Project 2: Clustering Algorithms

CSE 601: Data Mining and Bioinformatics

AVIJEET MISHRA
(AVIJEETM@BUFFALO.EDU)
UB#:50169242

PRITHVI GOLLU INDRAKUMAR
(PGOLLUIN@BUFFALO.EDU)
UB#:50169089

DEPT OF COMPUTER SCIENCE,
UNIVERSITY AT BUFFALO

Contents

Contents

Introduction

Clustering is a process of grouping similar set of objects such that objects in the same cluster belong are more similar than with objects of other cluster. It can be used in image processing, pattern recognition, spatial data analysis, etc.. as a preprocessing. Clustering algorithms can be implemented in the many ways such as Partitional, Hierarchical, Density-based, Mixture model and Spectral methods. We have implemented Partitional [Kmeans], Hierarchical, Density-based and Kmeans on MapReduce.

The given 2 datasets “cho” and “iyer” represent Genes with their 16 attribute values which are the genes experiment conditions and their clustering groundtruth values. In the above mentioned algorithms, we are grouping similar genes into the same cluster based on their attribute values. The main aim here is to identify co-expressed genes and coherent expression patterns in these 2 data sets.

Implementation

The first 3 algorithms are implemented in C# using Microsoft Visual Studio as a platform. KMeans MapReduce is implemented in Java on Oracle VM VirtualBox. The data is read from the Excel files of the datasets. In all the 3 algorithms, the similarity between 2 genes is calculated using Euclidean distance which is as follows:

$$dist(X,Y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

Here x_n and y_n are the attribute values. This formula gives the dissimilarity value "d" between Gene X and Gene Y. In our implementation, we have taken similarity value "s", where s is as shown below, since the attribute values are continuous.

$$s = \frac{1}{1+d}$$

KMeans Clustering

KMeans algorithm is a partitional clustering algorithm i.e. it divides the objects, here genes into non-overlapping clusters such that each gene is in exactly one cluster. K-means is a simple learning algorithm that follows a simple and easy way to classify a given data set through a certain specified "k" number of clusters.

We specify k initial centroids and then assign each gene or data object to 1 centroid based on its similarity to the centroids. Once the clusters are formed, we calculate the actual centroids of these clusters and reassign the data objects. This is repeated till the values of the centroids don't change.

In our implementation, the values of number of centroids[k], initial centroids, Number of iterations and the input data file are all entered by the user dynamically. The function "private void kmeans(double[,] cluster, int itr)" has the functionality of the kmeans algorithm. This function is called recursively till the centroids don't change. In each iteration the centroid data is sent as a parameter in variable "cluster", whereas the gene attributes are in global variable "attr" and also "cluster_count" has the "k" value.

In the following Code snippet:1, the similarity between the each gene and k cluster centroids are calculated and stored in a "dm" which is a hashmap. The key is "GeneId,ClusterID" and Value is its similarity.

Code Snippet 1: Calculate similarity between centroids and each data object

```
private void kmeans(double[,] cluster, int itr)
{
  int[,] cl = new int[rows + 1, 2];
  double[,] mean = new double[cluster_count, rows + 1];
  Dictionary<string, double> dm = new Dictionary<string, double>();
  for (int i = 1; i < rows + 1; i++)
  {
    for (int j = 0; j < cluster_count; j++)
    {
      double sum = 0;
      double val = cluster[j, 0];
      for (int k = 1; k < 17; k++)
      {
         if (itr == 0)
         {
            sum = sum + Math.Pow(attr[i, k] - attr[Convert.ToInt32(val), k], 2);
         }
         else
         {
            sum = sum + Math.Pow(attr[i, k] - cluster[j, k], 2);
         }
      }
      dm.Add(i + "," + Convert.ToInt32(val), 1 / (1 + Math.Sqrt(sum)));
    }
}
```

Code Snippet 2: Assign data objects to the most similar cluster.

```
foreach (KeyValuePair<string, double> kp in dm)
{
  count++;
  if (kp.Value > min)
  {
    min = kp.Value;
    key = kp.Key;
  }
  if (count == cluster_count)
  {
    string[] words = key.Split(',');
    cl[index, 0] = Convert.ToInt32(words[0]);
    cl[index, 1] = Convert.ToInt32(words[1]);
    index++;
    min = 0;
    count = 0;
    key = null;
  }
}
```

In Code Snippet 2 each gene is assigned to cluster whose centroid it is most similar to. The key-value pair of hashmap “dm” for each gene is processed and the key with

the minimum Value is selected and the gene is assigned to the cluster whose ClusterId is in the key.

Code Snippet 3: Calculate actual centroid of the cluster.

```
for (int j = 0; j < cluster_count; j++)
{
  count = 0;
  for (int i = 0; i < rows; i++)
  {
    if (cl[i, 1] == cluster[j, 0])
    {
      int id = cl[i, 0];
      mean[j, 0] = j;
      for (int k = 1; k < 17; k++)
      {
        mean[j, k] = mean[j, k] + attr[id, k];
        if (count == 0)
        {
          if (itr == 0)
          {
            mean[j, k] = mean[j, k] + attr[Convert.ToInt32(cluster[j, 0]), k];
          }
          else
          {
            mean[j, k] = mean[j, k] + cluster[j, k];
          }
        }
        count++;
      }
    }
    for (int k = 1; k < 17; k++)
    {
      mean[j, k] = mean[j, k] / (count + 1);
    }
  }
```

In Code Snippet 3 we show how we calculated actual centroids of the clusters formed. All the attributes of the genes belonging to a cluster are added respectively and stored in “mean”. “Count” has the number of genes in that cluster. The mean of the attributes are calculated, which is the attribute values of the new centroids.

Code Snippet 3: Check if the centroids have changed

```
count = 0;
int count1 = 0;
for (int j = 0; j < cluster_count; j++)
{
  for (int k = 1; k < 17; k++)
  {
    if (mean[j, k] == cluster[j, k])
    {
      count1++;
    }
  }
  if (count1 == 16)
  {
    count++;
    count1 = 0;
  }
}
itr++;
kmeans(mean, itr);
```

In Code Snippet 3 we check if the value of centroids have changed. If the value is changed then kmeans is again recursively called. This happened until the value of the centroids remain unchanged or number of specified iterations is reached.

Output:

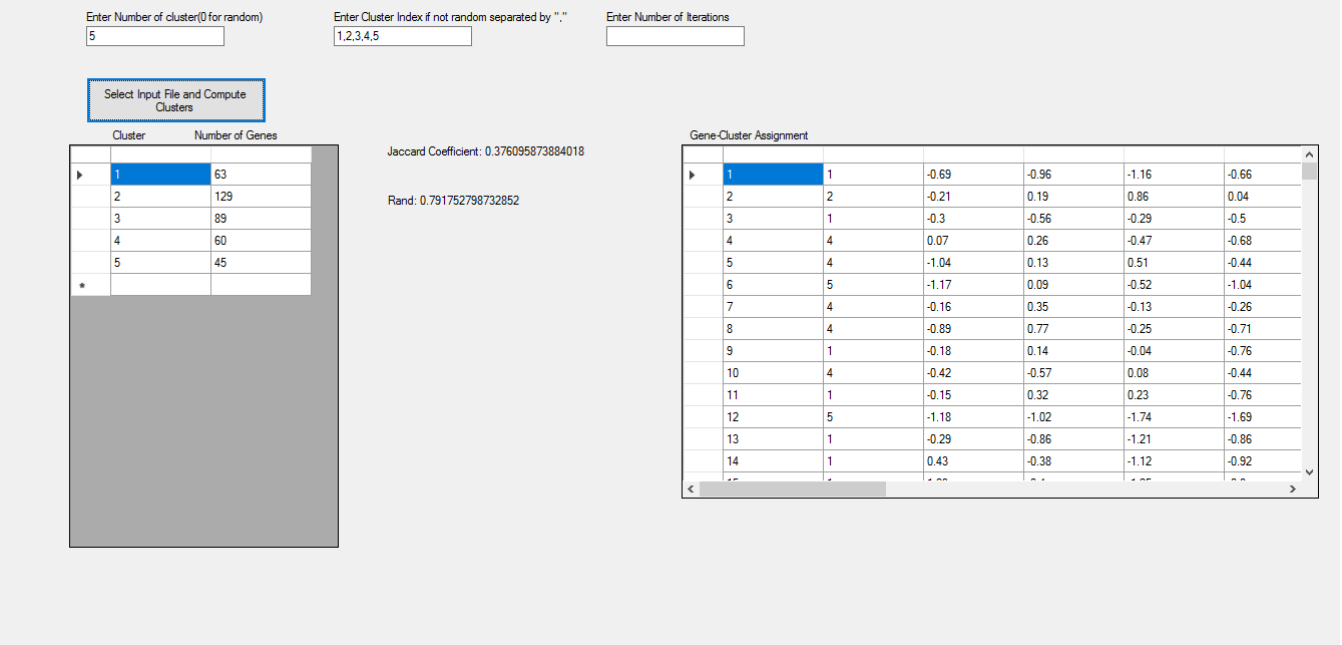
For Input file:"cho.xlsx"

Here the input parameters are

Number of Clusters = 5

Index of Centroids = 1,2,3,4,5

Number of Iterations= Not entered.



Number of Genes classified into each Cluster are

Cluster 1: 63 Cluster 2: 129 Cluster 3: 89 Cluster 4: 60 Cluster 5: 45

Cluster assignment of each gene along with its data is shown in right table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes.

Jaccard Coefficient: 0.376

Rand: 0.791

Result Evaluation

Rand and Jaccard are quite high for this dataset compared to other clustering techniques. This shows that k-means is quite suitable for data which is distributed in less number of clusters and less outliers.

