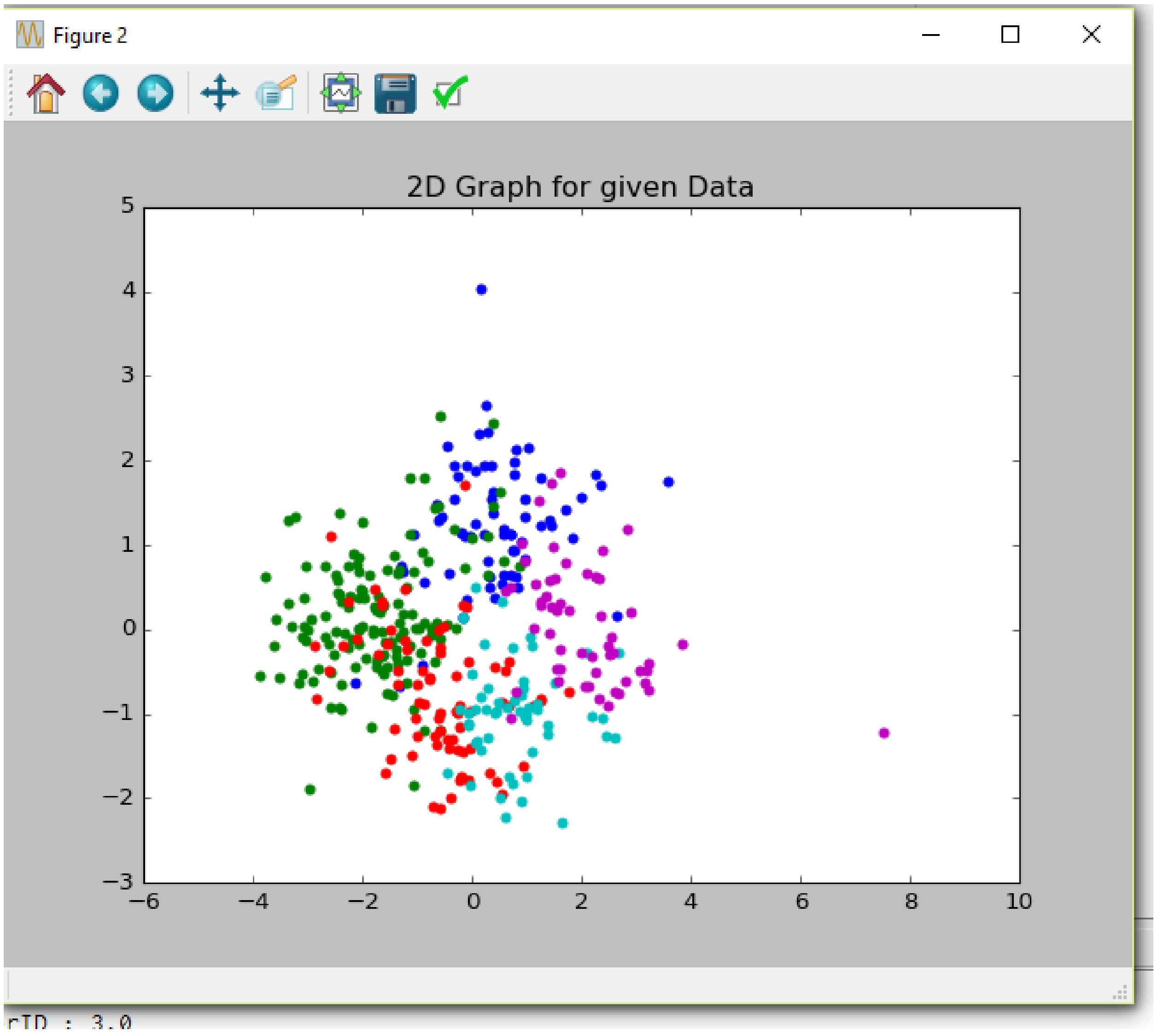
Jaccard and Rnd Coefficient (cho.txt)

Number of clusters-K	**Number of Iterations**	**External Index (Jaccard Coefficient)**	**External Index (Rand Index)**
3	**15**	0.408	0.757
3	**25**	0.351	0.715
3	**40**	0.408	0.757
5	**15**	0.379	0.796
		0.336	0.784

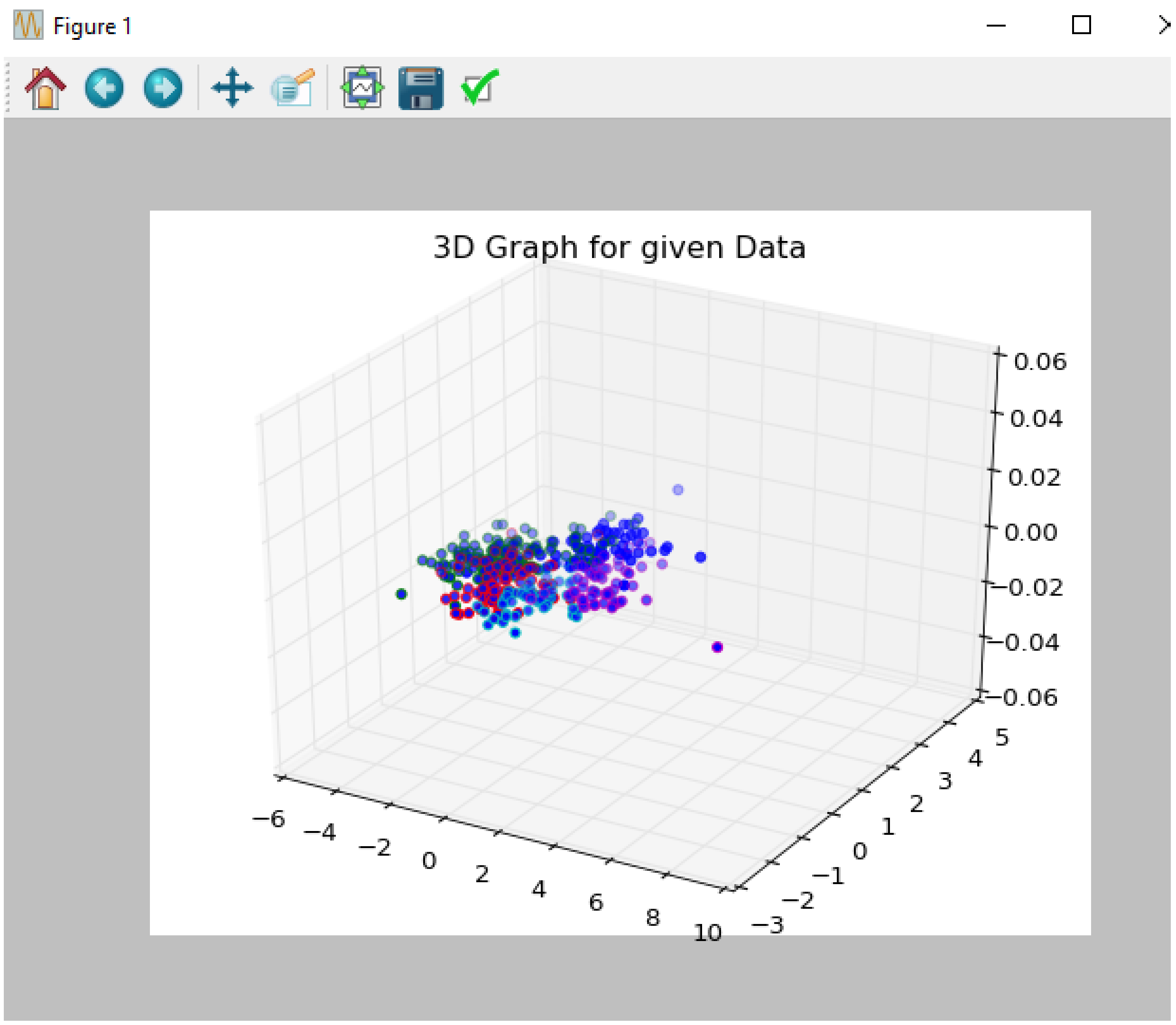
5	**25**		
5	**40**	0.428	0.800
7	**15**	0.328	0.795
7	**25**	0.299	0.785
7	**40**	0.296	0.787

PCA (cho.txt)

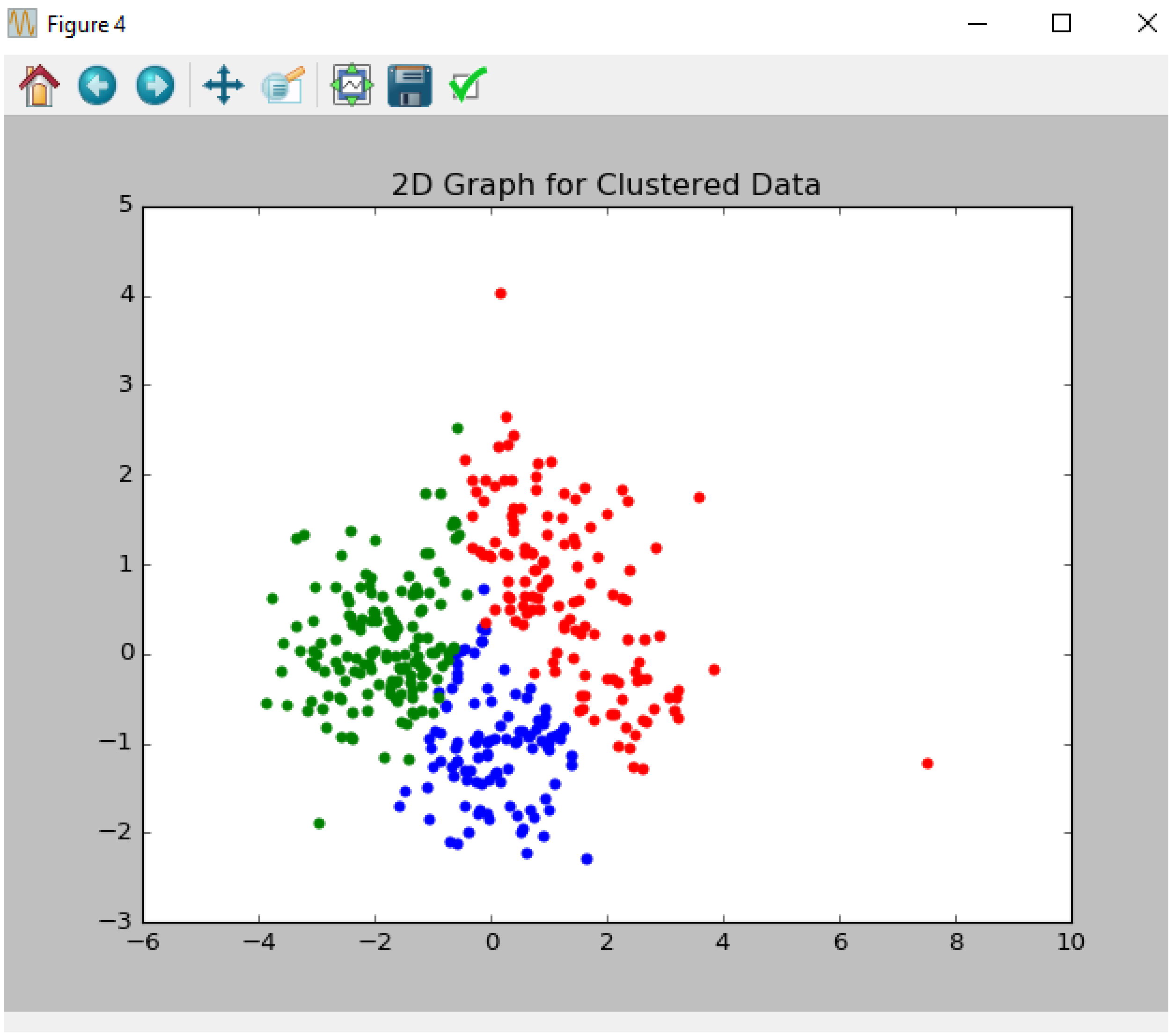
For K = 3, and number of iterations = 40



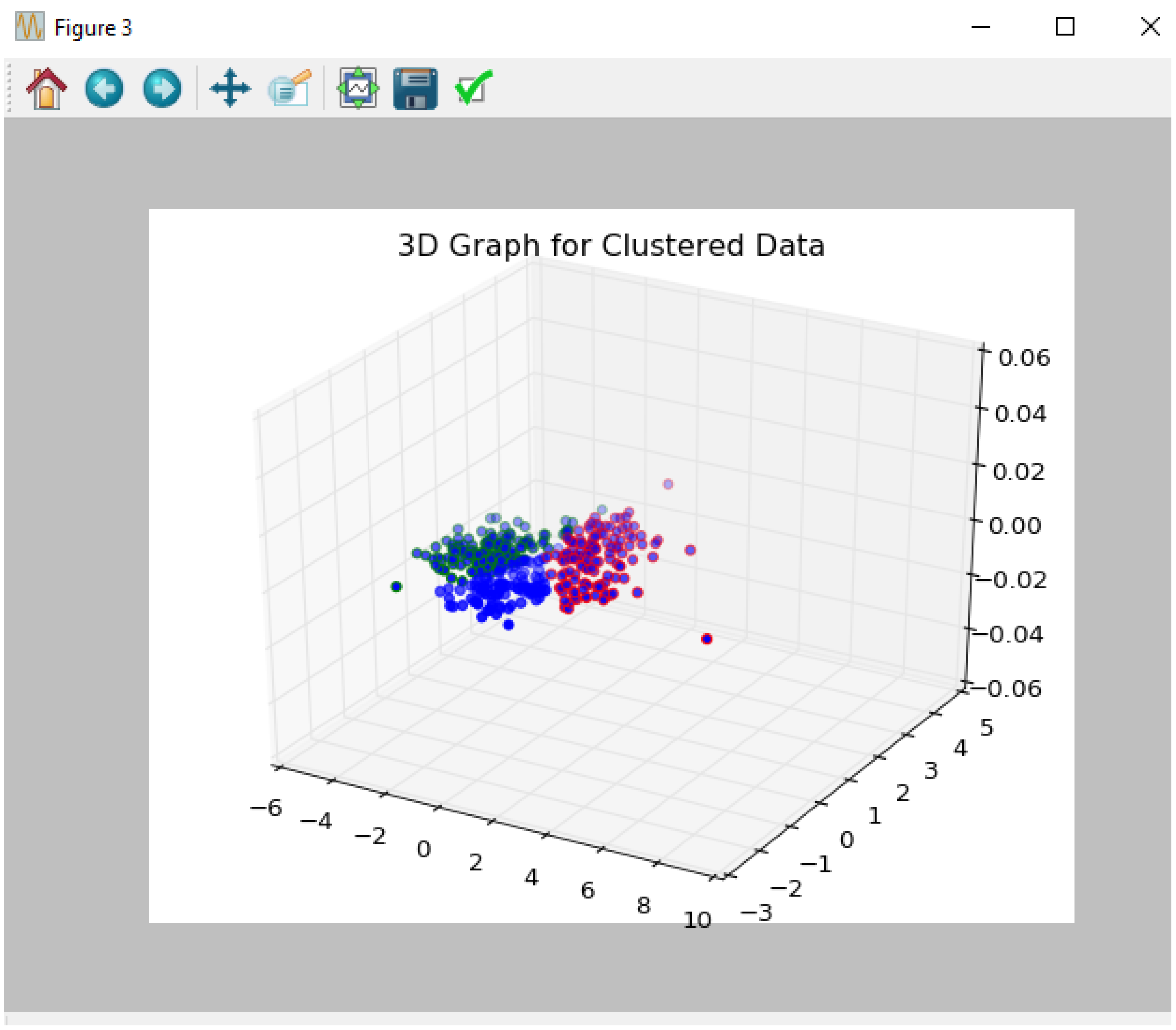
a) **2D graph of given data(cho.txt)**



b) 3D graph for given data(cho.txt)