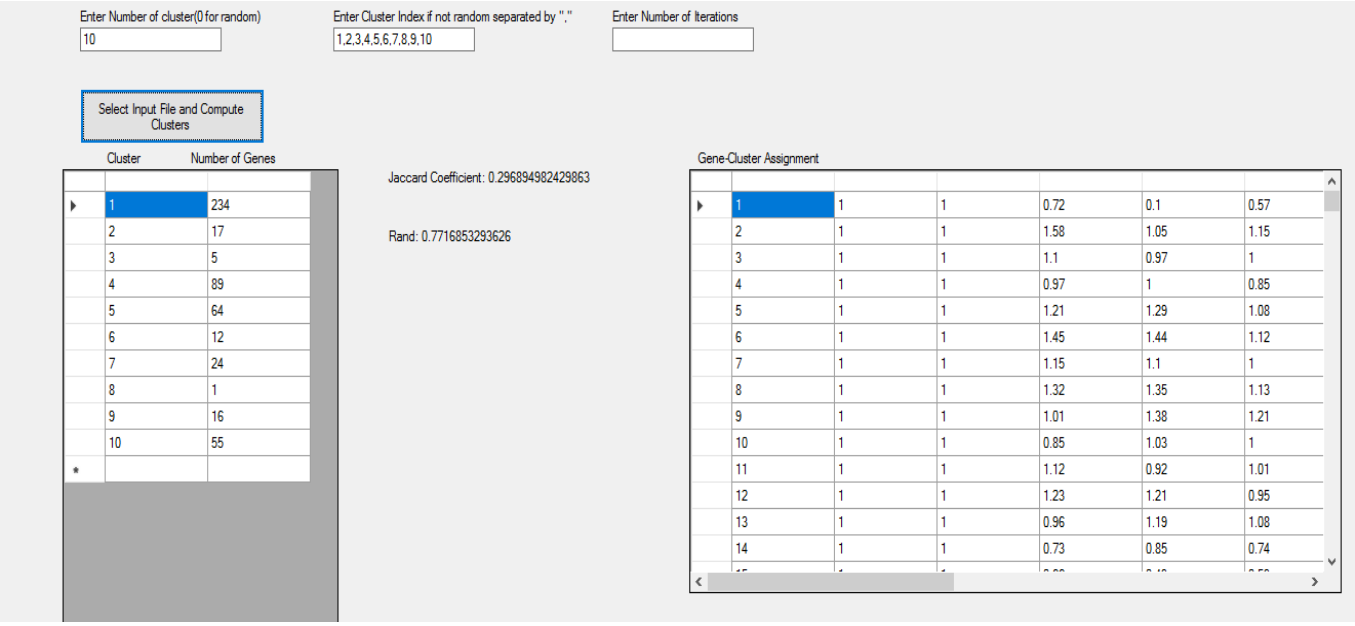
For Input file:"iyer.xlsx"

Number of Clusters = 10

Index of Centroids =1,2,3,4,5,6,7,8,9,10

Number of Iterations= Not entered.

There were 10 clusters formed as shown in the screenshot



Number of Genes classified into each Cluster are

Cluster 1: 234 Cluster 2: 17 Cluster 3: 5 Cluster 4: 89 Cluster 5: 64

Cluster 6: 12 Cluster 7: 24 Cluster 8: 1 Cluster 9: 16 Cluster 10: 55

Cluster assignment of each gene along with its data is shown in right table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes.

Jaccard Coefficient: 0.296

Rand: 0.771

Result Evaluation

Rand and Jaccard are little less for this dataset compared to previous dataset. It is mainly because of the fact that there are large number of outliers present in the data. Since K-means has a tendency to converge the centroids, it reduced the value of Jaccard and Rand. This shows that k-means is less suitable for data which has more outliers.

Hierarchical Agglomerative clustering with Single Link (Min)

In hierarchical clustering unlike in K-means, we need not choose initial Centroids. It does a sequence of clustering assignments, either by starting with each data object as a separate cluster. And ends by adding all the points into one single cluster which is known as "Agglomerative Hierarchical clustering" or vice versa known as. "Divisive Hierarchical clustering". In Agglomerative i.e. bottom-up, we merge the 2 most similar clusters as 1, where as in Divisive i.e. top-down, we split a cluster into 2 least similar clusters. Linkage or distance is a way to measure the similarity between 2 clusters.

In our implementation, the distance between two clusters is represented by the distance of the closest pair of data objects belonging to different clusters. Here also we use Euclidian distance to calculate the similarity.

We have used hashmap to store the distance matrix "dm" to store their similarity between all the genes. The key of the hashmap "dm" has the 2 gene id's "GeneId1-GeneId2" and similarity between the 2 genes is stored in its value. This is implemented as shown in code snippet 1.

Code Snippet 1: Calculate dissimilarity distance matrix between all the genes

```
for (int i = 1; i < rows+1; i++)
{
   cluster[i, 0] = i.ToString();
   for (int j = 1; j < rows+1; j++)
   {
     cluster[0, j] = j.ToString();
     double sum = 0;
     string key = i + "-" + j;
     if (i == j)
     {
       dm1.Add(key, 1);
       cluster[i, j] = 1.ToString();
     }
     else
     {
       for (int k = 1; k < 17; k++)
       {
         sum = sum + Math.Pow(attr[i, k] - attr[j, k], 2);
       }
       double sim_val = Math.Sqrt(sum);
       dm1.Add(key, sim_val);
       cluster[i, j] = sim_val.ToString();
       if (sim_val < min)
       {
         min = sim_val;
         min_key = key;
       }
     }
   }
 }
Hier(cluster, itr, min, min_key);
```

Also we are passing 2 most similar genes key and value as arguments to the function "private void Hier(string[,] cluster, int itr, double min_val, string min_key)" which has the actual functionality of the algorithm. The variable "cluster" has the initial clusters, in this case each gene as an individual cluster.

Code Snippet 2: Joining 2 most similar clusters

```
private void Hier(string[,] cluster,int itr,double min_val,string min_key)
{
  string[,] cl = new string[rows - itr, rows - itr];
  string[] words = max_key.Split('-');
  int col = 1;
  for (int k = 0; k < words.Length; k++)
  {
```

```
    if (k == 0)
    {
      cl[0, 1] = words[k];
    }
    else
    {
      cl[0, 1] = cl[0, 1] + "," + words[k];
    }
   }
```

In code snippet 2, 2 most similar clusters whose ids were passed as an argument min_key of the function “Hier” were joined and all the clusters were stored in cl for further computing.

Code Snippet 3: Finding the next most similar clusters

```
for (int i = 1; i < rows - itr; i++)
{
  string[] words_i = cl[0, i].Split(',');
  for (int k = 0; k < words_i.Length; k++)
  {
    for (int j = i + 1; j < rows - itr; j++)
    {
      string[] words_j = cl[0, j].Split(',');
      for (int l = 0; l < words_j.Length; l++)
      {
        double val = 0;
        if (dm1.TryGetValue(words_i[k] + "-" + words_j[l], out val))
        {
          if (val < t_min)
          {
            t_min = val;
            t_key = cl[0, i] + "-" + cl[0, j];
          }
        }
      }
    }
  }
}
```

In code snippet 3, we find the next most similar clusters. This is done by finding the clusters with the least minimum distance in the distance matrix “dm” which is represented in the form of a hashmap. The id’s and similarity value of this cluster is stored in t_key and t_min respectively. This is sent as parameters of the recursive

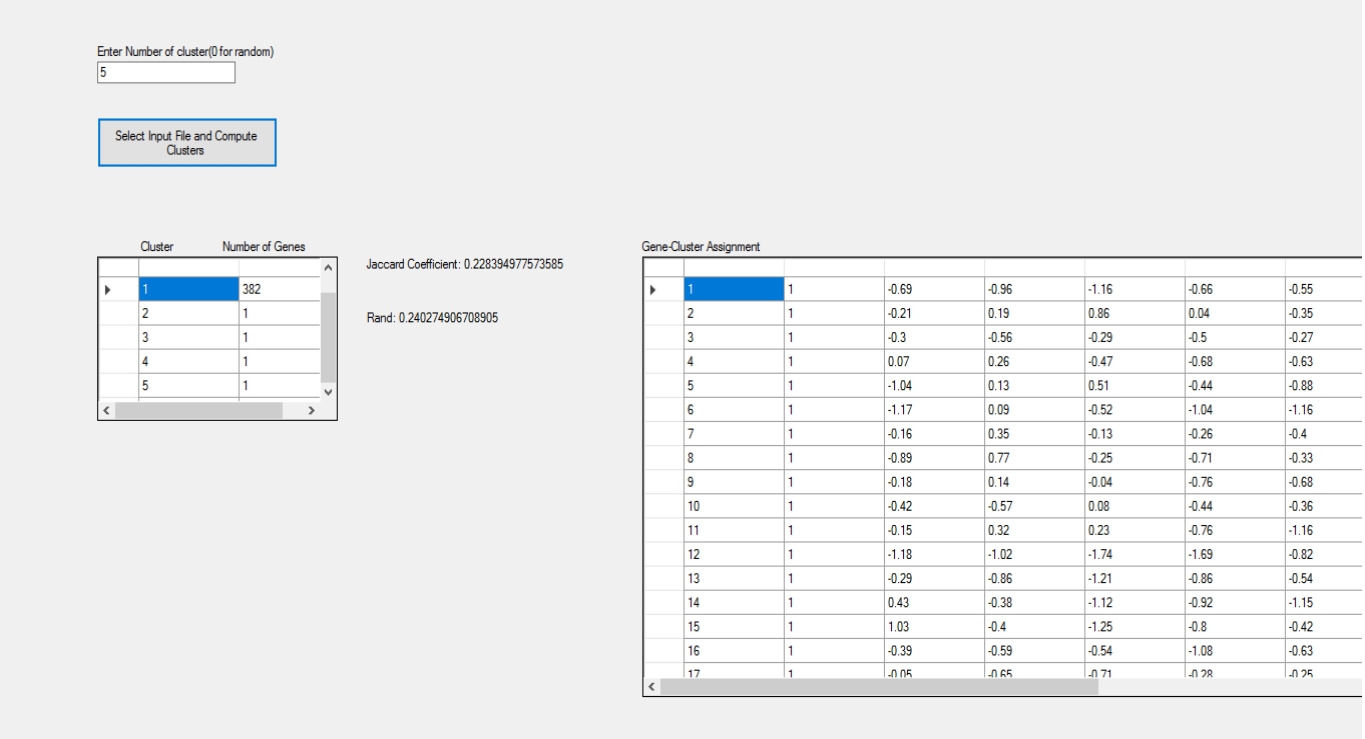
"Hier" call until all the clusters are merged as one or number of specified iterations are reached.

Output:

For Input file:"cho.xlsx"

Here the input parameters are

Number of Clusters = 5



Number of Genes classified into each Cluster are

Cluster 1: 382 Cluster 2: 1 Cluster 3: 1 Cluster 4: 1 Cluster 5: 1

Cluster assignment of each gene along with its data is shown in right table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes.