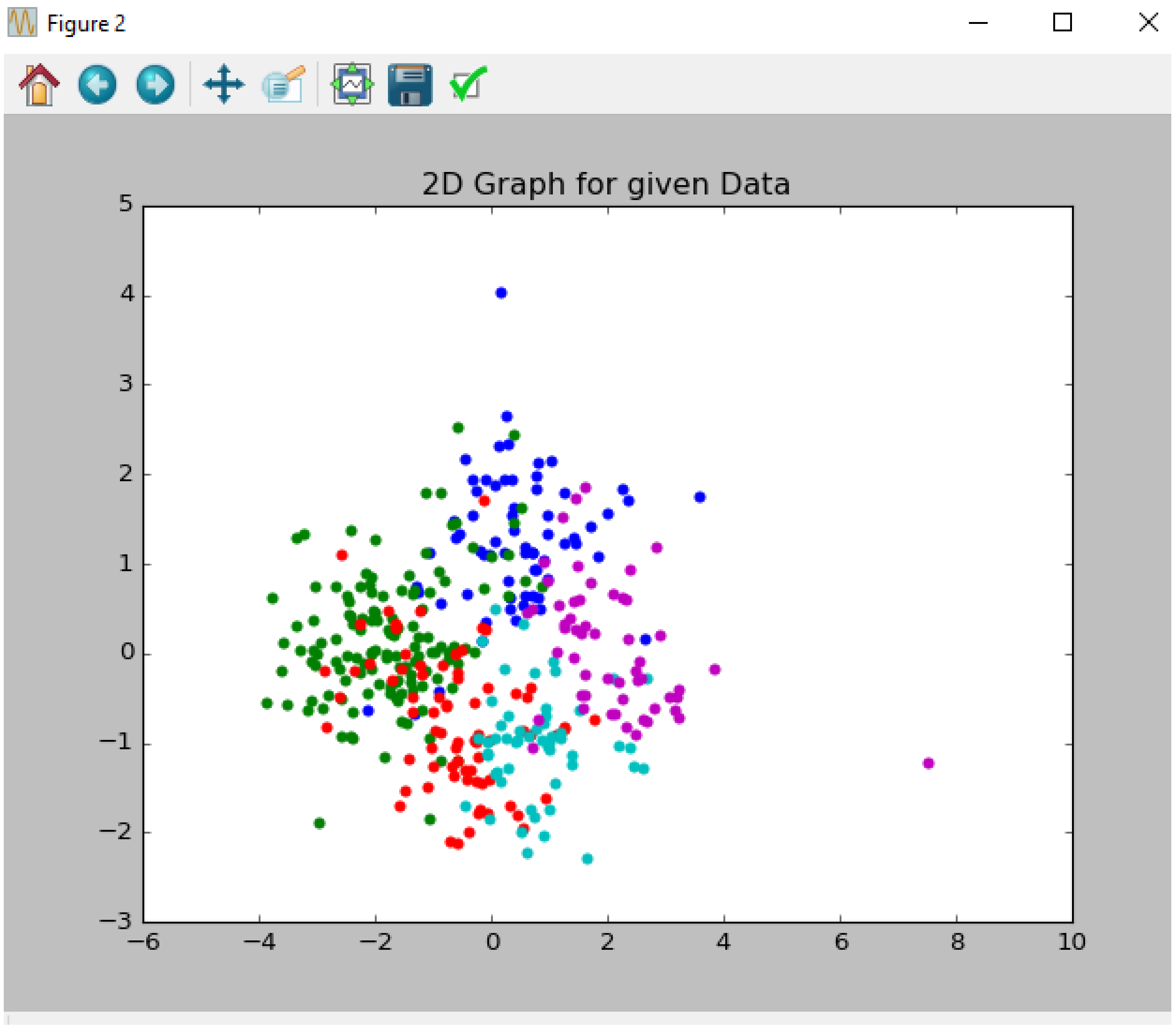
c) 2D graph for clustered data(cho.txt)



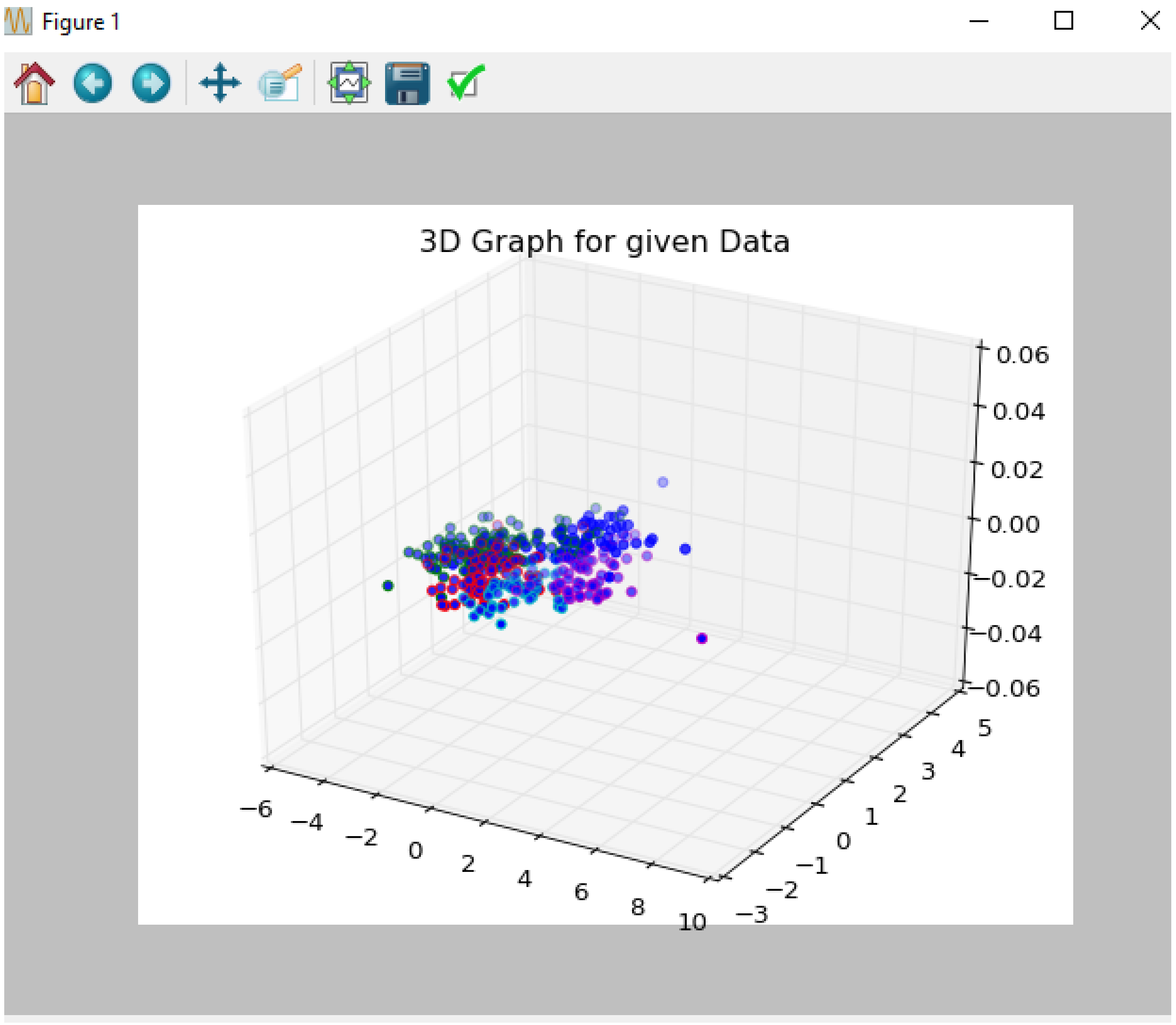
d) 3D graph for clustered data(cho.txt)

PCA (cho.txt)

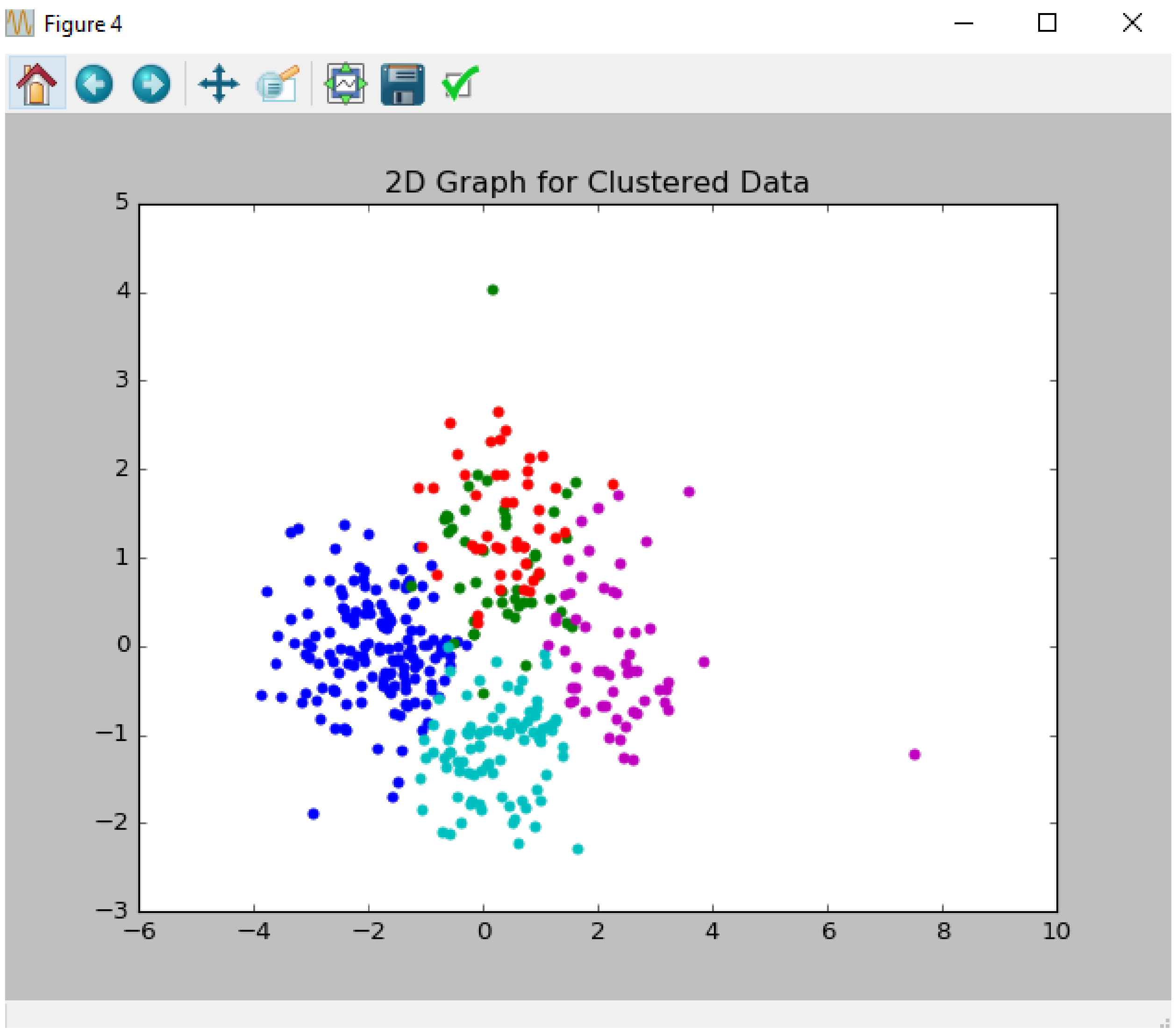
For K = 5, and number of iterations = 40



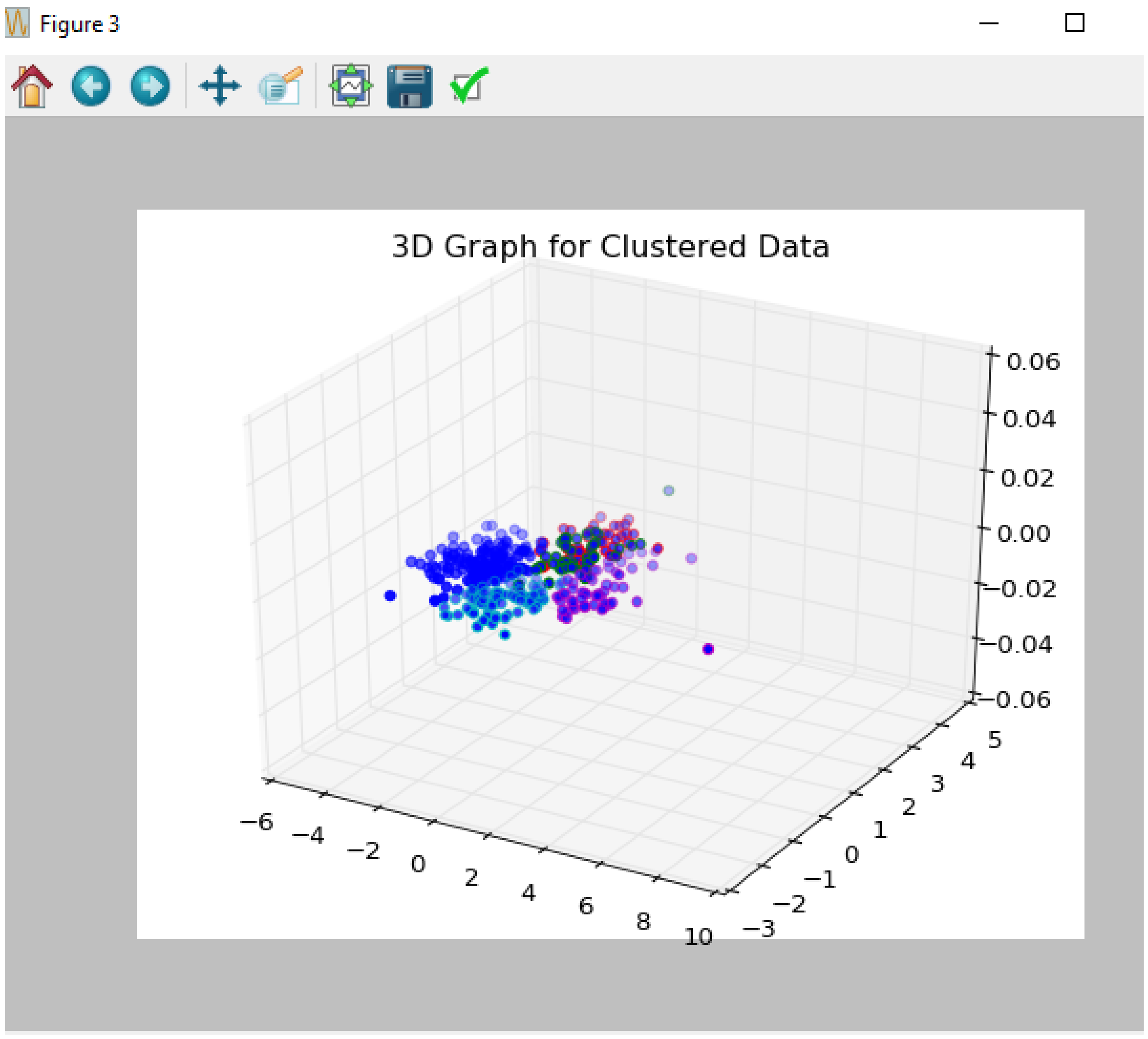
a) 2D graph for given data(cho.txt)



b) 3D graph for given data(cho.txt)



c) 2D graph for clustered data(cho.txt)



d) 3D graph for clustered data(cho.txt)

Results Visualization (cho.txt) –

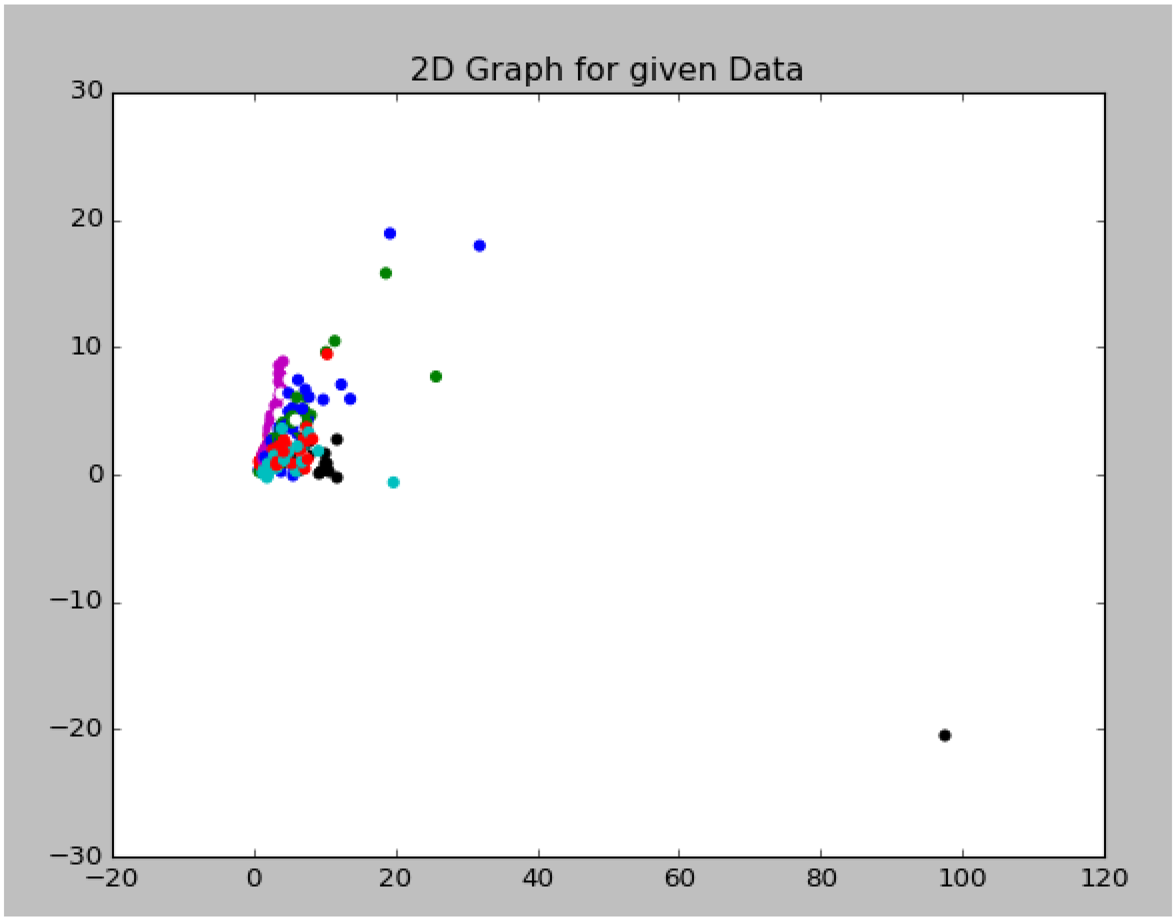
As can be comprehended from above results, Jaccard coefficient, which is a measure of similarity, decreased as we increased the number of clusters to 7 from which it can be deduced that the similarity between given ground truth and clustered data set decreased with number for clusters. We also plotted the PCA results on given data and clustered data in 2D as well 3D for cases where Jaccard coefficient was on a higher side since this might reveal that the there is a good amount of similarity between given ground truth data and clustered data. As can be seen above, the results are plotted for k=3, 5 and for n=40 for which the Jaccard coefficient is the highest. Although more similar as compared to the other combinations of n and k, plotted figures clearly indicate that data is clustered more properly than specified in the given ground truth data after running K Means algorithm for 40 iterations for 3 and 5 clusters. Also, we can see in 3D figures, that there are very few outliers and that too not far away from other data points. This explains the reason for the fact which we stated above that clusters formed for cho.txt are of significant sizes.