Jaccard: 0.228

Rand = 0.240

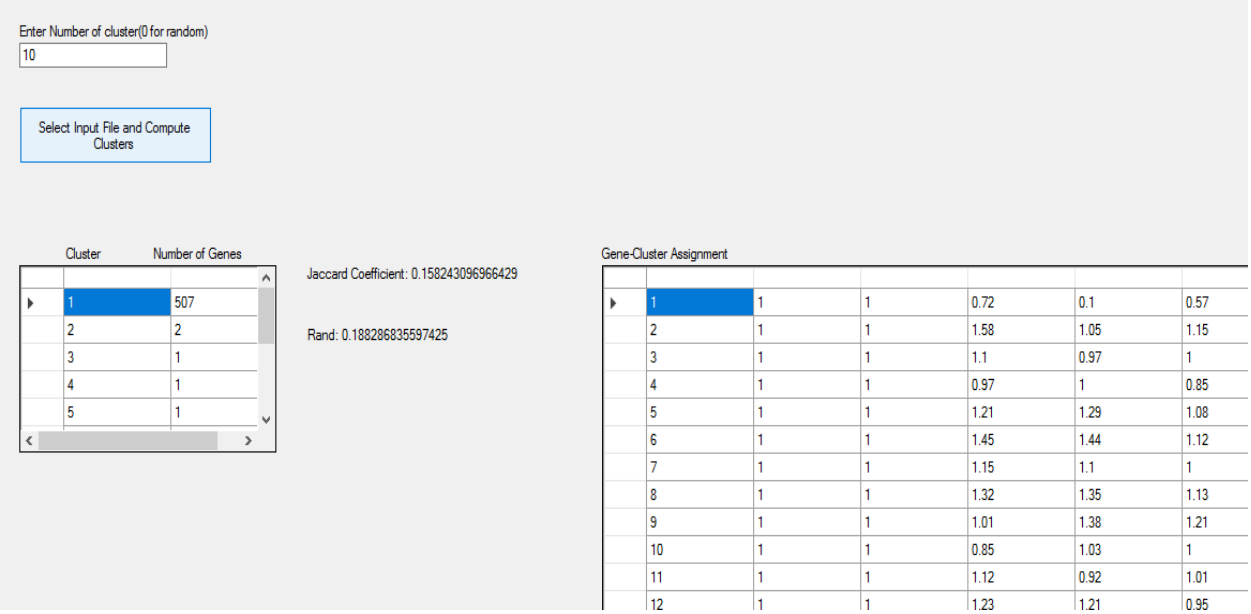
Result Evaluation

Rand and Jaccard are significantly less for this dataset compared to other clustering algorithms. It is mainly because of the fact that data is so similar that they are not following a hierarchy and being assigned to the same cluster which basically defeats the purpose of clustering. So hierarchical clustering is not suitable for this dataset.

For Input file:"Iyer.xlsx"

Here the input parameters are

Number of Clusters = 10



Number of Genes classified into each Cluster are

Cluster 1: 507 Cluster 2: 2 Cluster 3: 1 Cluster 4: 1 Cluster 5: 1

Cluster 6: 1 Cluster7: 1 Cluster 8: 1 Cluster 9: 1 Cluster 10: 1

Cluster assignment of each gene along with its data is shown in right table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes.

Jaccard: 0.158

Rand = 0.188

Result Evaluation

Rand and Jaccard are significantly less for this dataset compared to other clustering algorithms and previous dataset. It is mainly because of the fact that data is so similar that they are not following a hierarchy and being assigned to the same cluster which basically defeats the purpose of clustering. One of the another reason for so small value of rand and Jaccard is because of the presence of outliers which are being assigned to a cluster instead of being classified as outlier. So hierarchical clustering is not suitable for this dataset.

Density-based Clustering: DBSCAN

In density based clustering clusters are formed based on the density of points over a data space i.e. basically dense regions separated by regions of lower density. DBSCAN which refers to Density-Based Spatial Clustering of Applications with Noise,unlike K-Means, does not require the number of clusters as a parameter. Rather it infers the number of clusters based on the data. The 2 fundamental parameters for this algorithm are ε-neighborhood is ε: which is the radius of a data point p around which neighbor points are considered and minPts that is the minimum number of data points in a neighborhood required to define a cluster.

In DBSCAN, for each point that is not classified already, we need to find all the neighbor points within the ε-neighborhood. If the number of points in its ε-

neighborhood is greater than or equal to minPts, then the point is classified as core point and a cluster is formed with this point and the points in its ε-neighborhood. In case the point is not a core point then it is classified as noise. We have implemented this algorithm as shown in the following code snippets. The value of its ε "esp" and minPts are dynamically entered by the user. The similarity distance matrix is calculated the same way as calculated in KMeans and is stored in hashmap named "dm1".

Code Snippet 1: Function DBSCAN

```
private void DBSCAN(int[,] D, double eps, int MinPts)
{
  int C = 0;
  for (int i = 1; i <= rows; i++)
  {
    if (D[i, 3] == 0)
    {
      D[i, 3] = 1;
      int[] Neighborpts = regionQuery(D[i,1], D, eps);
      if (Neighborpts.GetLength(0) < MinPts)
      {
        D[i, 3] = -1;
        noise_count++;
        noise[0, 1] = noise[0, 1] + "," + D[i,1].ToString();
      }
      else
      {
        C++;
        expandCluster(D, D[i, 1], Neighborpts, C, eps, MinPts);
      }
    }
  }
  cluster_count = C;
}
```

In code snippet 1 shows the First function to be called, " private void DBSCAN(int[,] D, double eps, int MinPts)". Here "D" has the gene id's and will also store 0 or 1 or -1 to signify whether that particular gene has been classified as core point[1] or not classified[0] or noise [-1]. Here if the point in D has not been classified yet [0] then we call the function "regionQuery" which gives the points in the ε-neighborhood [Neighborpts] of that specific not classified point. If the number of points returned

[Neighborpts.GetLength(0)] is less than minPts then the point is classified as noise [-1] else function expandCluster() is called.

Code Snippet 2: Function "RegionQuery"

```
private int[] regionQuery(int p, int[,] D, double eps)
{
  int[] nbr = new int[rows];
  int count = 0;
  for (int i = 0; i <= rows; i++)
  {
    string key = null;
    if (p < D[i, 1])
    {
      key = p + "-" + D[i, 1];
    }
    else
    {
      key = D[i, 1] + "-" + p;
    }
    double val = 0;
    if (dm1.TryGetValue(key, out val))
    {
      if (val >= eps)
      {
        nbr[count] = D[i, 1];
        count++;
      }
    }
  }
  Array.Resize(ref nbr, count);
  return nbr;
}
```

In function "RegionQuery", a point p is sent as argument for which we find the points in its ε-neighborhood. We search the distance matrix "dm1" for similarity value between the point p and all other points by searching for key "PointPId-eachPointinDId". If its similarity value is greater than or equal to the specified ε value, then that point in D is added to ε-neighborhood [nbr] of point p and at the end "nbr"returned.

In function ExpandCluster we find the ε-neighborhood points of the unclassified points in ε-neighborhood of point p. It is implemented similar to function DBSCAN, except when ε-neighborhood "Neighborpts1" points returned from RegionQuery is

greater than minPts then “Neighborpts1” is joined with “Neighborpts”, which has the ε-neighborhood of the actual point p.

Code Snippet 3: Function ExpandCluster

```
private void expandCluster(int[,] D,int p,int[] Neighborpts,int C,double eps,int MinPts)
{
  cluster[C, 0] = C.ToString();
  cluster[C, 1] = "," + p.ToString() + ",";
  int col=1;
  for (int i = 0; i < Neighborpts.Length; i++)
  {
    int p1 = Neighborpts[i];
    for (int j = 0; j < D.GetLength(0); j++)
    {
      if (D[j, 1] == p1)
      {
        if (D[j, 3] == 0)
        {
          D[j, 3] = 1;
          int[] Neighborpts1 = regionQuery(p1, D, eps);
          if (Neighborpts1.Length >=MinPts)
          {
            for(int x = 0; x < Neighborpts1.Length; x++)
            {
              int flag = 0;
              for(int y = 0; y < Neighborpts.Length; y++)
              {
                if (Neighborpts1[x] == Neighborpts[y])
                {
                  flag= 1;
                }
              }
              if (flag ==0)
              {
                Array.Resize(ref Neighborpts, Neighborpts.Length+1);
                Neighborpts[Neighborpts.Length-1] = Neighborpts1[x];
              }
            }
          }
        }
      }
    }
    int flag_p = 0;
    for (int k = 1; k <=C;k++)
    {
      if(cluster[k, 1].Contains("," + p1.ToString() +","))
      {
        flag_p = 1;
      }
    }
    if (flag_p == 0)
    {
      cluster[C, 1] = cluster[C, 1] + "," + p1.ToString() + ",";
      col++;
    }
```

```
    }
}
```

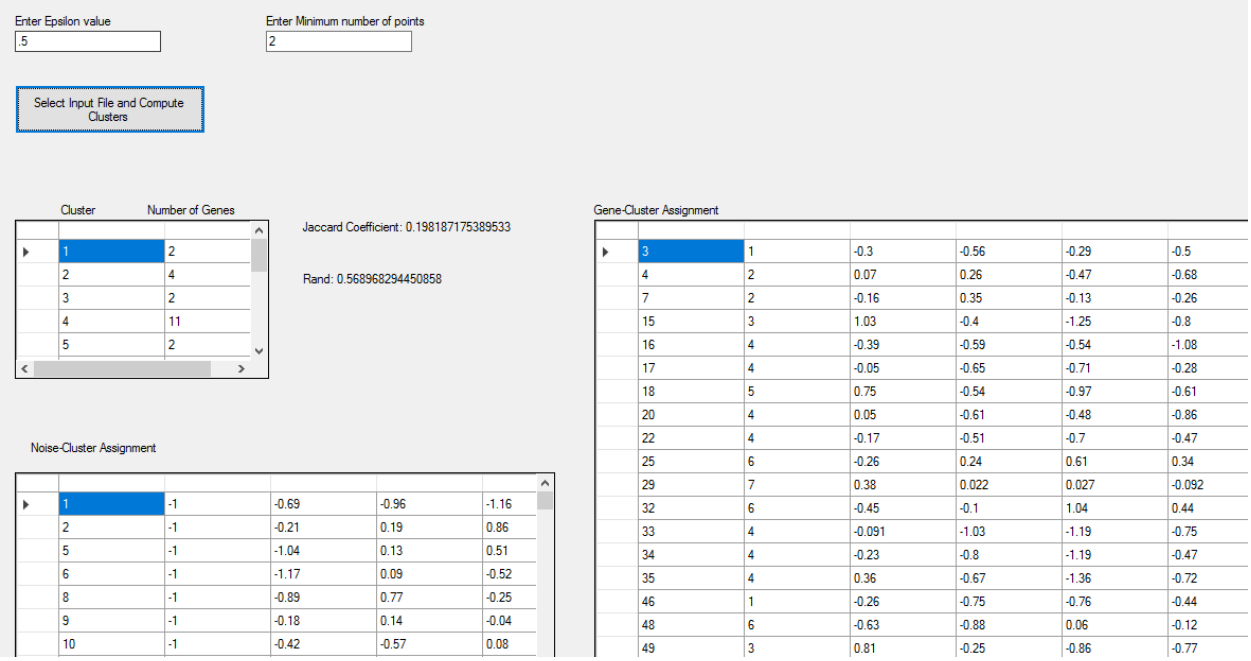
Output:

For Input file:"cho.xlsx"

Here the input parameters are

Epsilon Value = .5(Similarity) equivalent to 1(Dissimilarity)

Minimum number of points = 2



Number of Genes classified into each Cluster are

Cluster 1: 2 Cluster 2: 4 Cluster 3: 2 Cluster 4: 11 Cluster 5: 2

Cluster 6: 138 Cluster 7: 3 Cluster 8: 2 Cluster 9: 2 Cluster 10: 2

Cluster 11: 2 Cluster 12: 4 Cluster 13: 2 Cluster 14: 4

Outliers (Noise) = 206

Cluster assignment of each gene along with its data is shown in right table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes. Noise Assignment of each gene along with its data is shown in bottom left table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes.