<u>Jaccard and Rand Coefficient (iyer.txt)</u>

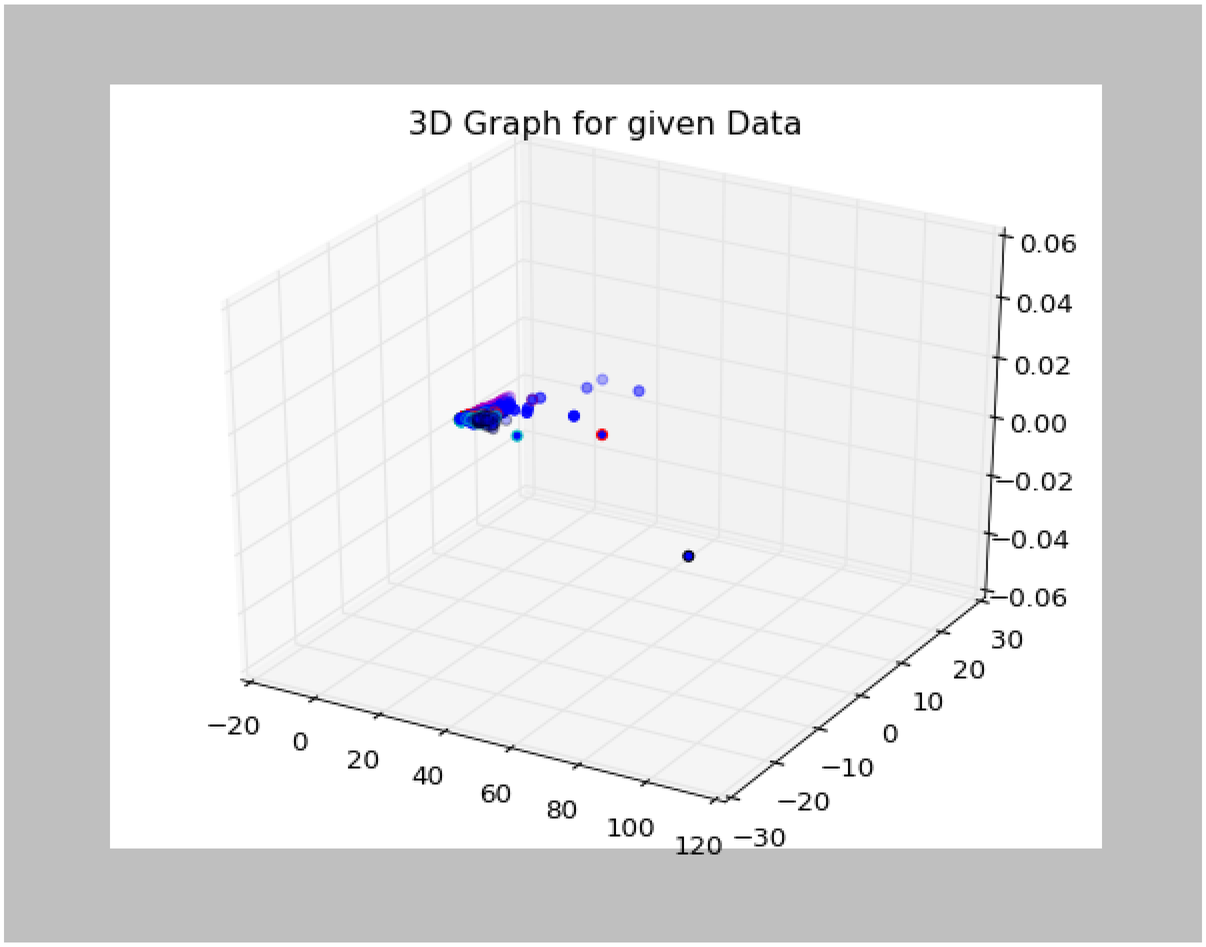
Number of clusters-K	**Number of Iterations**	**External Index (Jaccard Coefficient)**	**External Index (Rand Index)**
3	**15**	0.203	0.443
3	**25**	0.299	0.499
3	**35**	0.275	0.635
3	**40**	0.219	0.499
5	**20**	0.276	0.637
5	**30**	0.265	0.604
5	**50**	0.266	0.606
7	**25**	0.297	0.679
7	**40**	0.287	0.663
7	**50**	0.306	0.688

PCA (iyer.txt)

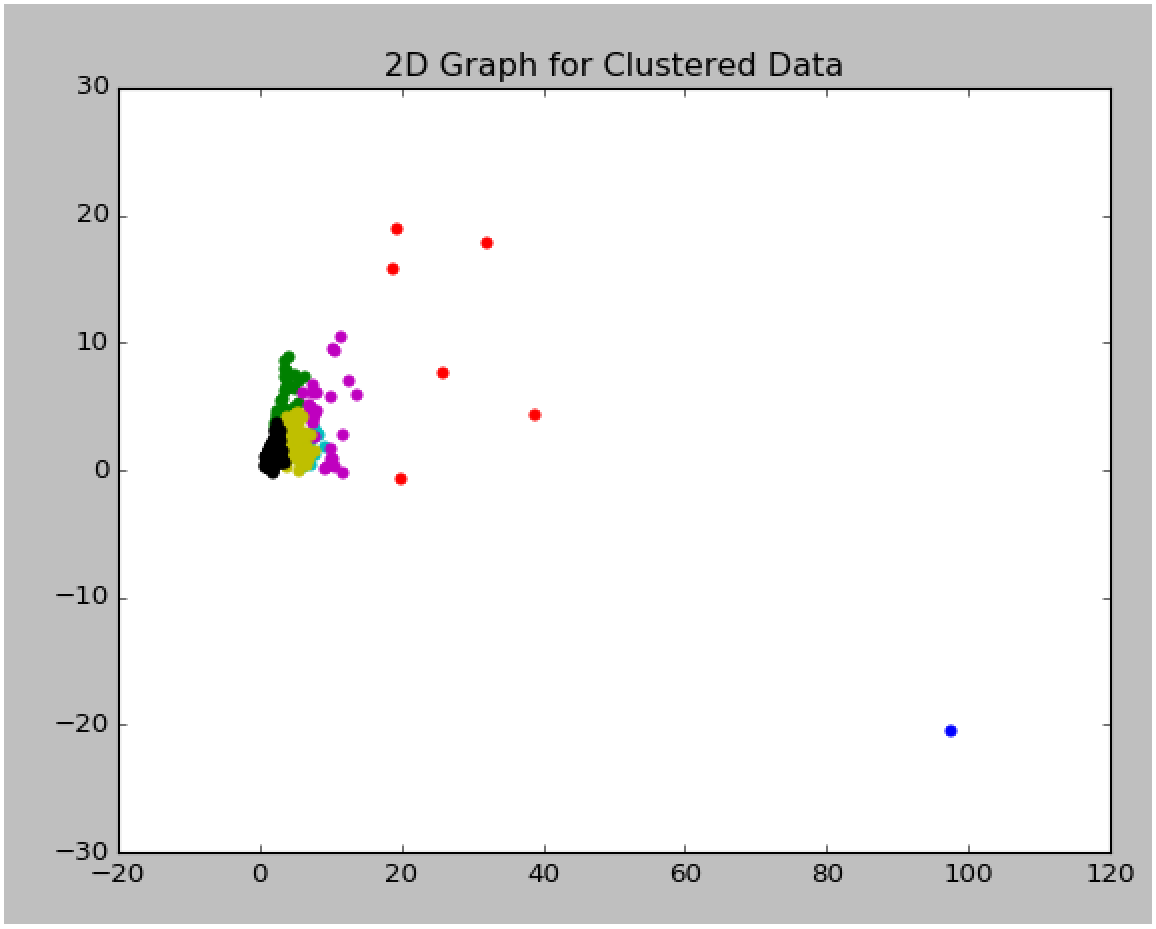
For K = 7, and number of iterations = 50



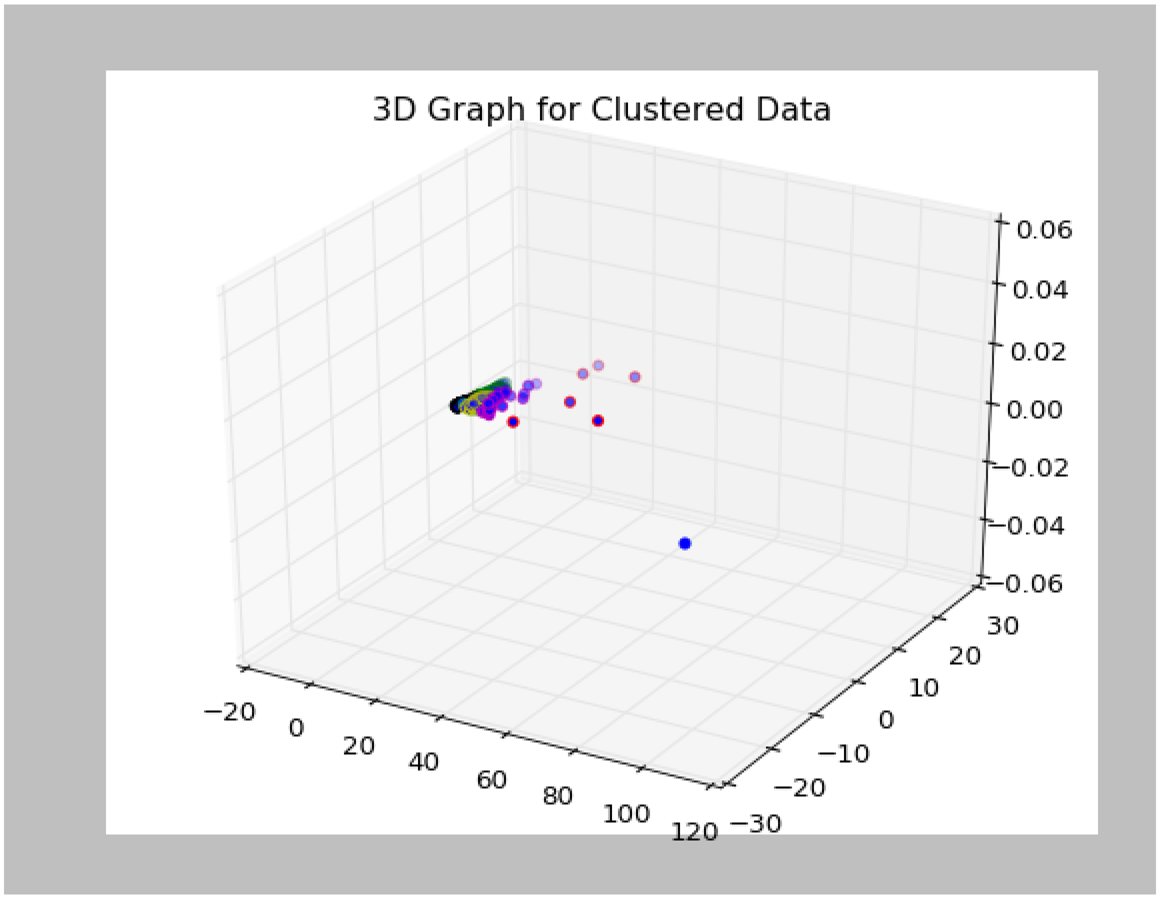
a) 2D graph of given data(iyer.txt)



b) 3D graph of given data(iyer.txt)



c) 2D graph of clustered data(iyer.txt)



d) 2D graph of clustered data(iyer.txt)

Results Visualization (iyer.txt)–

As can be comprehended from above results, Jaccard coefficient, which is a measure of similarity, came out to be very low for k=3 and n=15(k=number of clusters and n=number of iterations). This might be the case that since there are too many outliers in the dataset and number of clusters are small, the clustered data might be deviating the most from given ground truth data when number of clusters are very small. . We also plotted the PCA results on given data and clustered data in 2D as well 3D for the case for which Jaccard coefficient was on a higher side since this might reveal that the there is a good amount of similarity between given ground truth data and clustered data. As can be seen above, the results are plotted for k=7 and for n=40 for which the Jaccard coefficient is the highest. Although more similar as compared to the other combinations of n and k, plotted figures clearly indicate that data is clustered more properly than specified in the given ground truth data after running K Means algorithm for 40 iterations for 7 clusters. Also, we can see in 3D figures, that there are more outlier points which are more farther away from the rest of the data points in iyer.txt as compared to cho.txt file. This explains the reason of the fact that there are some clusters formed for iyer.txt for which cluster size comprises of one or two data points due to the presence of outliers initially in iyer.txt

2) <u>Hierarchical Agglomerative Clustering</u>

Description –

Hierarchical Agglomerative clustering is a bottom-up approach clustering. Initially, each data object is assigned a cluster. In each iteration we merge 2 clusters which are closest together. After the merging step in the end, we are left with a single cluster. Thus, we obtain a dendrogram of clustering of data objects. By cutting the dendrogram at the desired level we obtain the desired clustering of the data objects.

To find the inter-cluster distance we have used **Single link MIN** approach wherein we find the 2 data points in each of the cluster which have minimum distance between them.

Algorithm Implementation–

1) Read the dataset and store it in a 2D matrix of size [m * n] where m is the number of genes in the dataset and n is the number of gene expressions + 2 (for sample id and ground truth).
2) Compute the distance matrix of size [m * m]. The distance matrix is computed by calculating Euclidian distance between each pair of genes.
3) Populate an ArrayList<ArrayList<Integer>> named “clusters” with the initial cluster pattern. Initially, all the genes will be in distinct cluster of size 1. Each element in the “clusters” array will be a single cluster containing all the genes belonging to that cluster.
4) **Repeat** till 1 cluster remains.
    1) Find the minimum entry in the distance matrix which represents the 2 closest clusters. These 2 clusters needs merging.
    2) Update the distance matrix specifically the row and column of the cluster with min index. This will merge the 2 clusters into lower index cluster.
    3) To update the distance of 2 clusters, find 2 genes each belonging to 1 of the cluster and have minimum distance between them (**Single Link MIN**). Update the distance matrix with this minimum distance.
    4) Remove the higher index cluster from the distance matrix.
    5) Update the “clusters” array by merging the 2 clusters and removing the higher index cluster from the “clusters” array.

Result Evaluation –

For cho.txt dataset, Single Link MIN Hierarchical Agglomerative Clustering generated 5 clusters with 382 gene data points belonging to 1 cluster and rest 4 genes belonged to 4 distinct clusters.

For iyer.txt dataset, Single Link MIN Hierarchical Agglomerative Clustering generated 5 clusters with 381 data points belonging to 1 cluster. 3 other gene data points were assigned 3 distinct cluster and 1 other cluster contained 2 gene data points.

Since cho.txt had no outlier points, it had no effect on the algorithm. While iyer.txt had outlier points and the algorithm assigned them to a cluster. Thus, Hierarchical agglomerative clustering is sensitive to outliers and noise points.

Pros and Cons of the Algorithm –

Pros –

1) Gives best results for dataset which has inherent hierarchical structure.
2) Unlike K-means, this algorithm does not require initial number of clusters as input.

Cons –

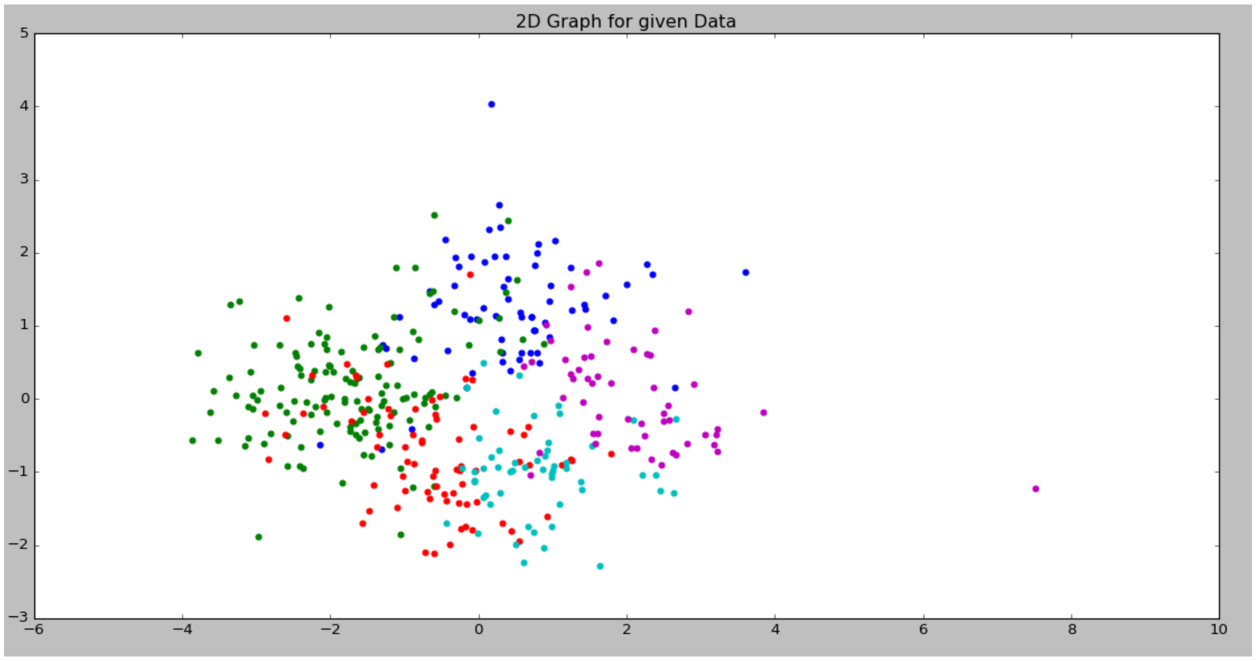
1) It has running time of $O(N^3)$ and space complexity of $O(N^2)$ to maintain the distance matrix. Therefore, not suitable for large size dataset.
2) Very sensitive to noise and outlier points.
3) Difficult to cluster irregular shaped data.

Experiment Values –

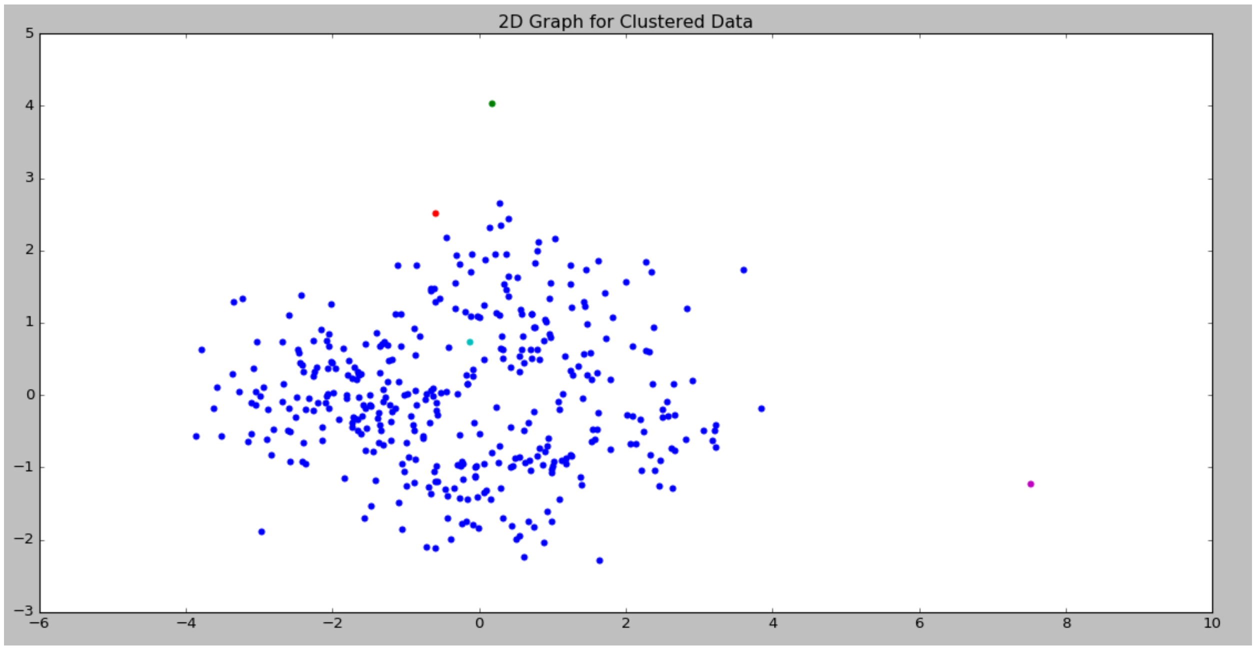
Dataset - cho.txt

Number of clusters	**External Index (Jaccard Coefficient)**	**External Index (Rand Index)**
5	0.22839497757358454	0.24027490670890495
7	0.22879187984815977	0.2473220757604231
9	0.22884212570572487	0.2537786249295283

PCA-



a) 2D graph of Given Data (cho.txt).

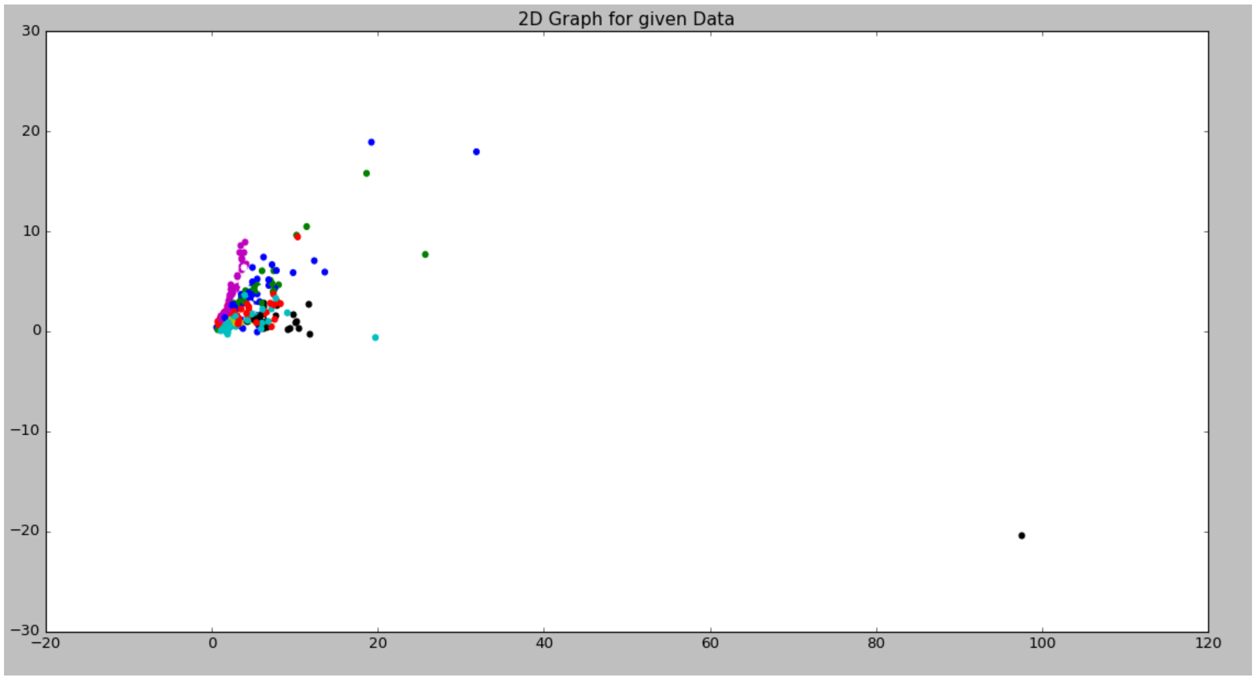


b) 2D graph of clustered data(cho.txt).

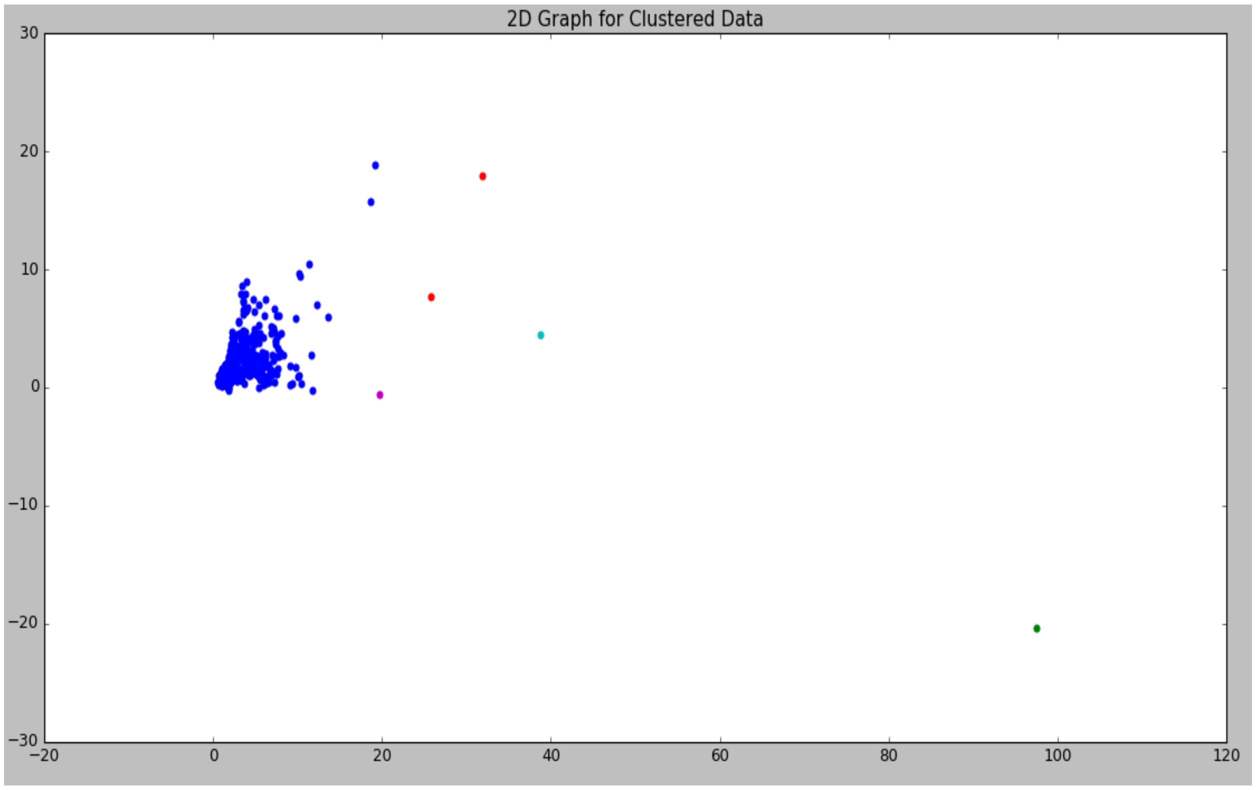
<u>Dataset</u> – iyer.txt

Number of clusters	**External Index (Jaccard Coefficient)**	**External Index (Rand Index)**
5	0.15232436344245748	0.1673694016588786
7	0.1524654734008077	0.17322822862145468
9	0.15402581581305844	0.18422007639670918

PCA



a) 2D graph of given data (iyer.txt).



b) 2D graph of clustered data (iyer.txt).

3) <u>Density Based Scan Clustering</u> –

Description –

Density Based clustering algorithm is an algorithm which tries to chase the density of data. Hence, if it continues to find good density of data based on certain parameters then it classifies them in the same class. One distinct thing about the density based clustering is the fact that it can help us get clusters of varying shapes and sizes. Also, unlike many other clustering algorithms like K means, it has an intelligence to know how many clusters to form based on the data and some assigned parameters.

The basic idea behind clustering is:

1. Clusters are dense regions in the data space, separated by regions of lower object density.
2. A cluster is defined as a maximal set of density-connected points.

3. Discovers clusters of arbitrary shape.

In density based clustering, every point is classified into 2 types based upon the parameters we specify and how much a point satisfies these parameters.

We have 2 parameters to specify:

1. **Epsilon**: The minimum distance from a point or around a point where to look for neighbors.
2. **Minimum Points**: The minimum number of points that have to be present within the specified epsilon, for a point to be categorized as core point.

There are three types of points based on these parameters:

1. **Core Points**: Points which have more than or equal to minimum points around them within a specified region, epsilon, are called as Core Points.
2. **Border Points**: Points which have less than minimum number of points within a specified region, epsilon.
3. **Outliers**: A point neither a core point nor a border point is classified as an outlier.

When the algorithm starts to run, when it finds a core point it reaches all the neighbors of that point one by one. Once it reaches a neighbor it finds if that neighbor is a core point or not. If that point is a core point all its neighbor are also visited and this process is carried on till it reaches an end. Vice versa if it finds a neighbor which is a border point, only that point is considered within the cluster. The neighbor of such a point are not visited. Once all the points which are reachable from the initial point of start are done, a cluster formation is completed.

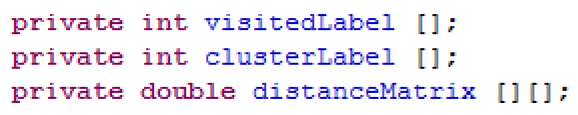
Then the algorithm picks another random point to start clustering.

Algorithm Implementation –

A pseudo code of the algorithm implementation is given below. The following steps are followed.

1. Firstly, two arrays are created within the code. Both array are of the size of the data set. One array records if the point is visited (visited Array) and one records the cluster id (cluster label Array) corresponding to that gene Id.
2. Each index of the array corresponds to a gene Id.

3. One distance matrix is also created, which is a (number of Genes * number of Genes) dimension matrix, where matrix[i][j] has a distance of gene id “i” with gene Id “j”.



```
private int visitedLabel [];
private int clusterLabel [];
private double distanceMatrix [][];
```

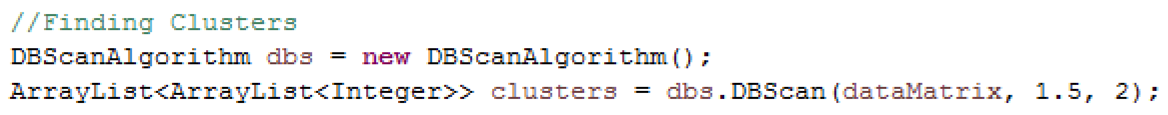
4. When the algorithm starts it picks a point.
5. Then it goes to the distance matrix and the row corresponding to that gene Id.
6. It traverses the row and finds out the gene Ids (indexes) which are within Epsilon distance from that gene Id (corresponding to the matrix row).
7. It then checks if the number of gene Ids close to the point within Epsilon distance, are more than or equal to the specified minimum number of points.
8. If so, the point is a core point, and for this point we do an “expandCluster”, a function below in pseudo code.
9. In expand cluster, it, first of all goes to the visited Array and the index = gene Id of the point. Here it marks the point as visited i.e. “1”.
10. Also, in the cluster label Array at the same index we mark the point in cluster Id, say “X”.
11. Then we start visiting the neighbor gene Ids of the point.
12. For each gene Id if the gene Id has more neighbors, within Epsilon, than specified minimum points, then all it neighbors are also visited in the same call of the expandCluster function.
13. All those points which are visited are marked in visited array as 1 and in the cluster label their corresponding index (= gene Id) is marked as “X”.
14. If there is a neighbor whose number of neighbor given from “regionQuery” function is less than minimum number of points, then that point is a border point. For such points, their neighbors are not included to visit in the same call of the expandCluster function.
15. Once, one call to the expandCluster function is over, all the points visited are marked or come under one cluster Id.
16. The entire procedure ends when all the points are visited.

Code Explanation:

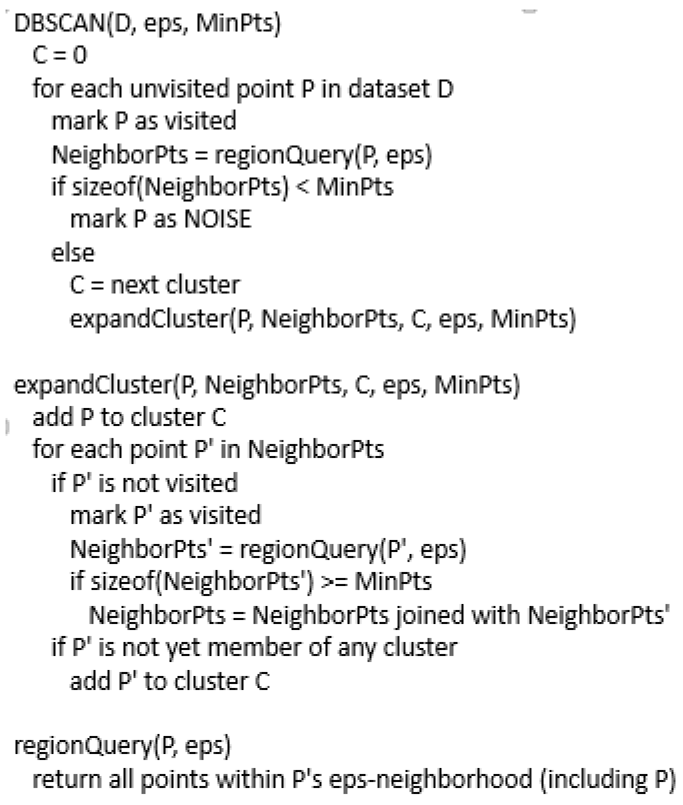
1. In program, there are 2 classes:
    - ***a.*** `DBScanAlgorithm` Class
    - ***b.*** `DBScan Class.`

To run the code an object of the DBScanAlgorithm is made. Then, a function DBScan is called with specific values of Epsilon and minimum number of points. As soon as the execution begins it gives an Array List of cluster Ids, Each Cluster Id is also an array List containing gene Id of the points within the same cluster.