Jaccard: 0.198

Rand: 0.569

Result Evaluation

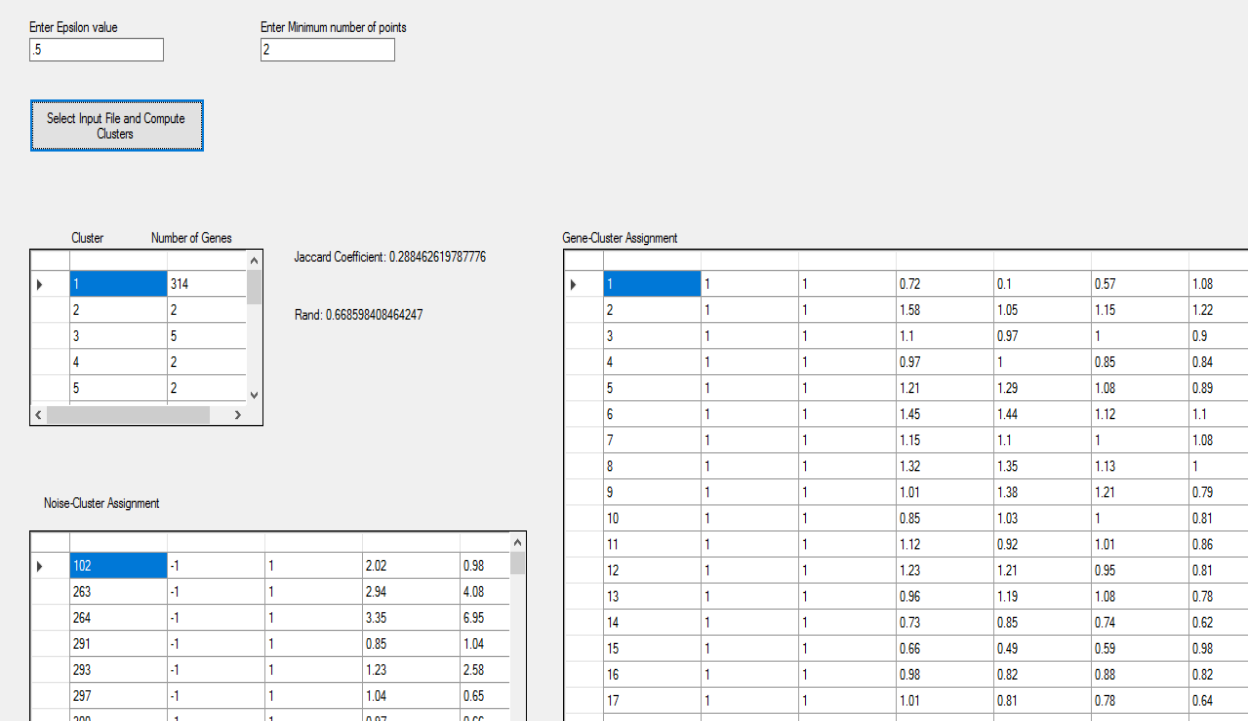
Rand and Jaccard are very less for this dataset compared to other clustering algorithms. It is mainly because of the fact that data is so dense that this algorithm is not able to distinguish on the basis of density. DBSCAN also has a tendency to mark the outliers very strictly so that's why outliers are specified on the data which has no outliers in the ground truth. So Density based clustering is not suitable for this dataset.

For Input file:"Iyer.xlsx"

Here the input parameters are

Epsilon Value = .5(Similarity) equivalent to 1(Dissimilarity)

Minimum number of points = 2



Number of Genes classified into each Cluster are

Cluster 1: 314 Cluster 2: 2 Cluster 3: 5 Cluster 4: 2 Cluster 5: 2

Cluster 6: 3 Cluster7: 6 Cluster 8: 2 Cluster 9: 2 Cluster 10: 30

Cluster 11: 2 Cluster 12: 2 Cluster 13: 2 Cluster 14: 2 Cluster 15: 2

Outliers (Noise) = 139

Cluster assignment of each gene along with its data is shown in right table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes. Noise Assignment of each gene along with its data is shown in bottom left table in which 1st column is Gene Id, 2nd Column is Cluster Id and rest of the columns are gene attributes.

Jaccard: 0.288

Rand: 0.668

Result Evaluation

Rand and Jaccard are significantly less for this dataset compared to other clustering algorithms but better than previous dataset. It is mainly because of the fact that data is so dense that this algorithm is not able to distinguish on the basis of density. DBSCAN also has a tendency to mark the outliers very strictly so that's why outliers are specified on the data which has no outliers in the ground truth. Because of this reason, Jaccard and Rand has increased because there were lot of outliers present in the data. So Density based clustering is not suitable for this dataset.

KMeans- MapReduce

MapReduce is a programing model which processes massive scalable data using multiple servers in a Hadoop cluster. The input to the is of a form "key-Value" pair, upon which the map Job computes and gives "Key-Value" pairs as the output. This is the input to the reducers after the combiners have sorted and combined the map "key-Value" pair output based on the keys.

In our KMeans MapReduce implementation, the Mapper function gets dataset as input. As shown in the following code snippet of the mapper function, we read the gene data one by one and find its similarity with the centroids. The initial centroids are read from a file in the main function. Based on these similarity values we find the min value that is most similar centroid "if(sim>min)". The key of the mapper id the cluster Id that the gene belongs to. And the value "c or words" is the gene id and its attribute values.

Code Snippet 1: Mapper Function

```
public static class Mapper extends Mapper<Object, Text, IntWritable, Text>{
```

```
private String[][] gt;

private double[] centroid;

private Text word = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

    try {

        double min=0;

        String[] line = value.toString().split(","); // Gene Id with attributes 18 column

        String c = null;

        IntWritable in_key = null;

        String s="";

        for (int j = 0; j < n_cent; j++){

                double sum = 0;

                 for (int k = 2; k < 18; k++){

                sum = sum + (Double.parseDouble(line[k])   -
        Double.parseDouble(cent[j][k]))*(Double.parseDouble(line[k])   - Double.parseDouble(cent[j][k]));

                }

        double sim=1 / (1 + Math.sqrt(sum));

        if(sim>min){

        c = null;

        min=sim;

        for (int k = 1; k < 18; k++){

                if(c==null){
                        c =   line[k].toString();
                }
        else{
                c =   c + "," + line[k].toString();
        }

        in_key = new IntWritable(j+1);

        }
        s=c;

        s=in_key+ "," + s;

        s=line[0] + "," + s+ "#";
```

```
            c =   line[0] + "," + c;

            }

            }
cluster=cluster + s;
word.set(c);
context.write(in_key, word);// centroid id with attr and point with id and attr

} catch (Exception e) {}

}
```

In the reducer function, the values of the mapper output is combined and sorted based on their keys. Say in case there were 5 centroids specified in the input file, then the input to the reducer is 5 keys which are the centroid cluster id's and their value is a iterable variable which has the genes and their attributes belonging to that centroid cluster. The average value of all the gene attributes belonging are calculate as shown in the code below. These average values form the centroids of the new cluster, its value is stored in "new_cent".

Code Snippet 2: Reducer Function

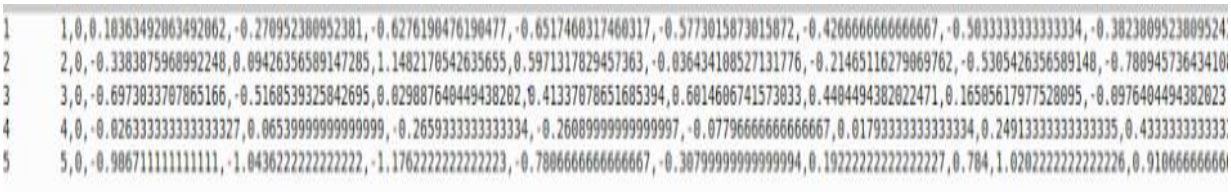
```
 public static class Reducer1 extends Reducer<IntWritable, Text, IntWritable, Text> {

 public void reduce(IntWritable key, Iterable<Text> values, Context context

) throws IOException, InterruptedException {

  for (Text val : values){

          String[] line = val.toString().split(",");
       for(int j = 2; j<line.length;j++){

       sum[j]= Double.parseDouble(line[j]) + sum[j];

       }

       count++;

      }
          Int kn=key.get();
          for(int j = 2; j<18;j++){

         sum[j]= Double.parseDouble(cent[kn-1][j]) + sum[j];

      }
          count++;

      for(int j = 2; j< length;j++){
```

```
        sum[j]=sum[j]/count;
            if(c==null){
            c =   String.valueOf(sum[j]);
            }
            else{

            c =   c + "," + String.valueOf(sum[j]);
            }

        }

c =   "0" + "," + c;
c =   key.toString() + "," + c;
new_cent[new_cent_count][0] = key.toString();
```

In the reducer function itself we see whether the values of the previous centroids and the new calculated centroids have changed or not. The reducer sets a value "Iter_exit" to 1 in the case of unchanged centroids, which ends the iteration else the Map function is called again.

Output for Input dataset: "Cho.xlsx"

The following is the final data centroids after the last iteration.