The first Array List of cluster Id, in the Array List of cluster Ids is a noise cluster, always.



```
//Finding Clusters
DBScanAlgorithm dbs = new DBScanAlgorithm();
ArrayList<ArrayList<Integer>> clusters = dbs.DBScan(dataMatrix, 1.5, 2);
```



```
DBSCAN(D, eps, MinPts)
  C = 0
  for each unvisited point P in dataset D
    mark P as visited
    NeighborPts = regionQuery(P, eps)
    if sizeof(NeighborPts) < MinPts
      mark P as NOISE
    else
      C = next cluster
      expandCluster(P, NeighborPts, C, eps, MinPts)

expandCluster(P, NeighborPts, C, eps, MinPts)
  add P to cluster C
  for each point P' in NeighborPts
    if P' is not visited
      mark P' as visited
      NeighborPts' = regionQuery(P', eps)
      if sizeof(NeighborPts') >= MinPts
        NeighborPts = NeighborPts joined with NeighborPts'
    if P' is not yet member of any cluster
      add P' to cluster C

regionQuery(P, eps)
  return all points within P's eps-neighborhood (including P)
```

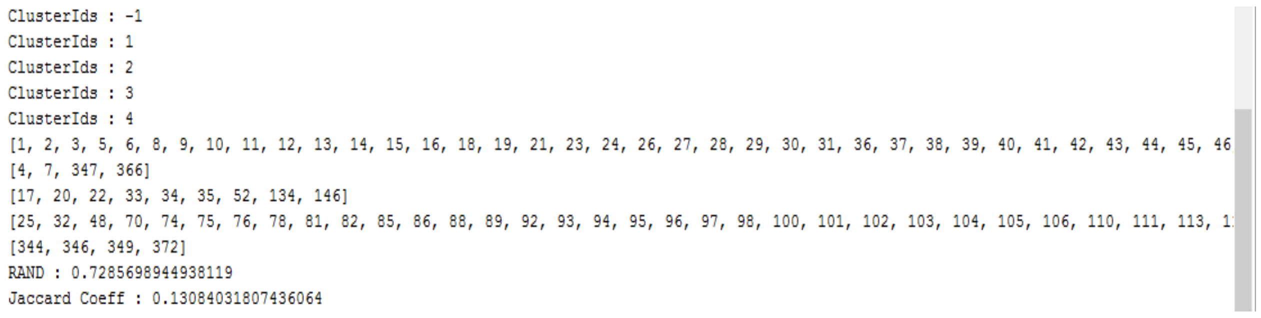
<u>Code Complexity:</u>

1. *Looking at the worst case complexity of the algorithm, it is O(n^2).*
2. *But for relatively low data sets and points it runs fairly fast.*
3. *In case of huge data sets the curve of complexity increases in a squared way.*

Result Evaluation –

- cho.txt

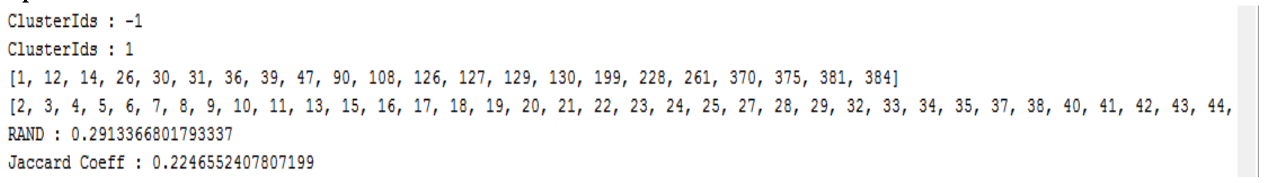
When we the algorithm for values of Epsilon = 1.0 and minimum number of points as 4, 4 clusters are formed with one noise cluster represented by -1. The order of cluster Id has one to one correspondence with the list below it. Also the Jaccard and RAND Coefficients are also specified.



```
ClusterIds : -1
ClusterIds : 1
ClusterIds : 2
ClusterIds : 3
ClusterIds : 4
[1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19, 21, 23, 24, 26, 27, 28, 29, 30, 31, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46
[4, 7, 347, 366]
[17, 20, 22, 33, 34, 35, 52, 134, 146]
[25, 32, 48, 70, 74, 75, 76, 78, 81, 82, 85, 86, 88, 89, 92, 93, 94, 95, 96, 97, 98, 100, 101, 102, 103, 104, 105, 106, 110, 111, 113, 1
[344, 346, 349, 372]
RAND : 0.7285698944938119
Jaccard Coeff : 0.13084031807436064
```

Since, the data for cho.txt is scattered, hence, Density based clustering does not perform very well to form the clusters. Also, results depend a lot upon initial parameters selected as Epsilon and Minimum number of points. Since points are scattered apart and the value of Epsilon is small, hence, it forms small clusters of close by points and the rest of the points are regarded as noise points.

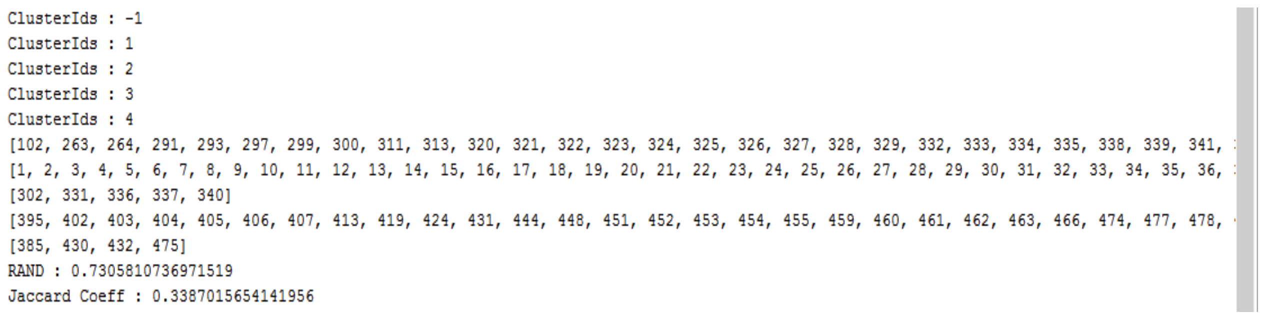
Similarly, for values of Epsilon = 2 and minimum number of points as 55, there is one cluster and one noise cluster stated as -1. Also the Jaccard Coefficients are stated below. Here we see that much better clustering performance is given if the value of Epsilon and number of points is increased. This if for the simple reason that, for scattered points, if we want to club them under clusters then they will form a cluster and come together if we increase the value of epsilon, since they are far apart.



```
ClusterIds : -1
ClusterIds : 1
[1, 12, 14, 26, 30, 31, 36, 39, 47, 90, 108, 126, 127, 129, 130, 199, 228, 261, 370, 375, 381, 384]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 32, 33, 34, 35, 37, 38, 40, 41, 42, 43, 44,
RAND : 0.2913366801793337
Jaccard Coeff : 0.2246552407807199
```

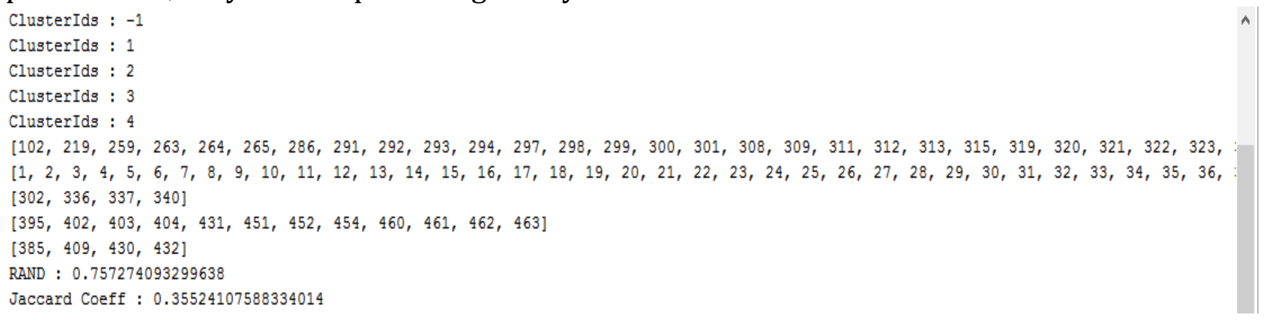
- Iyer.txt

For values of Epsilon = 1 and Minimum number of Points as 4, the Jaccard Coefficient of iyer is much higher compared to cho. This is to do with the data. In PCA visualization we will see that since the data in iyer.txt is much more compact and distribution is not all scattered hence, the clustering algorithm does better than it does in cho.



```
ClusterIds : -1
ClusterIds : 1
ClusterIds : 2
ClusterIds : 3
ClusterIds : 4
[102, 263, 264, 291, 293, 297, 299, 300, 311, 313, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 332, 333, 334, 335, 338, 339, 341,
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
[302, 331, 336, 337, 340]
[395, 402, 403, 404, 405, 406, 407, 413, 419, 424, 431, 444, 448, 451, 452, 453, 454, 455, 459, 460, 461, 462, 463, 466, 474, 477, 478,
[385, 430, 432, 475]
RAND : 0.7305810736971519
Jaccard Coeff : 0.3387015654141956
```

For values of Epsilon = 0.8, and Minimum number of points as 3, the Jaccard coefficient slightly increases. Hence, the clustering algorithm performs well in case of compact data. Also, it is worth noticing that it also efficiently classifies the noise points since, they are not pertaning to any cluster.

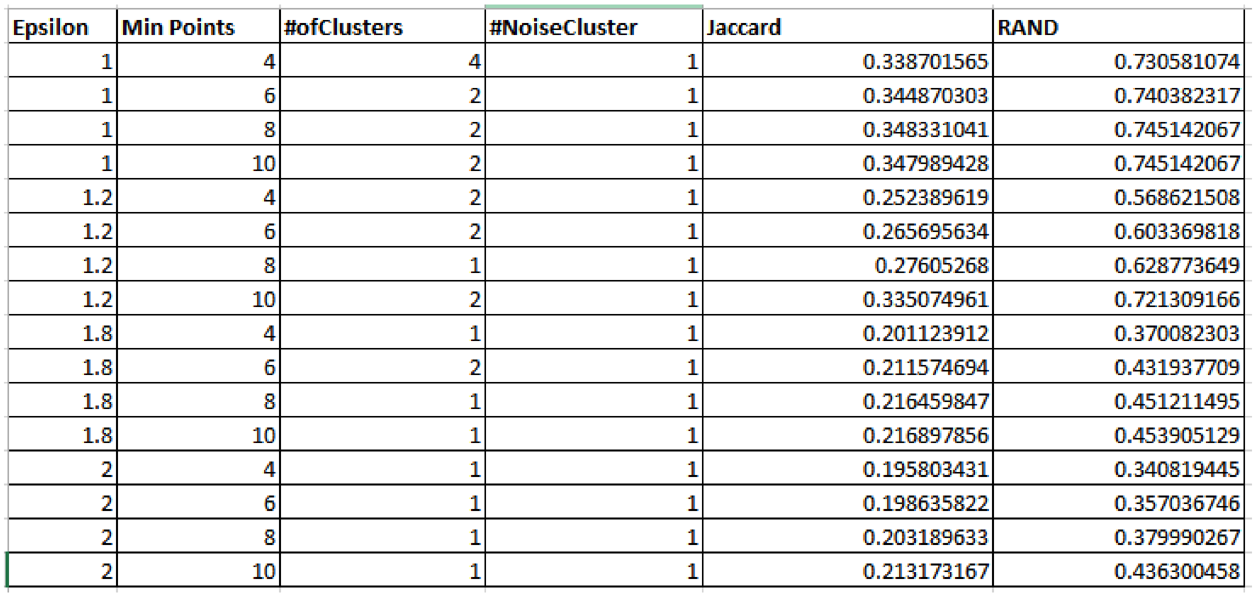


```
ClusterIds : -1
ClusterIds : 1
ClusterIds : 2
ClusterIds : 3
ClusterIds : 4
[102, 219, 259, 263, 264, 265, 286, 291, 292, 293, 294, 297, 298, 299, 300, 301, 308, 309, 311, 312, 313, 315, 319, 320, 321, 322, 323,
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
[302, 336, 337, 340]
[395, 402, 403, 404, 431, 451, 452, 454, 460, 461, 462, 463]
[385, 409, 430, 432]
RAND : 0.757274093299638
Jaccard Coeff : 0.35524107588334014
```

One thing clear about the Density Based clustering is, it is not effected by the outliers. In fact, it is quite efficient in case of outlier recognition, since it chases the data. Hence, the impact of outliers while clustering is nullified. Also, among all the three algorithms i.e. K-Means, Hierarchal and Density Based, it is the only algorithm that determines the outliers and is not susceptible to the impact of outliers.

Experiment Values –

- **Iyer.txt**



Epsilon	Min Points	#ofClusters	#NoiseCluster	Jaccard	RAND
1	4	4	1	0.338701565	0.730581074
1	6	2	1	0.344870303	0.740382317
1	8	2	1	0.348331041	0.745142067
1	10	2	1	0.347989428	0.745142067
1.2	4	2	1	0.252389619	0.568621508
1.2	6	2	1	0.265695634	0.603369818
1.2	8	1	1	0.27605268	0.628773649
1.2	10	2	1	0.335074961	0.721309166
1.8	4	1	1	0.201123912	0.370082303
1.8	6	2	1	0.211574694	0.431937709
1.8	8	1	1	0.216459847	0.451211495
1.8	10	1	1	0.216897856	0.453905129
2	4	1	1	0.195803431	0.340819445
2	6	1	1	0.198635822	0.357036746
2	8	1	1	0.203189633	0.379990267
2	10	1	1	0.213173167	0.436300458

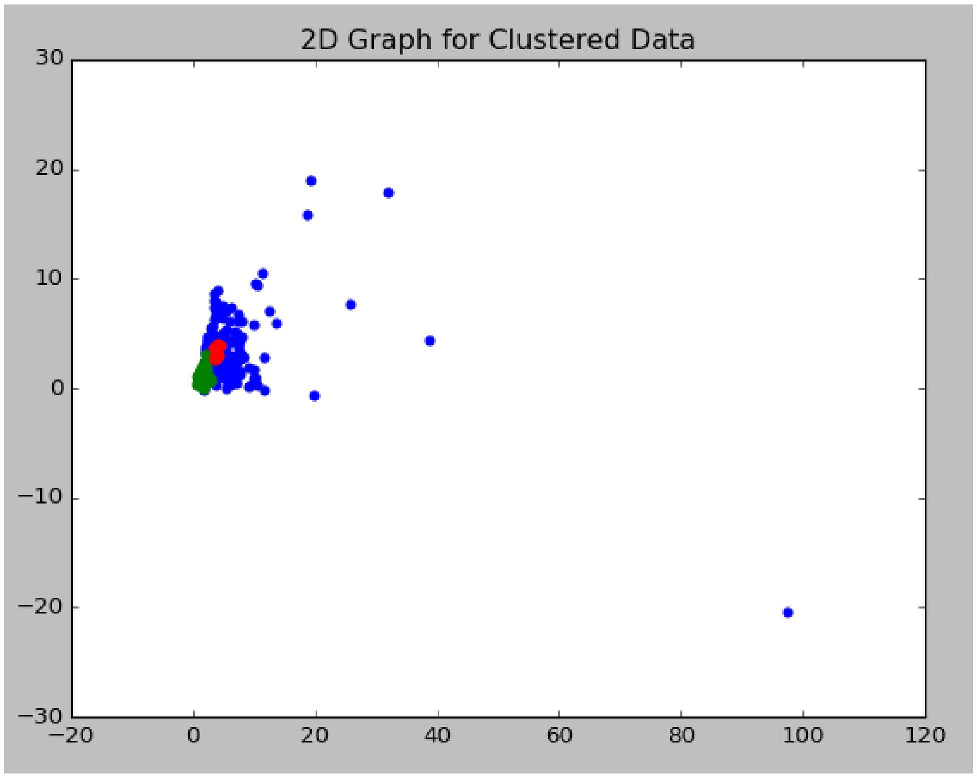
***RESULT VISUALIZATION**: If we compare the evaluation results of density based clustering for 2 data sets, then it is clear that density based clustering performs better on iyer data set. This is because of the structure of dataset being more compact. The highest values of Jaccard occur at Epsilon= 1 and min points= 8.*
Also, it is successful in finding the outliers.

<u>PCA For iyer.txt</u>

- Epsilon = 1
- Min Points = 8
- 2 Clusters and 1 Noise Cluster 0.0

2-D (Algorithm output)

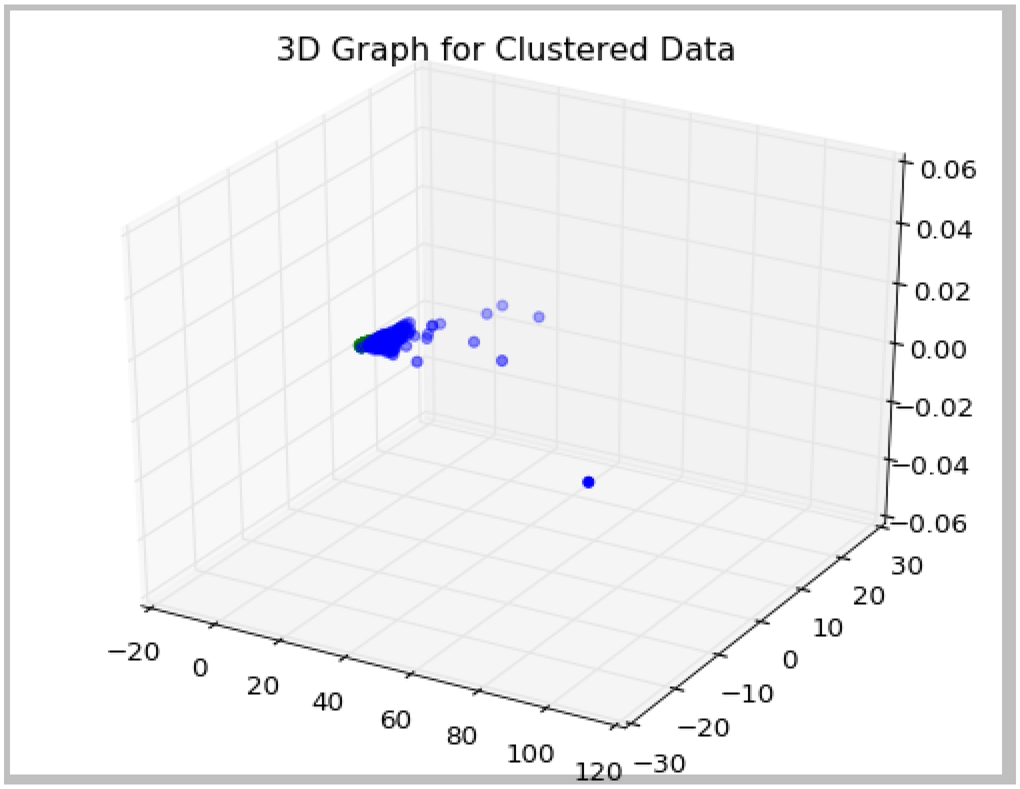
```
Running for clustered data set 2D
 : color : b : clusterID : 0.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
```



NOTE: ClusterID: 0.0 corresponds to Noise Cluster

3-D (Algorithm Output)

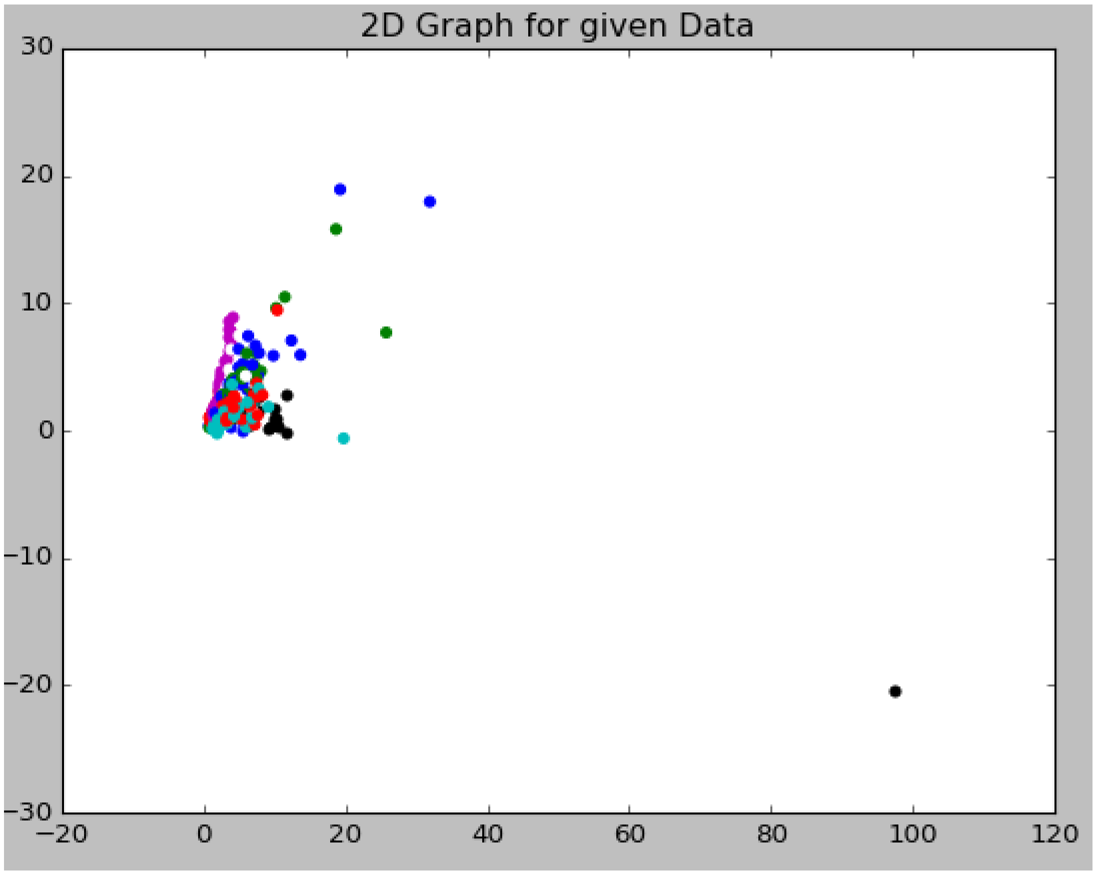
```
Running for clustered data set 3D
 : color : b : clusterID : 0.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
```



NOTE: ClusterID : 0.0 corresponds to noise cluster

2-D (Given Data)

```
Running for given data set 2D
 : color : b : clusterID : -1.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
 : color : c : clusterID : 3.0
 : color : m : clusterID : 4.0
 : color : y : clusterID : 5.0
 : color : k : clusterID : 6.0
 : color : w : clusterID : 7.0
 : color : bg : clusterID : 8.0
 : color : rw : clusterID : 9.0
 : color : cr : clusterID : 10.0
```

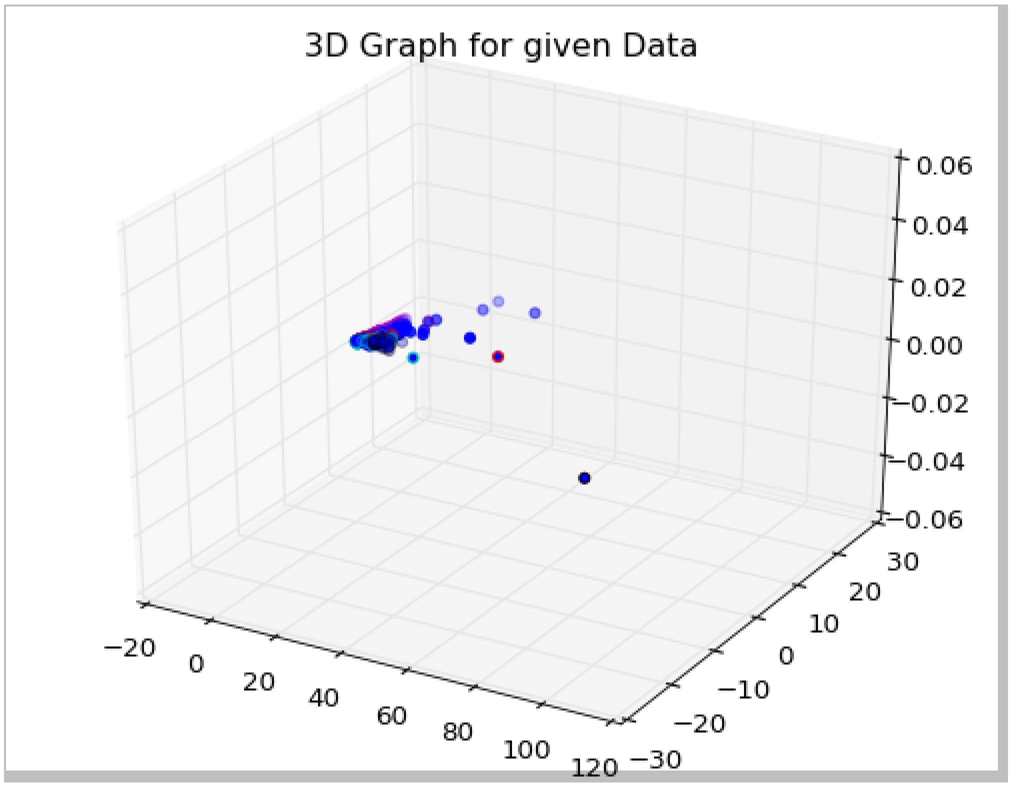


NOTE: ClusterID : -1 corresponds to noise cluster

3-D (Given Data)

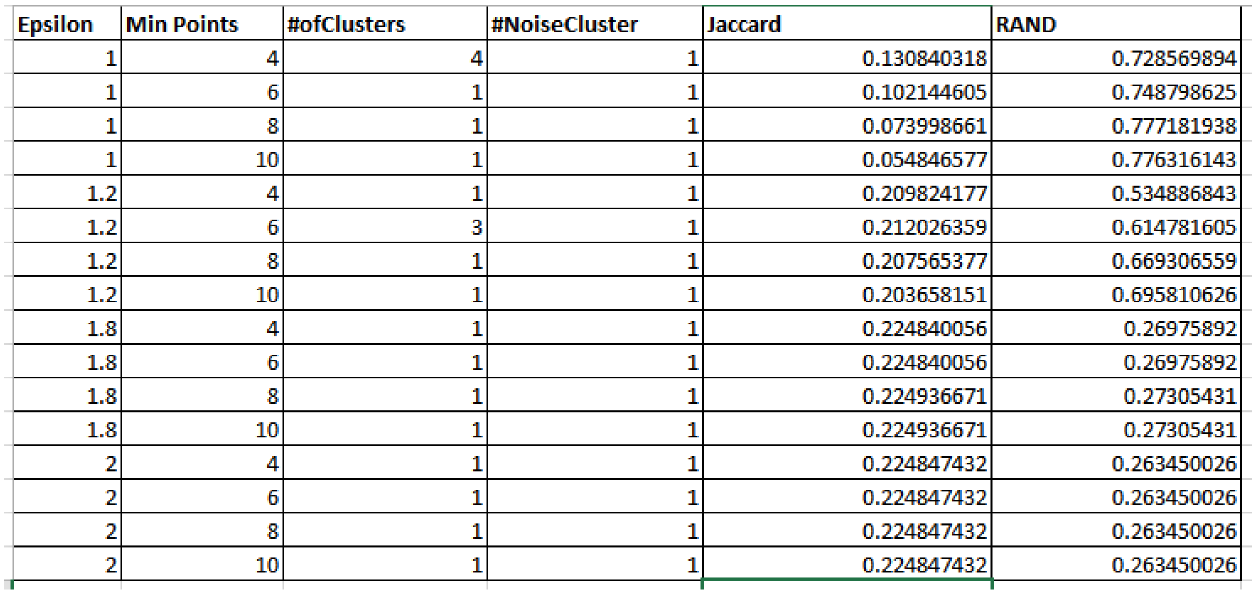
```
Running for given data set 3D
 : color : b : clusterID : -1.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
 : color : c : clusterID : 3.0
 : color : m : clusterID : 4.0
 : color : y : clusterID : 5.0
 : color : k : clusterID : 6.0
 : color : w : clusterID : 7.0
 : color : bg : clusterID : 8.0
 : color : rw : clusterID : 9.0
 : color : cr : clusterID : 10.0
```

NOTE: ClusterID: -1 corresponds to noise cluster



- **Cho.txt**

RESULT VISUALIZATION*: Due to the scattered way of data set cluster formation based on density is difficult. Highest values of Jaccard occur at Epsilon 1.8 and Minimum Points 8 and 10, but for these points RAND Index is small. Also, at Epsilon 1.2 and Minimum points 6 the value of algorithm is quite good giving 3 clusters.*
A high value of Jaccard means that the performance of the clustering algorithm is closer to the ground truth. Similarly for RAND Index as well the same holds true.
If we look at the results table, it would be clear that the value of the Jaccard coefficient is not the highest for this setting, but on close inspection we can see that RAND is very low for lower values of the table. Since for Epsilon 1.2 and Minimum points 6, both Jaccard and RAND are considerably high hence, we choose this value for visualization.



Epsilon	Min Points	#ofClusters	#NoiseCluster	Jaccard	RAND
1	4	4	1	0.130840318	0.728569894
1	6	1	1	0.102144605	0.748798625
1	8	1	1	0.073998661	0.777181938
1	10	1	1	0.054846577	0.776316143
1.2	4	1	1	0.209824177	0.534886843
1.2	6	3	1	0.212026359	0.614781605
1.2	8	1	1	0.207565377	0.669306559
1.2	10	1	1	0.203658151	0.695810626
1.8	4	1	1	0.224840056	0.26975892
1.8	6	1	1	0.224840056	0.26975892
1.8	8	1	1	0.224936671	0.27305431
1.8	10	1	1	0.224936671	0.27305431
2	4	1	1	0.224847432	0.263450026
2	6	1	1	0.224847432	0.263450026
2	8	1	1	0.224847432	0.263450026
2	10	1	1	0.224847432	0.263450026

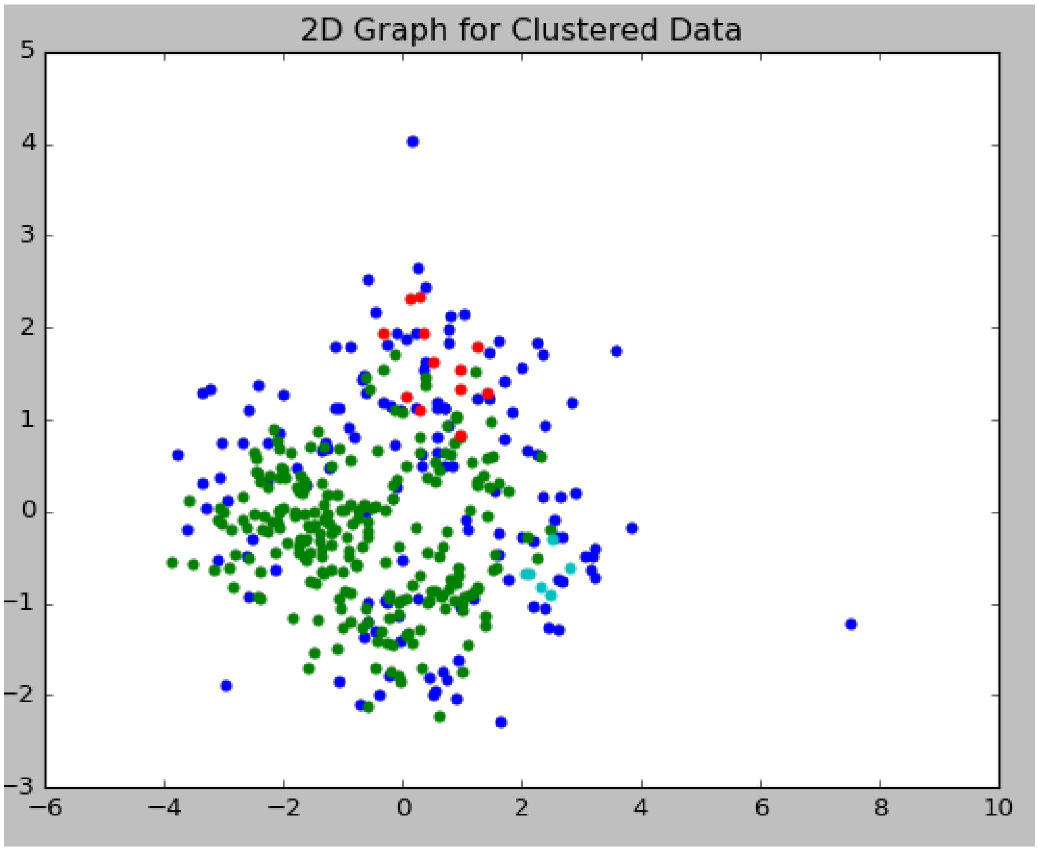
PCA for cho.txt

- Epsilon = 1.2
- Minimum Points = 6
- Clusters = 3 clusters and one noise cluster 0.0

2-D (Algorithm Output)

```
Running for clustered data set 2D
 : color : b : clusterID : 0.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
 : color : c : clusterID : 3.0
```

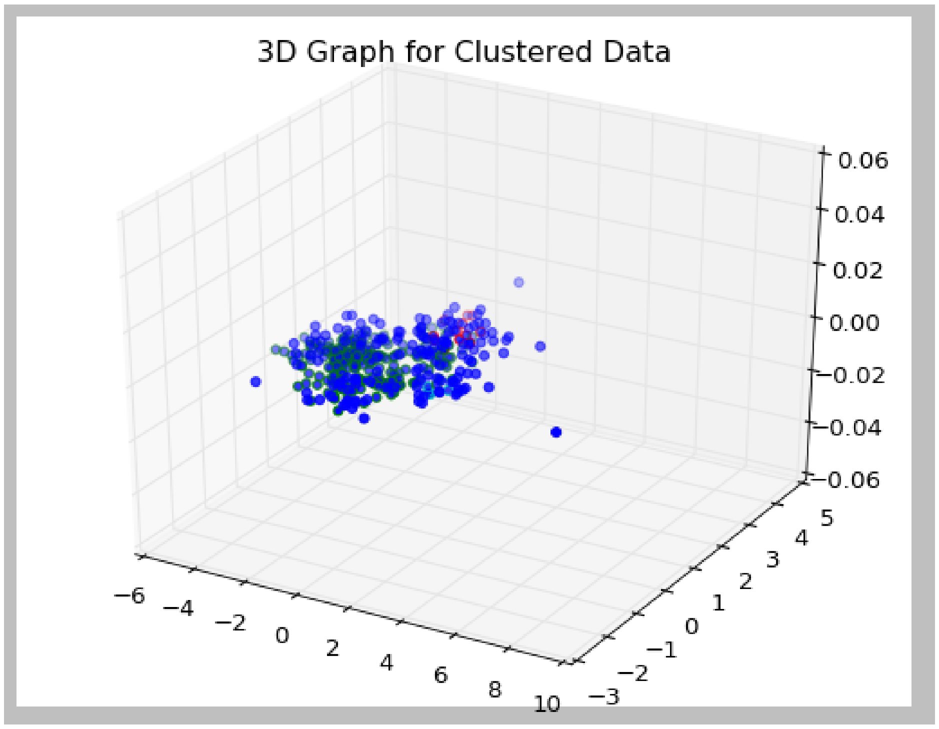
NOTE: A ClusterID 0.0 is the noise cluster



3-D (Algorithm Output)