Number of Genes classified into each Cluster are

Cluster 1: 63 Cluster 2: 129 Cluster 3: 89 Cluster 4: 60 Cluster 5: 45

Jaccard Coefficient: 0.376 Rand: 0.791

Result Evaluation

Rand and Jaccard are quite high for this dataset compared to other clustering techniques. This shows that k-means is quite suitable for data which is distributed in

less number of clusters and less outliers. We can observe that the results of both the normal implementation of kmeans and MapReduce kmeans is the same. Even the Jaccard Coefficient and Rand Index value are the same. This says that both the algorithms are working correctly. While considering the complexity between the 2 implementations, ideally the MapReduce implementation is both space and time efficient. As it's a parallel processing model which is scalable and fast. But since we have implemented it on a single node Hadoop, it is more time consuming than the normal implementation.

Validation based on External Indexes- Jaccard Coefficient and Rand Index

Based on the resulted clustered genes and their ground truth value we calculate the Rand Index and Jaccard Coefficient. These are the external indexes which are used to evaluate our clustering algorithms.

External Indexes on "Cho"	Jaccard Coefficient	Rand Index
KMeans Clustering	0.376	0.791
Hierarchical Agglomerative Clustering	0.228	0.240
Density Based Clustering: DBSCAN	0.198	0.569

When the Jaccard Coefficient is 1, then it means the clustering was an exact match to the ground truth. Where as in the case of Rand Index 0 indicates that the two data

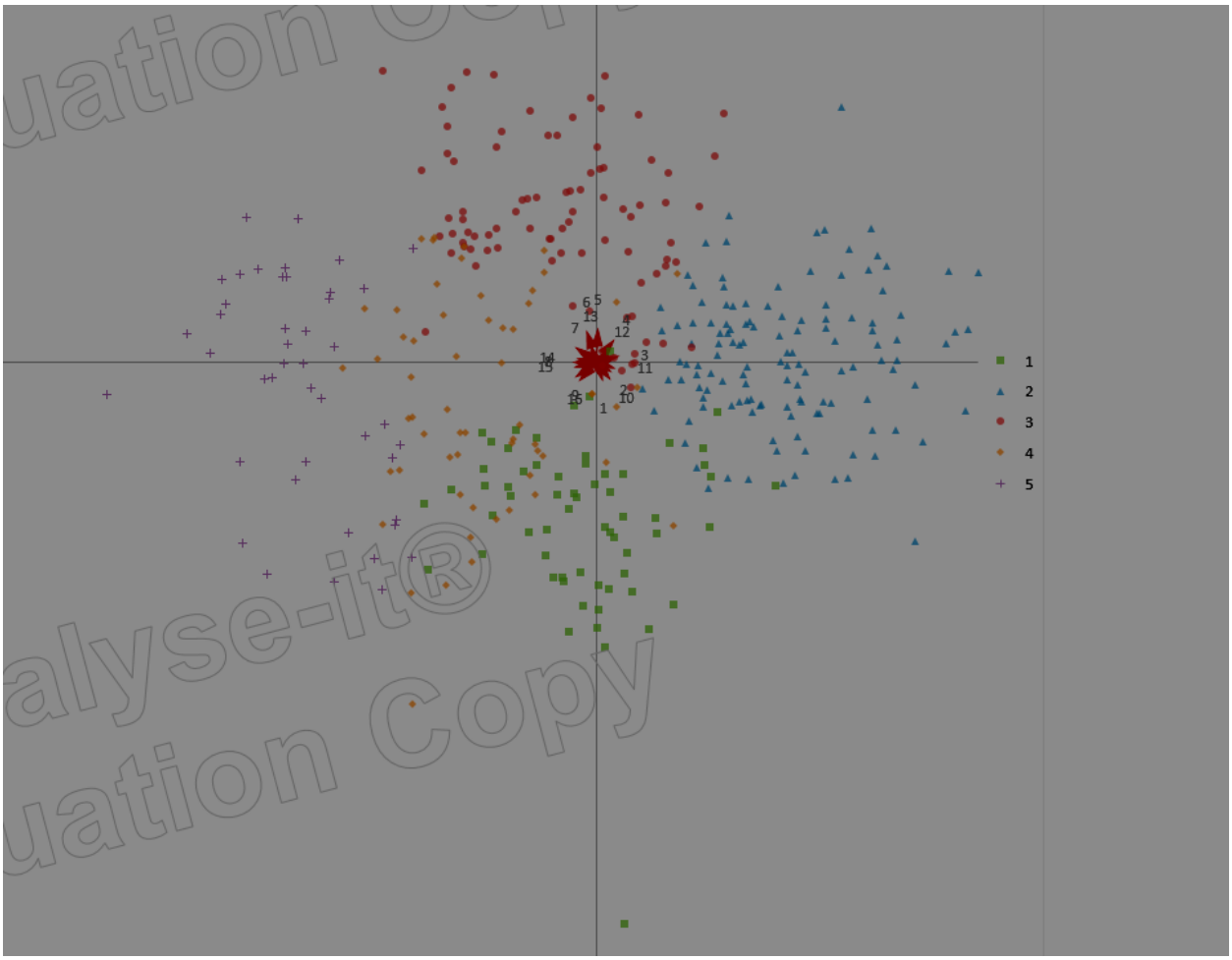
clusters do not agree on any pair of points and 1 indicates that the data clusters are exactly the same.

External Indexes on "Iyer"	Jaccard Coefficient	Rand Index
KMeans Clustering	0.296	0.771
Hierarchical Agglomerative Clustering	0.158	0.188
Density Based Clustering: DBSCAN	0.288	0.668

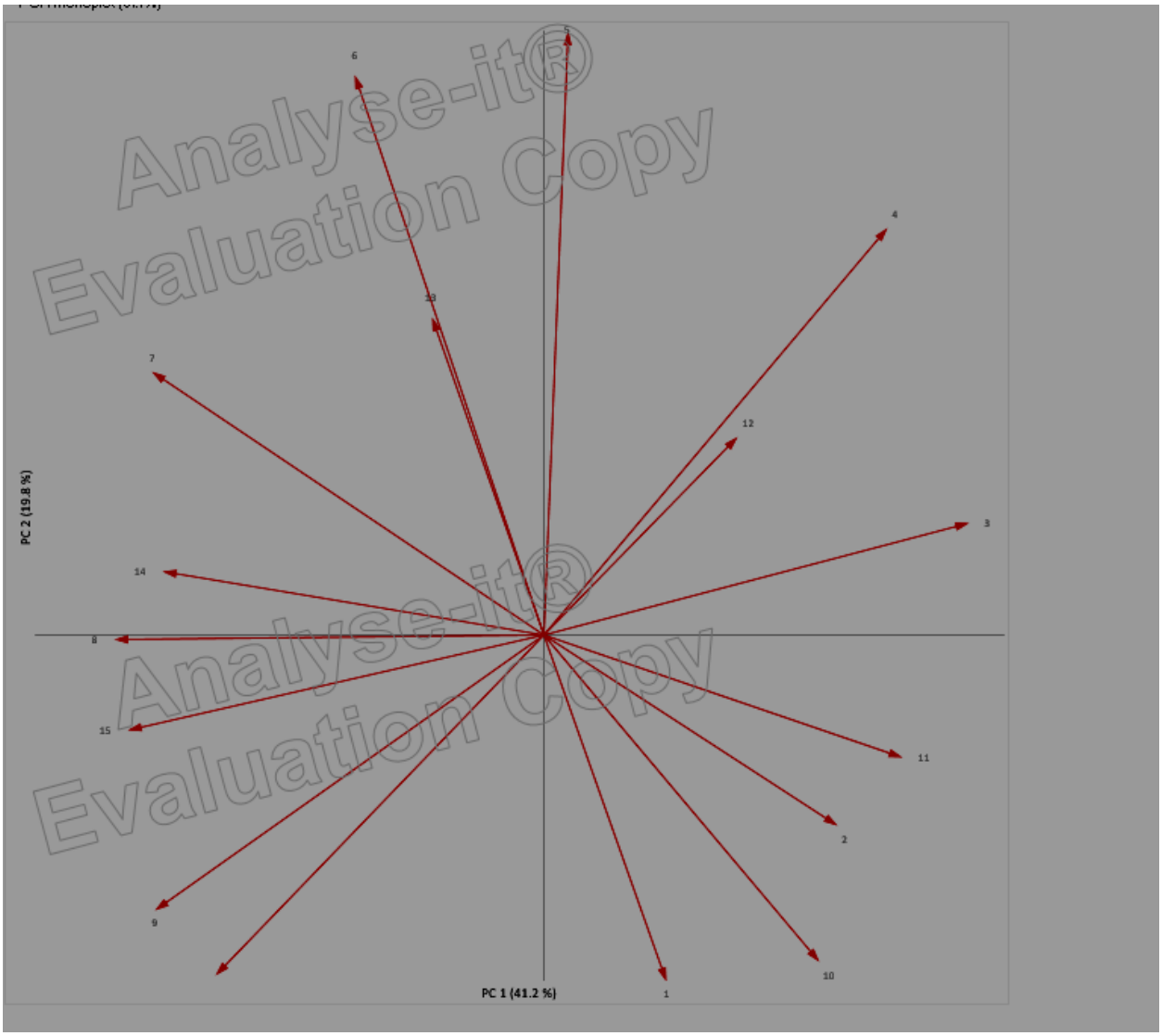
PCA: Principal Component Analysis

Principal component analysis (PCA) is a technique used for exploration and visualization of datasets. It emphasizes variation and brings out strong patterns in a dataset.

KMeans Clustering
Dataset "cho.xlsx"

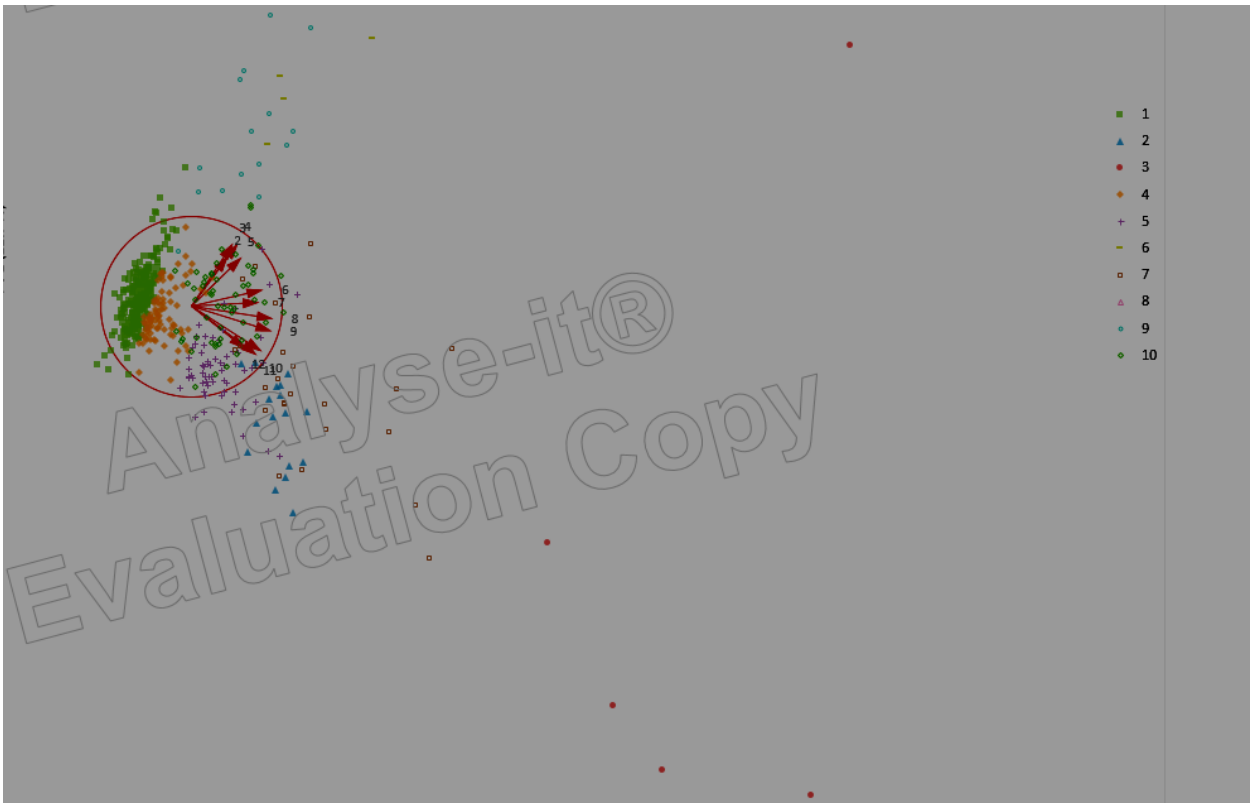


From above scatter plot it could be easily deduced that data is very dense in nature and concentrated across centroids and because of that, k means was able to get the best result in terms of Jaccard and Rand.



Above image depicts the relation of gene attributes to each other. For example Attribute 6 and 13 are very similar to each other while attribute 7 and 11 are almost dissimilar in nature.

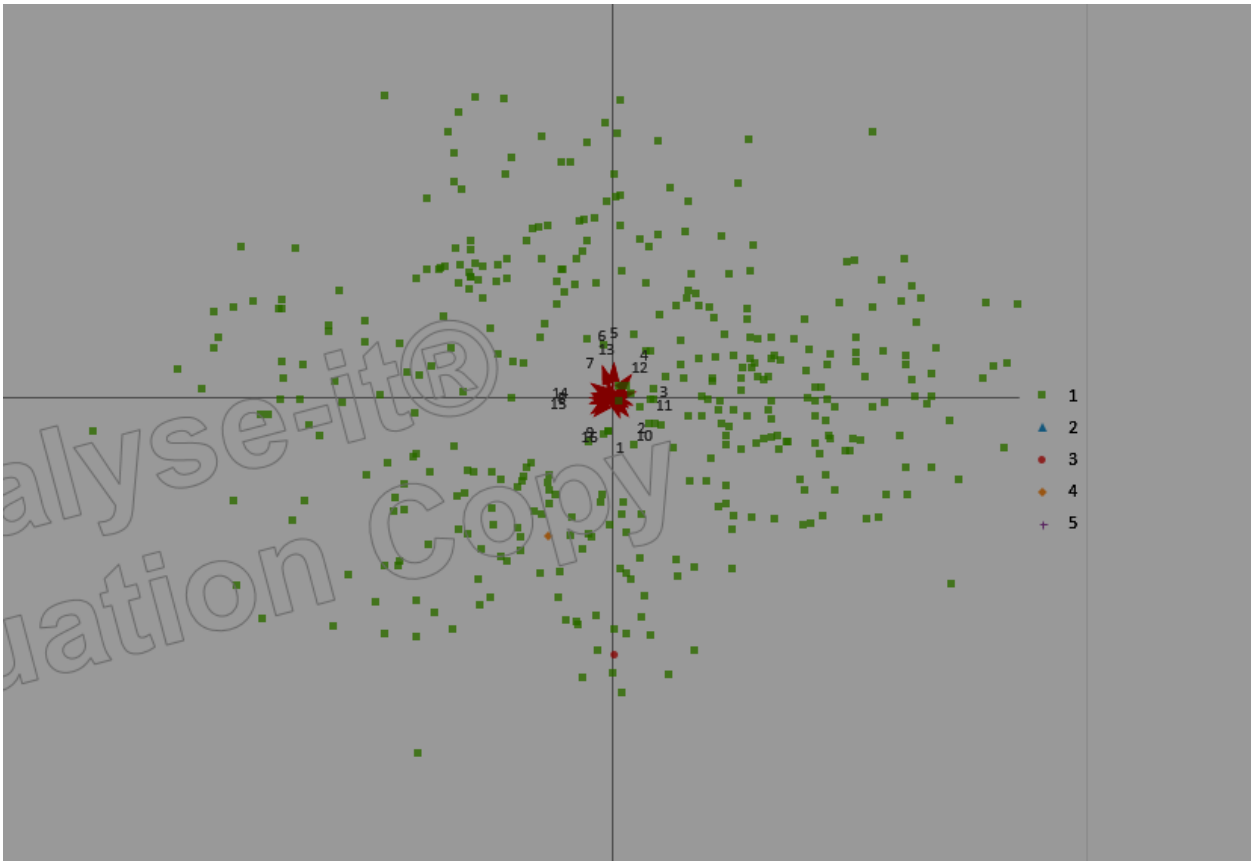
Dataset "Iyer.xlsx"



From above scatter plot it could be easily deduced that data is very dense in nature at some points but there are lot of outliers and because of that, Jaccard and Rand was less than previous dataset.

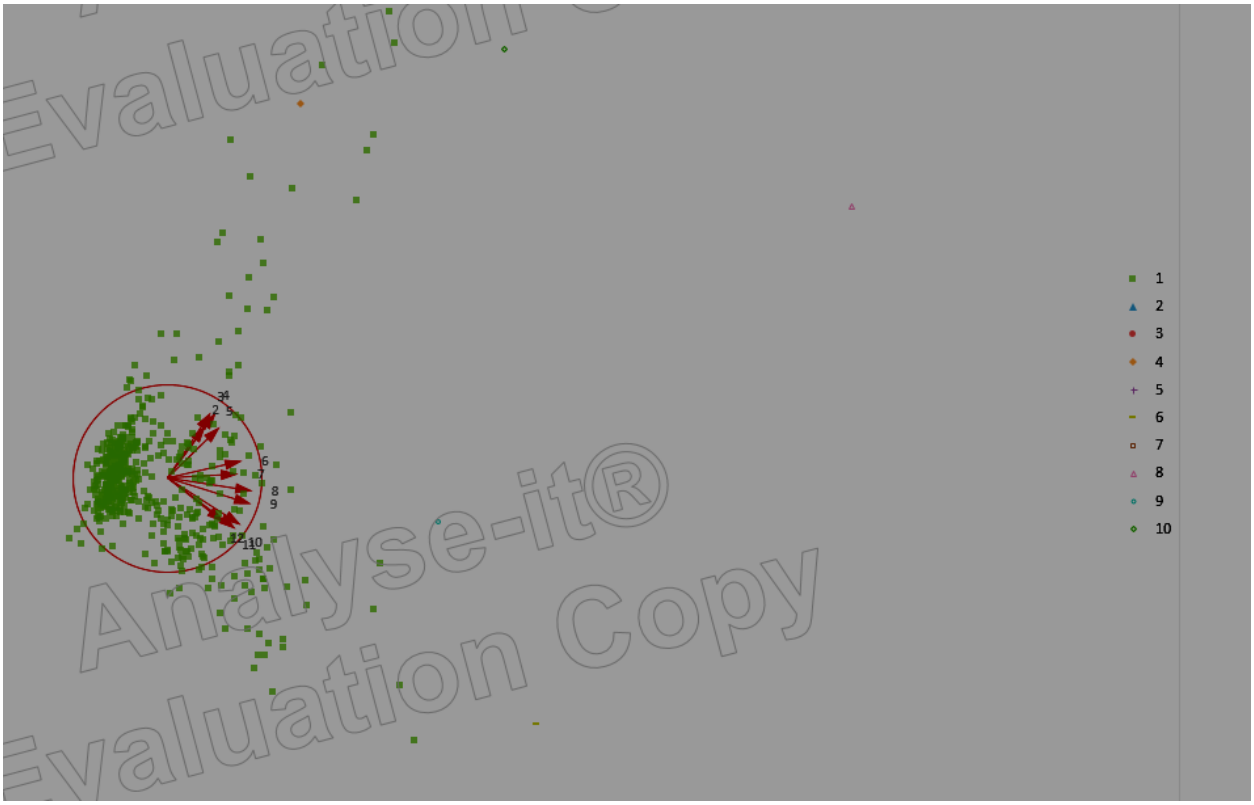
Above image depicts the relation of gene attributes to each other. This is very different from “cho” as its attributes were all over the circle i.e. widely spread but in “Iyer” they are inclined to a similar direction.