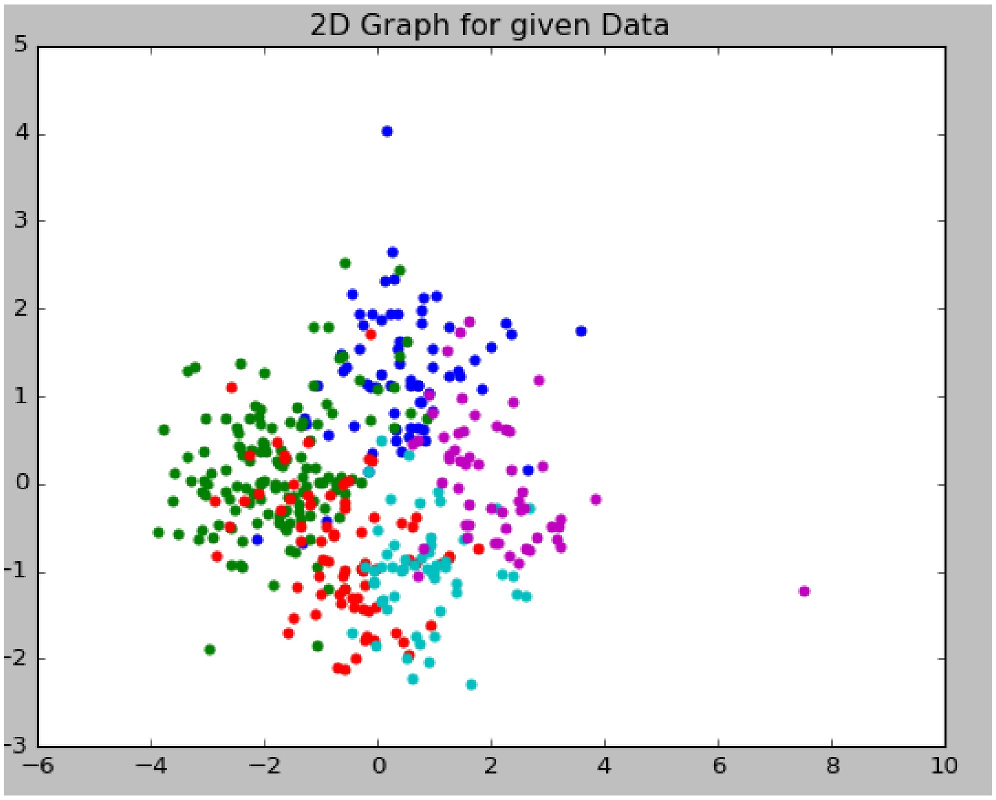
```
Running for clustered data set 3D
 : color : b : clusterID : 0.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
 : color : c : clusterID : 3.0
```

NOTE: ClusterID: 0.0 corresponds to noise cluster



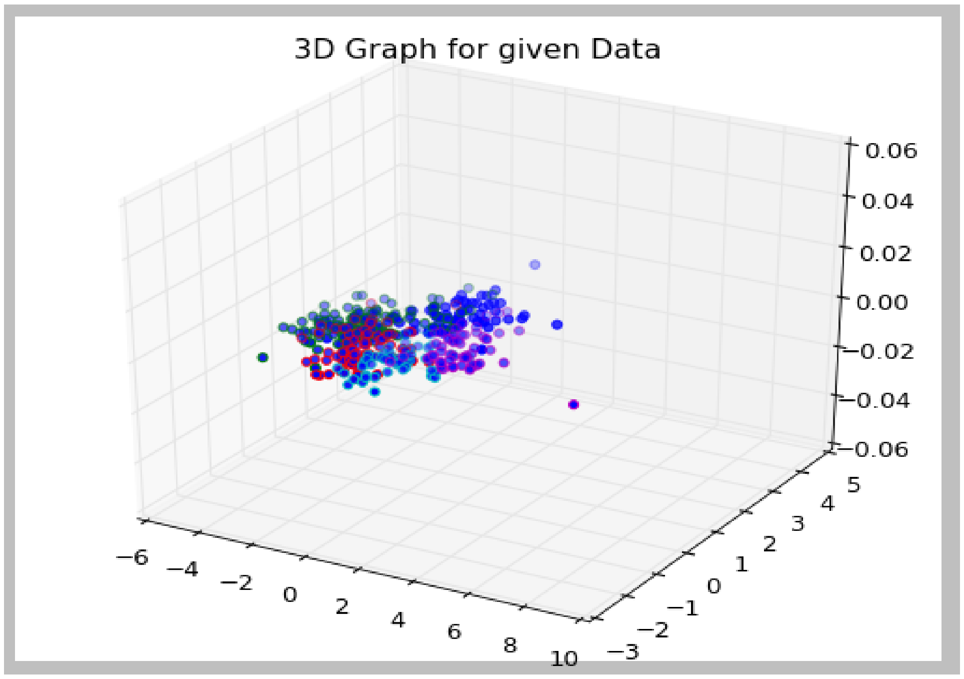
2-D (Given Data)

```
Running for given data set 2D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0
```



3-D (Given Data)

```
Running for given data set 3D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0
```



Pros and Cons of the algorithm –

PROS

- Density based clustering can form clusters of different shapes. Unlike K-means where cluster are more or less of circular shape, density based clustering tries to follow the density of data. In our above visualization in iyer data set, we can see the noise cluster that gets formed and how density based clustering forms the cluster shape.
- Density Based Clustering, does not chase the outliers at all. It is clear from the above two data sets that density based clustering is capable of nullifying the effects of outliers.

CONS

- Cannot handle varying densities. This is pretty much evident from the PCA and visualization of the data sets. In cho.txt when the data is scattered, the algorithm struggles to find a good cluster.
- Sensitive to parameters—hard to determine the correct set of parameters. This is also one issue with Density Based Clustering as initially we are not aware of the average distance between the data points. Hence, it is pretty hard to find a good value of Epsilon and minimum points.

Final Analysis:

From the above visualization, it is pretty much clear that for iyer.txt Density based clustering is better compared to cho.txt where density of data is variable. Hence, for data sets like cho.txt density based clustering is not recommended.

4) **Map Reduce K-Means Clustering** –

Description –

K-Means clustering is a partition based clustering algorithm. In this algorithm we have to specify initially, the number of clusters, number of iterations and need to specify initial centroid dimensions corresponding to the given number of clusters. For this, we chose initial centroids randomly from the dataset since choosing these datasets will ensure that we achieve convergence in lesser number of iterations. Choice of centroids is extremely essential since it governs the number of iterations at which the centroid dimensions stop changing and the clusters formed till this point doesn't change further and are the final clusters. Henceforth, we believe that choosing random centroids from the dataset is a good strategy choosing a random point unknowingly as a centroid might produce bad clusters and may take greater number of iterations to converge. Subsequent to this, in the initial pass, we calculate the Euclidian distance between each data point and each cluster centroid and assign the data point to the cluster with closest centroid in terms of Euclidean distance. Finally, we perform an iteration step at which we repeatedly recalculate centroid dimensions for each centroid based on the dimensions of the data sets assigned to that cluster and assign clusters to the data sets based on new centroid dimensions using Euclidean distance. This iteration step runs as many times as are the number of iterations specified initially in the k means configuration parameters. Usually, the number of iterations are chosen so as to ensure that k means clustering achieves convergence before the specified number of iterations.

The algorithm for K Means clustering is almost the same as in the first section except the fact that in this part, we need to achieve the same using single node Hadoop cluster based on Map Reduce parallel implementation. For achieving this, initially we set up Hadoop on a virtual host namely Oracle VM Virtual Box based on Ubuntu. Setup mainly comprises of installing java development kit and Hadoop along with specifying the correct paths (as mentioned below) and few configuration changes in core-site.xml file and hdfs-site.xml file. The following section describes the steps we followed for Hadoop setup and for running Map Reduce code on Virtual Box installed in our machine followed by algorithm implementation.

Set Up and Code Run on Virtual Box –

- export HOME=/home/hadoop
- export HADOOP_HOME=/home/hadoop/hadoop
- export PATH=${PATH}:$HADOOP_HOME/sbin
- export PATH=${PATH}:$HADOOP_HOME/bin
- export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk-i386
- export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
- export PATH=${PATH}:/usr/lib/jvm/java-1.6.0-openjdk-i386/bin
- echo $JAVA_HOME
- echo $HADOOP_CLASSPATH
- echo $HOME
- echo $HADOOP_HOME
- echo $PATH
- cd hadoop/
- bin/hadoop com.sun.tools.javac.Main ../WordExample/KMeansClustering.java
- cd ..
- cd WordExample/
- jar cf kmc.jar KMeansClustering *.class
- start-hadoop.sh
- jps
- hdfs dfs –put $HOME/data input
- hadoop jar kmc.jar KMeansClustering input output
- hdfs dfs –get output ../data
- hdfs dfs –get centroids ../data

In brief, the steps basically comprise of setting up initial paths, compiling KMeansClustering.java class, making a jar file in the same folder where all the compiled class files are located, starting up Hadoop through Hadoop start up script, placing the input file in a specific HDFS location and eventually running the map reduce code present in jar file based on the input and output arguments which generate the final output file at a specific HDFS location. Finally, we extract the desired output files from HDFS into our local filesystem.

Algorithm Implementation –

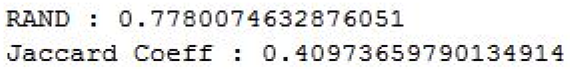
1. Initially we specify the number of clusters and number of iterations that we need for running Map Reduce k means Clustering code using static variables. We adopted this strategy since it removes unnecessary hardcoding and one can easily run the algorithm for differing number of clusters and iterations by just changing values of the initial static variables

2. For the initial pass of K Means Clustering, where we choose initial centroids based on the number of clusters and assign clusters to data sets based on the initial centroids, we chose as many centroids as are the number of clusters randomly from the data set using a single dedicated Map Reduce job which comprises of InitMapper.java and InitReducer.java. Here, mapper basically reads the input gene sample file and outputs all the gene expressions with the same key. This strategy is adopted so that we accumulate the list of all the gene expressions at the reducer and choose initial centroids randomly from all the bunch of gene expressions. This was done since we understood that mapper reads one line at a time from the input file. But, in order to obtain the list of all the gene expressions, we needed to push all the gene expressions using the same key in mapper stage and obtaining all the values with the same single key in the reducer stage so as to choose random initial centroids for starting up k means clustering. One thing to note here is that the initial pass of means of k means clustering is still not over yet since it also includes assigning clusters to the data sets based on initial centroids. This step is performed in the first mapper stage of the next bunch of iterations. For this reason, we run the map reduce stage one time more than the number of iterations specified initially to incorporate this initial pass step in the very first mapper stage of the number of the subsequent number of map reduce stages running for the desired number of iterations. The output file generated in InitReducer step contains all the gene sample expressions along with their assigned cluster ids. However, this file will be of use only after the last mapper reducer stage where the algorithm will terminate once it exceeds the number of iterations. Besides this, there will also be a centroids file generated at this step which will be used up in the next mapper stage to specify the initial centroids for the first iteration of K Means Clustering algorithm. We would also like to highlight that this centroids file is a kind of temporary file which we are generating in each and every reducer stage and reading out in each and every mapper stage since it contains information about centroid dimensions generated from the previous iteration of K Means for the mapper stage in the next iteration of K Means
3. Subsequent to the previous job, we implemented a series of map reduce jobs based on a single configuration which are designed in a manner such that each and every map reduce stage serves as one single iteration step of K

Means algorithm. In every map reduce stage, input file is the original gene sample data file to the mapper which outputs closest cluster id as key and the corresponding single gene expression information(gene id and its dimensions) as value based on the minimum Euclidean distance between the gene dimensions and one out of all the centroid dimensions. Reducer accumulates all the gene samples for a single cluster id as key, calculates the new centroid dimensions of that particular cluster based on the data points belonging to that cluster and outputs pairs of cluster id as key and its corresponding gene expressions information as value. Besides this output file, we also dump the all the new centroid dimensions in a single file centroid.txt which will be used by the mapper in the next map reduce stage for evaluating the closest centroids to the data points, as mentioned briefly in the previous step. In this step, we implemented the mapper code inside Iteration Mapper and reducer code inside Iteration Reducer. This code is generic for all the iteration of K Means algorithm and will be executed whenever number of jobs equal to the number of iterations are run subsequent to the initial pass job comprising of Init Mapper and Init Reducer. Again, we are not using the output file generated from Iteration Reducer in the next mapper stage and for every mapper, input file is the given gene sample data file and the centroid file generated from the previous reducer stage. However, when the iterations exceed the number of iterations, the output file generated in the final reducer stage will display the final information related to all cluster ids along with their assigned gene samples.

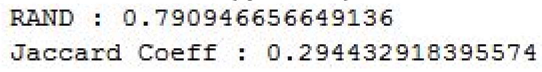
Experiment Values –

External Index (cho.txt), for k=5, n=20



```
RAND : 0.7780074632876051
Jaccard Coeff : 0.40973659790134914
```

External Index (iyer.txt), for k=5, n=20



```
RAND : 0.790946656649136
Jaccard Coeff : 0.294432918395574
```

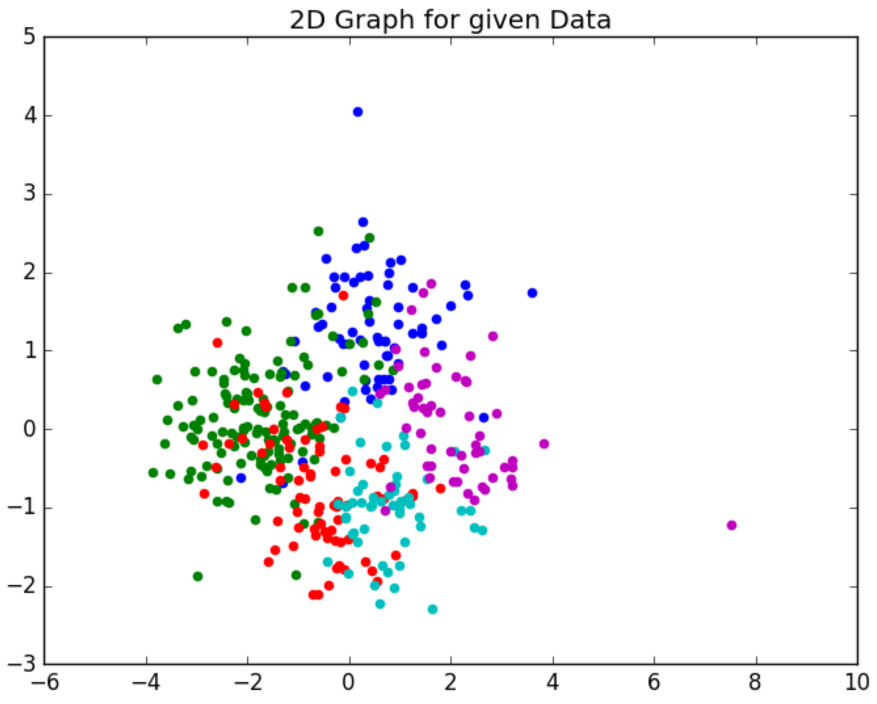
Result Evaluation/Visualization –

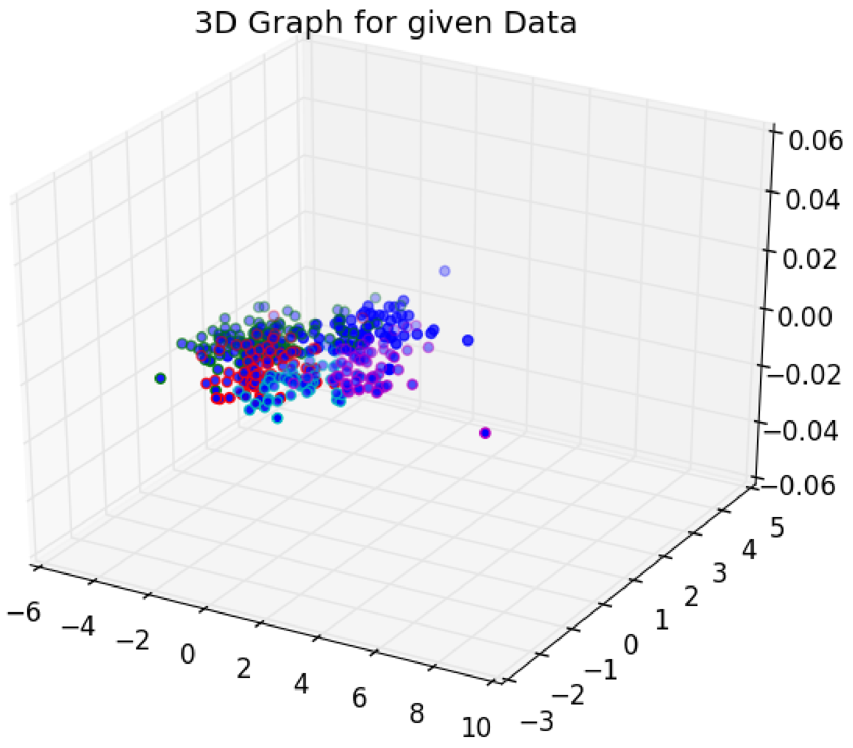
Result Analysis

- For cho.txt, since the data is not overlapping and is a bit clustered. So, it becomes easy for K Means algorithm to find clusters. However, due to the impact of outliers, some of clusters can be seen as more inclined towards those outlier points. Also, in the worst case, it formed a cluster comprising of only a single outlier point as can be seen in the PCA visualization below

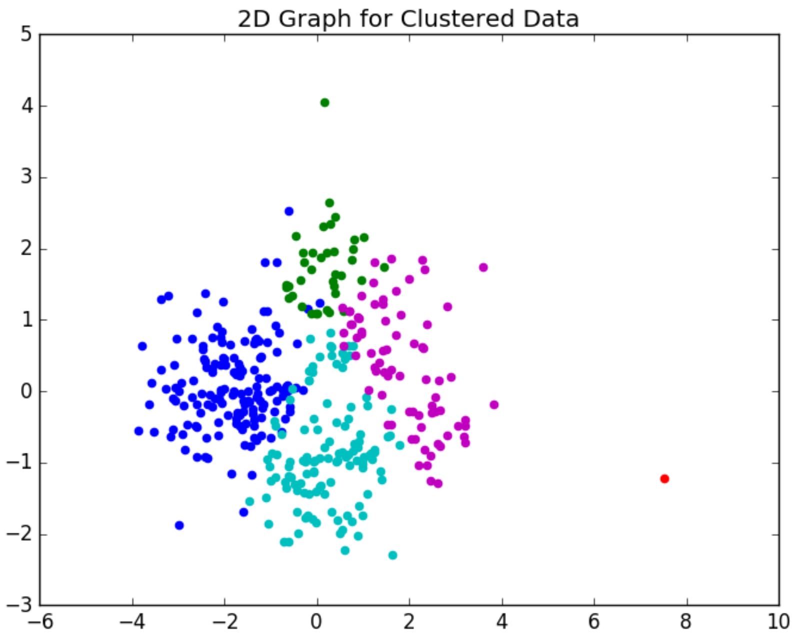
- Also, if one is not aware of the data set, it is hard to predict the number of clusters that should be formed so as the clustering algorithm can resemble ground truth classification. In fact, if the value of k is not appropriately selected, it can lead to clusters with irregular shapes and densities, as can be seen in the case of PCA visualization for iyer.txt
- **Recommendation** – From PCA visualizations and based on the nature of dataset, we can deduce that K Means is suitable for data spread as in cho.txt in which case the data forms cluster of significant sizes and good shapes and there is a clear demarcation between data points. But, in case of iyer.txt, it can be seen that presence of too many outliers leads to formation of clusters with very small, very large (irregular) sizes and there is no clear separation between different clusters. Also, in case of iyer.txt, most of the clusters are formed very close to each other wherein ideally only a single cluster for such a space comprising of close data points should have been formed. Besides this, based on the values obtained from Jaccard coefficient, we can corroborate the above analysis by saying that clustered data set in case of cho.txt resembles more to ground truth as compared to clustered data set in case of iyer.txt.