Hierarchical Agglomerative clustering with Single Link (Min)
Dataset "cho.xlsx"



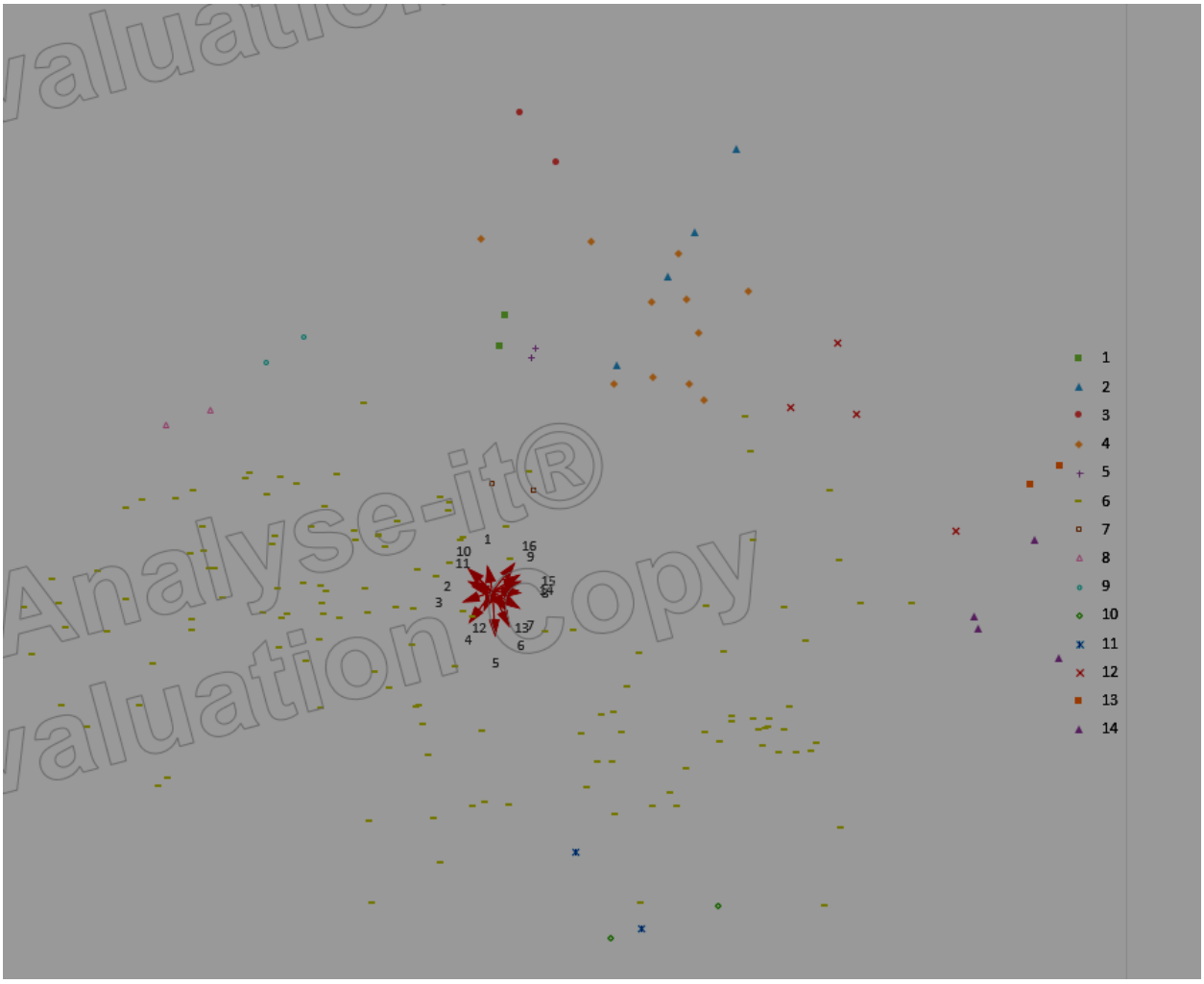
Above image depicts that maximum of the genes are assigned in a same cluster instead of being scattered all around the place. So this is not a proper clustering method and hence values of jaccard and rand are very low.

Dataset "Iyer.xlsx"

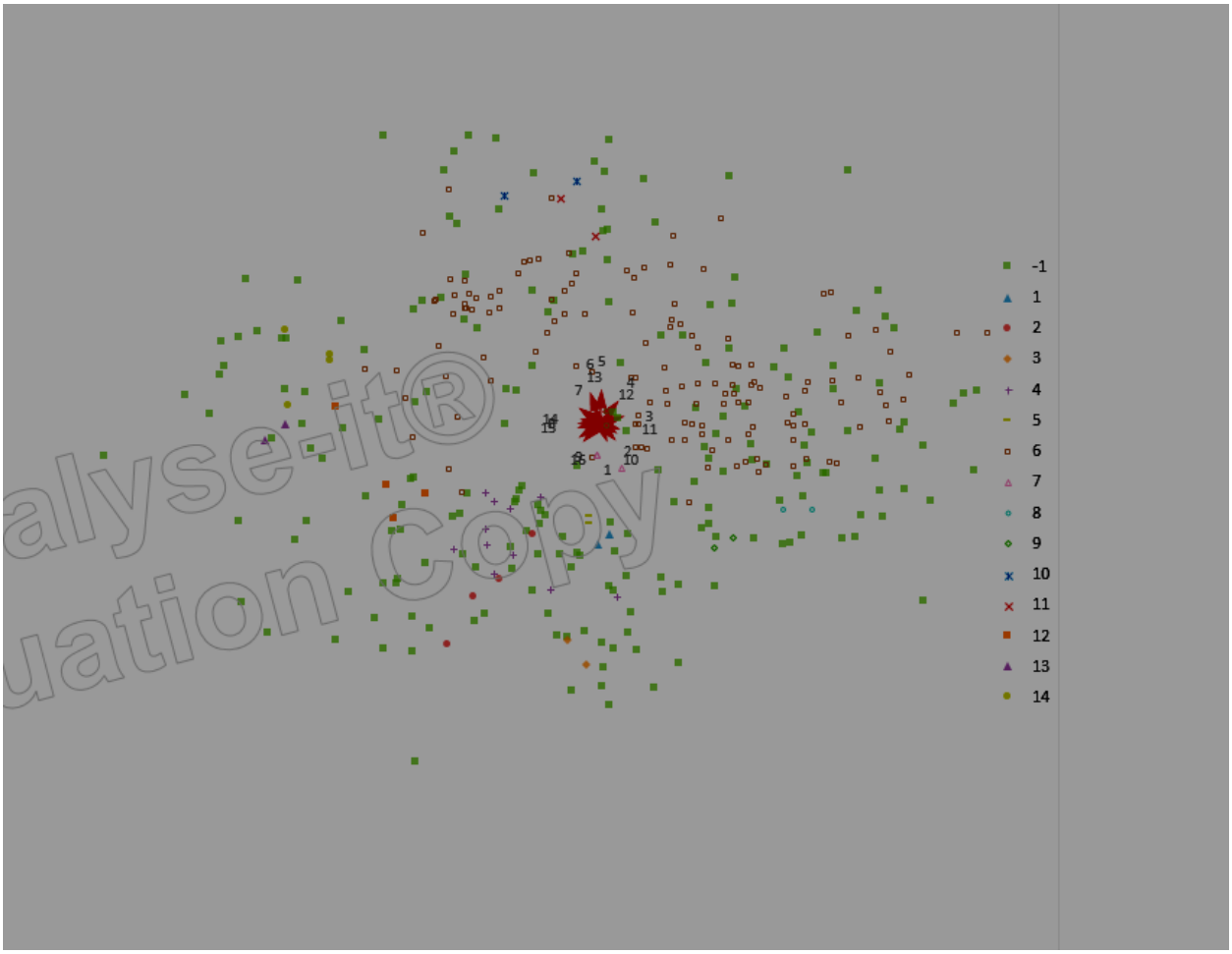


Above image depicts that maximum of the genes are assigned in a same cluster instead of being scattered all around the place. So this is not a proper clustering method and hence values of jaccard and rand are very low.

Density-based Clustering: DBSCAN
Dataset "cho.xlsx"

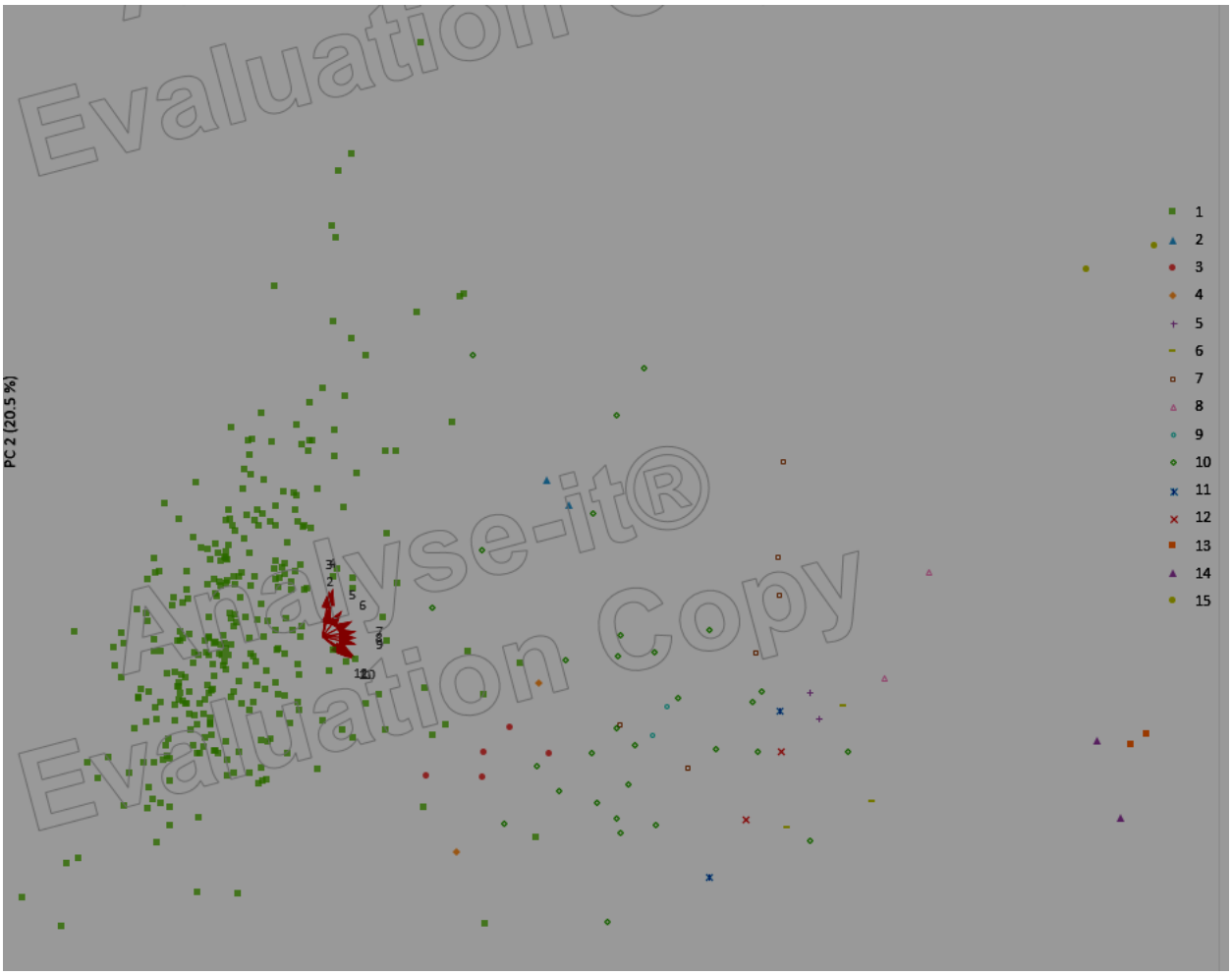


Above image is the depiction of cluster assignment excluding outliers so this looks like a fair clustering but it generates a lot of outliers as shown in following image.