PCA For cho.txt

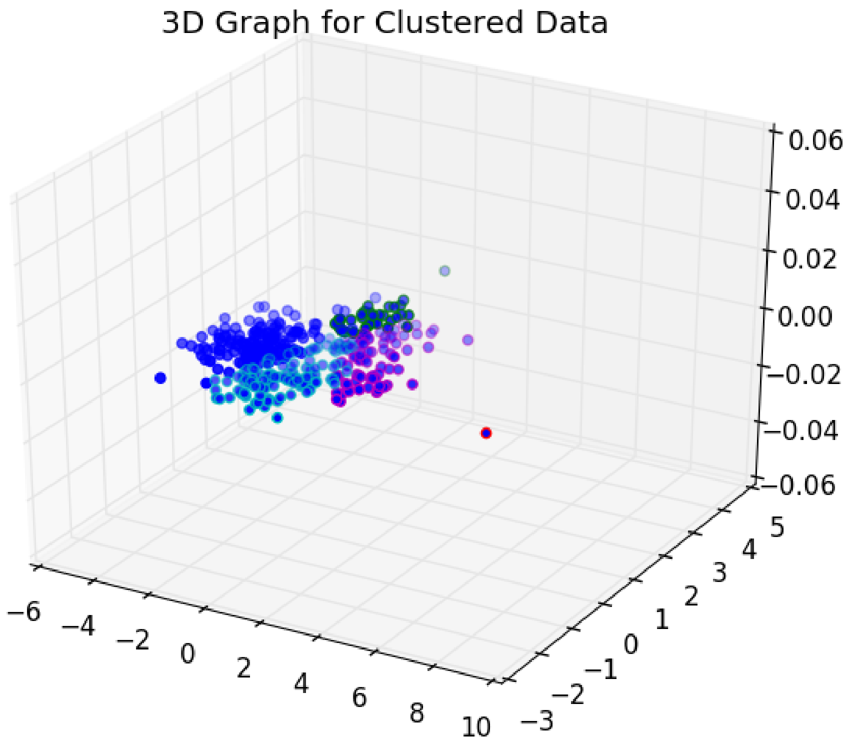




```
Running for given data set 3D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0

Running for given data set 2D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0

Running for clustered data set 3D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0

Running for clustered data set 2D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0
```

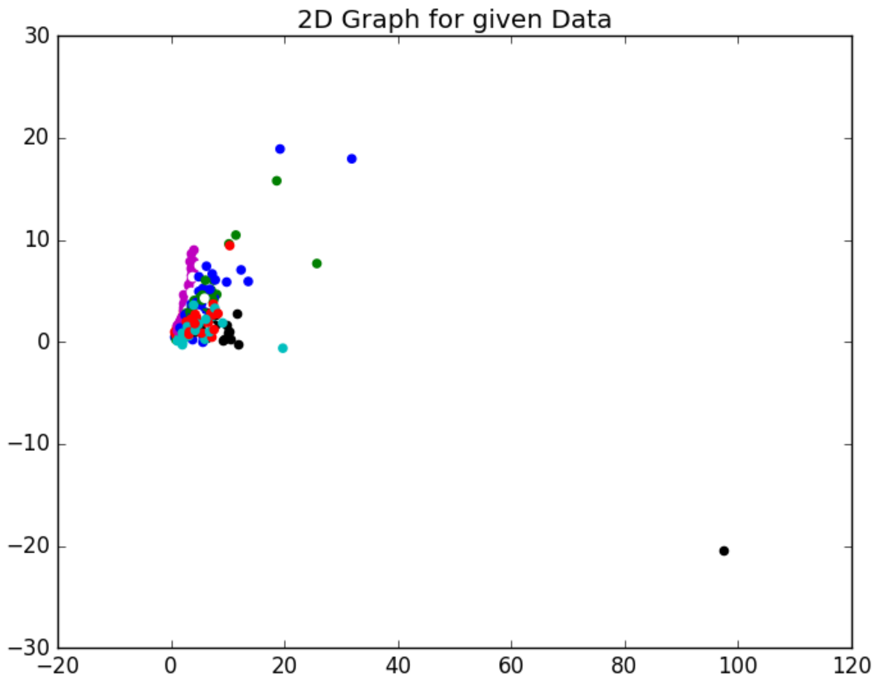


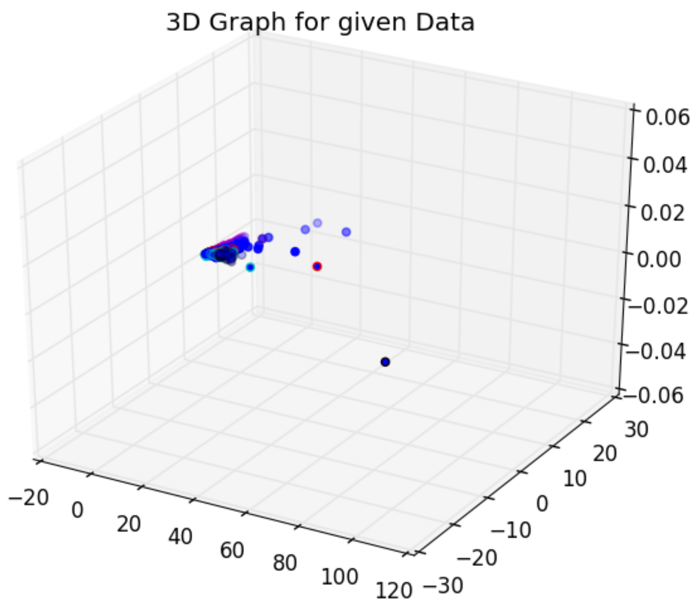
2D Graph for Clustered Data
5
4
3
2
1
0
−1
−2
−3
−6
−4
−2
0
2
4
6
8
10

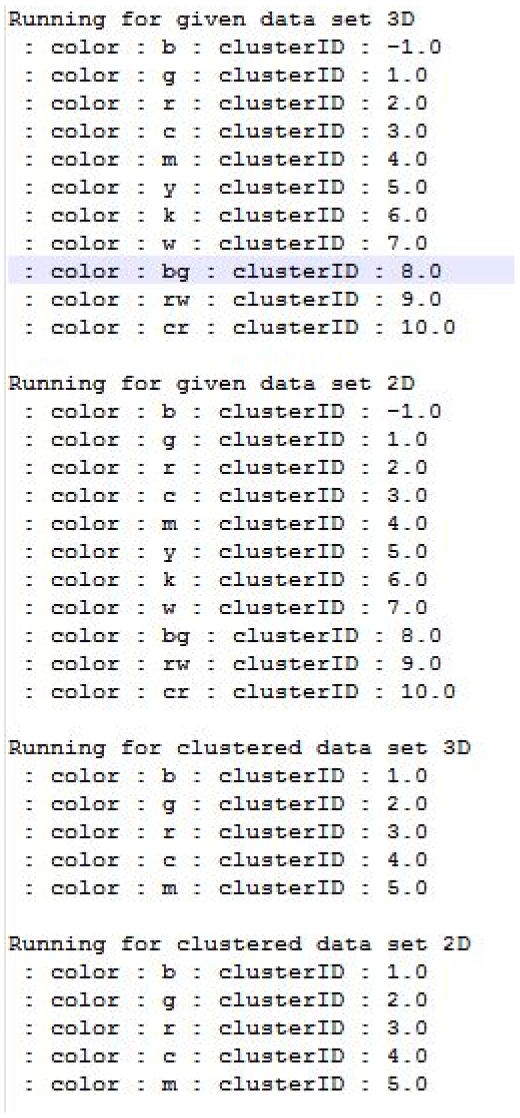


3D Graph for Clustered Data
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−6
−4
−2
0
2
4
6
8
10
−3
−2
−1
0
1
2
3
4
5

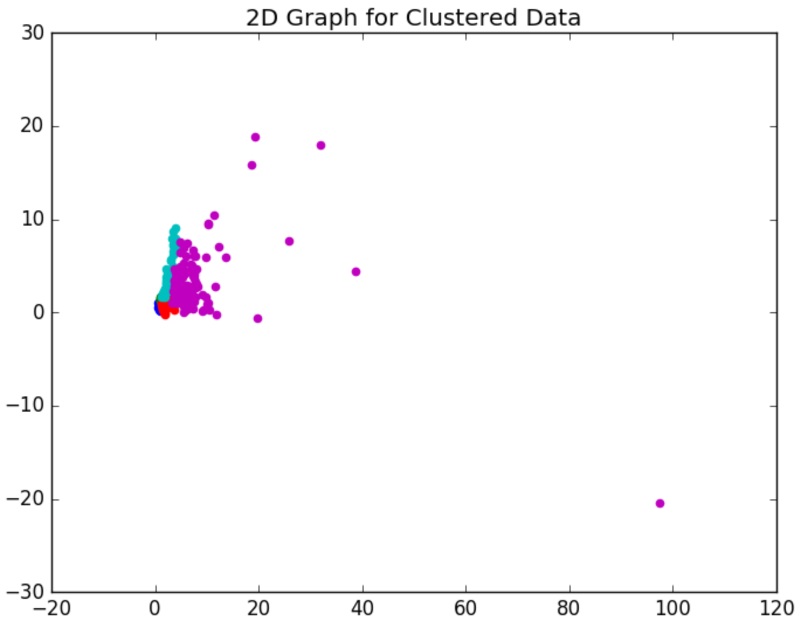
PCA for iyer.txt

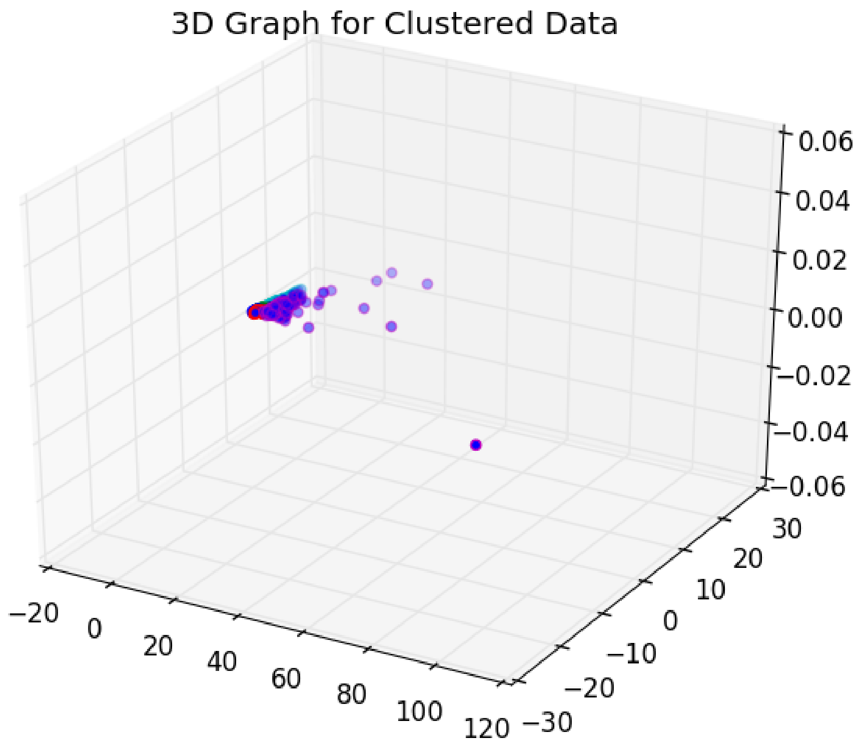






```
Running for given data set 3D
 : color : b : clusterID : -1.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
 : color : c : clusterID : 3.0
 : color : m : clusterID : 4.0
 : color : y : clusterID : 5.0
 : color : k : clusterID : 6.0
 : color : w : clusterID : 7.0
 : color : bg : clusterID : 8.0
 : color : rw : clusterID : 9.0
 : color : cr : clusterID : 10.0

Running for given data set 2D
 : color : b : clusterID : -1.0
 : color : g : clusterID : 1.0
 : color : r : clusterID : 2.0
 : color : c : clusterID : 3.0
 : color : m : clusterID : 4.0
 : color : y : clusterID : 5.0
 : color : k : clusterID : 6.0
 : color : w : clusterID : 7.0
 : color : bg : clusterID : 8.0
 : color : rw : clusterID : 9.0
 : color : cr : clusterID : 10.0

Running for clustered data set 3D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0

Running for clustered data set 2D
 : color : b : clusterID : 1.0
 : color : g : clusterID : 2.0
 : color : r : clusterID : 3.0
 : color : c : clusterID : 4.0
 : color : m : clusterID : 5.0
```


2D Graph for Clustered Data
30
20
10
0
−10
−20
−30
−20
0
20
40
60
80
100
120


3D Graph for Clustered Data
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−20
0
20
40
60
80
100
120
30
20
10
0
−10
−20
−30

Pros and Cons of the algorithm –

Comparison with Non-Parallel K Means –

- In case of parallel K Means, we can perform clustering with lesser and cleaner code as compared to non-parallel K Means
- Parallel K Means is more suitable for large sized datasets. However, in case of not so large datasets as provided in this project, parallel K Means took more time to run as compared to non-parallel K Means
- Non Parallel K Means assumes that all the data has to reside in the computer's memory at the same time and while running, it loads the data into the main memory at once. Henceforth, in case of large datasets there can arise a problem related to shortage of memory. Such problems do not arise in case of running Map Reduce since it parallelizes the computation automatically by dividing the input data into small chunks of many files
- In case of Map Reduce, the computation is done only in two functions namely, mapper and reducer unlike lots of different functions used the in the sequential non parallel K Means. Also, underlying runtime system in Hadoop, facilitates automatic parallelization and computation across large clusters of machines. In our case, all the cluster centroids for a cluster id were computed in a parallel fashion which made the computation time efficient.
- Parallel K Means based on Map Reduce and run using Hadoop is fault tolerant which can be a point of concern when dealing with very large datasets in the future
- In our case, shuffle and sort phase of Hadoop grouped the gene sample data for a cluster id together by itself for providing input to the reducer in the next stage reducing significant manual computation

Improvements – Map Reduce K Means

Final Comparison of Clustering Algorithms

- For cho.txt, based on the external index, it can be deduced that K Means is the best algorithm for such datasets.
- For parallel and non-parallel K Means, we need to provide the number of clusters, number of iterations and initial centroids as a parameter which is usually difficult to predict beforehand whether it's suitable for the given dataset. However, in case of other clustering algorithms, we don't need to have any prior knowledge about any initial parameters required for clustering
- For iyer.txt, where there is a significant impact of outliers and the data is dense, unlike cho.txt, for such a case density based clustering algorithm better than other

clustering algorithms which require clear separation of data points. This is visible from the external indexes calculated above where density based clustering gave the best performance

- Another advantage of density based clustering over K Means is that in case of density based algorithm, irregular shapes of data set can also be clustered because density based does not always tend to form a circle
- Parallel K means algorithm is usually slower for small data sets since each map reduce stage takes significant amount of time to perform a single iteration and there are multiple iterations in case of Parallel K Means. It is only suitable for very sized data sets