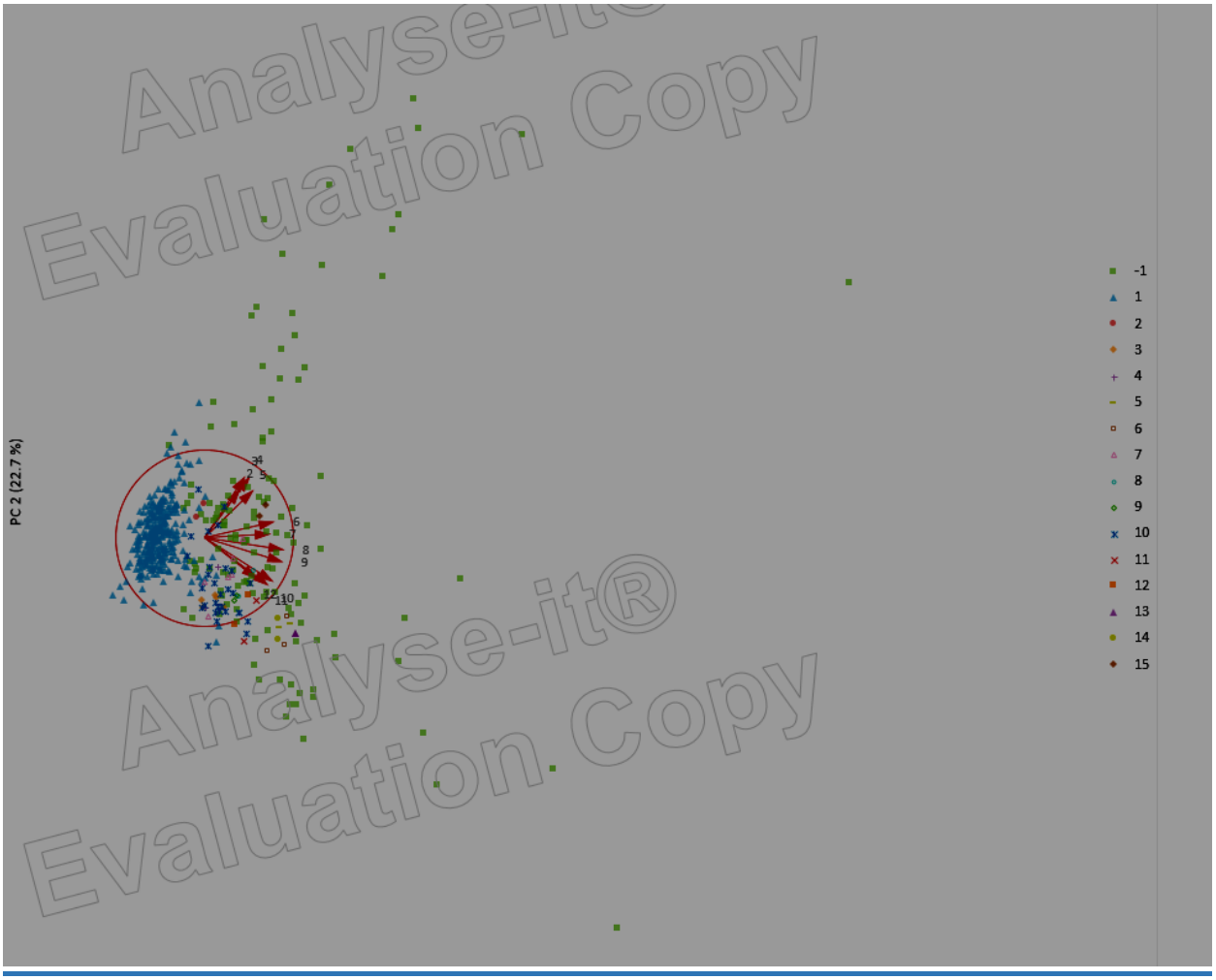


As -1 is noise, lot of genes are still considered as noise in spite of being close to the the genes assigned to a cluster so because of this its value of jaccard and rand decreases.

Dataset "Iyer.xlsx"

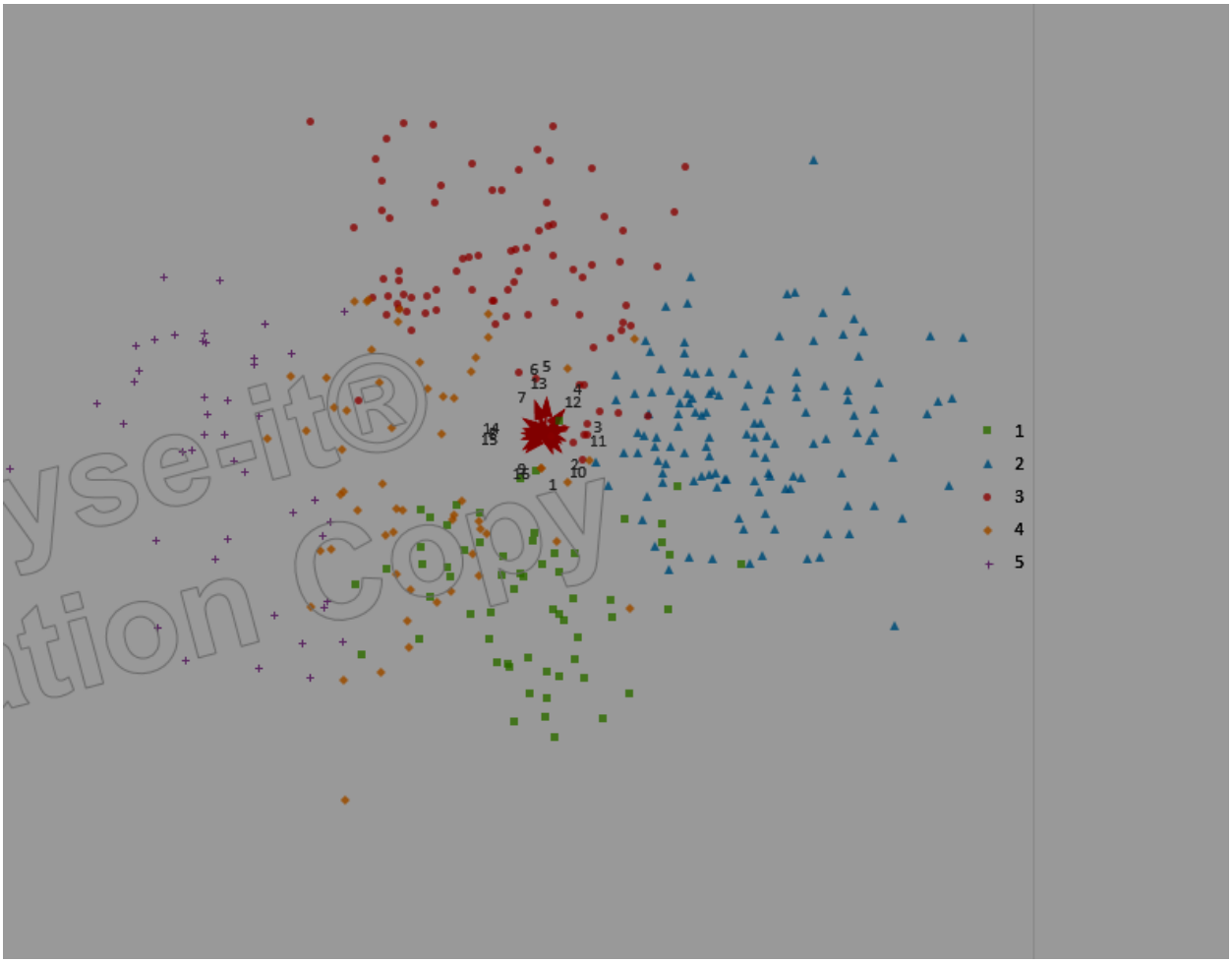


Above image is the depiction of cluster assignment excluding outliers so this looks like a fair clustering but it generates a lot of outliers as shown in following image.



As -1 is noise, lot of genes are still considered as noise in spite of being close to the the genes assigned to a cluster so because of this its value of Jaccard and Rand decreases. Another point to note here is that maximum number of genes are assigned to cluster 10 which also decrease the value of Jaccard and Rand.

K-Means:Map-Reduce
Dataset "cho.xlsx"



K-means map reduce is generating the exact same result as normal k means so it has exactly similar cluster assignment and PCA.

Analysis of Algorithms

KMeans Clustering	Hierarchical Agglomerative Clustering	Density Based Clustering: DBSCAN
Easier to implement.	Difficult to implement as compared to Kmeans.	Difficult to implement as compared to Kmeans.
Requires the number of clusters "k" as a parameter.	Gives just 1 cluster at the last. Hence needs to be sliced at a particular point to get clusters.	Infers number of clusters based on the data.
Usually the clusters are circular in shape..	Can handle non-elliptical shapes.	The clusters formed are of arbitrary shape.
Not sensitive to noise and outliers.	Sensitive to noise and outliers.	Is resistive to noise.
O(tkn) time complexity, t is the number of iterations.	$O(n^3)$ time complexity, n is the number of objects.	O(n * log n) time complexity.
Dependent on the input parameters.	Not dependent on the input parameters but we need to know at which point to slice the cluster dendogram.	Sensitive to input parameters, difficult to get correct parameters.

Conclusion

We have performed 3 types of clustering in this project on the two datasets; K-Means, Hierarchical (Min) and Density Based Clustering. K-Means is also implemented in Hadoop system using Map-Reduce method. On the basis of two datasets provided (”cho.xslx” and “iyer.xlsx”), it could be easily deduced using external index and PCA visualization that K-means is performing the best clustering on these datasets. If dataset is changed, other clustering methods could perform well as we have tried out with more datasets. Hierarchical clustering came as clear winner in terms of external index when another dataset was used. One of the major issue with K-means is the initial selection of centroids and number of centroids which very much manipulates the result. But in our case, it performed best among other algorithms. So it can be inferred that performance of clustering algorithms is very much dependent on the type of dataset available.