CSE 601: Data Mining and Bioinformatics Project-2
Clustering Algorithms

1. **K-Means**
2. **Hierarchical Agglomerative**
3. **Density based**
4. **GMM**
5. **Spectral**

Professor
Jing Gao

Submitted By
Akshara Santharam (50313950)
Swathi Ravindran (50314300)
Sai Kumar Aindla (50321372)

K-Means Clustering

K-Means divide N observations with P dimensions (variables) into K clusters so that the within-cluster sum of squares is minimized. Since the number of possible arrangements is enormous, it is not practical to expect the best solution. Rather, this algorithm finds a "local" optimum. This is a solution in which no movement of an observation from one cluster to another will reduce the within-cluster sum of squares. The algorithm may be repeated several times with different starting configurations. The optimum of these cluster solutions is then selected.

The generic steps include:

- Partitioning the given data points into given K clusters.
- Specifying the initial cluster centroids.
- For each data point x_i, calculating the distance between x_i and K centroids and (re)assigning them to the nearest K centroid cluster, until there is no change.
- Updating the current centroid of the cluster based on cluster assignment (cluster centroid--the mean of all points in the cluster).

Implementation

1. Load the file in a pandas data frame.

```
#Load the file in a Data Frame
file = 'cho.txt'
df = pd.read_csv(file,sep="\t", header=None)
```

2. Take the ground truth values and gene data from the loaded data frame into separate variables.
3. Give the number of clusters you want to give and number of iterations you want to run the algorithm.

```
#ground truth values
gr_true_values = df[1]

#Gene Data
gene_data = df.iloc[:,2:].to_numpy().astype(np.float)

#Niumber of clusters and iterations
print('Enter the number of clusters')
k = int(input())
print('Enter the number of iterations')
iterations = int(input())
centroids = Initialization(k, gene_data)
new_centroids = []
```

4. Enter the initial cluster centroid indices and assign each data point to their nearest given initial centroid index.

```
#Intial Assignment of clusters
def Initialization(k, gene_data):
    centroids = []
    for i in range(k):
        centroid_indices = int(input("Enter centroid gene data indices: "))
        centroids.append(gene_data[centroid_indices-1])
    centroids = np.asarray(centroids)
    return centroids
```

5. Loop through the given maximum number of iterations to reassign the clusters based on the Euclidean distance function.

```
#Assignment of clusters
def assignment(centroids, gene_data):
    clusters = []
    for i in range(gene_data.shape[0]):
        dist = []
        for cent_point in centroids:
            dist.append(calc_distance(cent_point,gene_data[i]))
        index = np.argmin(dist)+1
        clusters.insert(i,index)
    return clusters

#Euclidean distance
def calc_distance(x1,x2):
    return (sum((x1 - x2)**2))**0.5
```

6. Update the centroids of the cluster by taking the mean of the current data points cluster assignments in each iteration until there is no change in the cluster assignment.

```
#Updating Centroids
def update_centroids(assigned_cluster,centroids,gene_data):
    updated_centroids = []
    gene_data = pd.DataFrame(gene_data)
    assigned_cluster = pd.DataFrame(assigned_cluster,columns=['Cluster'])
    df_cluster = pd.concat([gene_data,assigned_cluster], axis=1)

    for cent in set(df_cluster['Cluster']):
        cur_cluster = df_cluster[df_cluster['Cluster'] == cent][df_cluster.columns[:-1]]
        mean = cur_cluster.mean(axis=0)
        updated_centroids.append(mean)
    return updated_centroids
```

```
#Algorithm
for i in range(iterations):
    assigned_cluster = assignment(centroids,gene_data)
    new_centroids = update_centroids(assigned_cluster,centroids,gene_data)

    if pd.DataFrame(centroids).equals(pd.DataFrame(new_centroids)):
        break
    centroids = new_centroids
```

7. 'assigned_cluster' variable will contain the final values of clusters assigned to each gene data index
8. Perform the PCA dimensionality reduction on the gene data using 2 components.

```
#PCA Part
pca_data = PCA(n_components=2).fit_transform(gene_data)
```

9. Plot the clusters using seaborn package by passing the pca_data and 'assigned_cluster' variable.

```
#Scatter Plots
plots(pca_data,assigned_cluster,file)
```

```
#Plotting the clusters
def plots(pca_data,assigned_cluster,file):
    df_new = pd.DataFrame({'y1':np.array(pca_data)[:,0], 'y2':np.array(pca_data)[:,1], 'clusters': assigned_cluster})
    sns.lmplot(x='y1', y='y2', data=df_new, fit_reg=False,hue='clusters',palette = 'dark', height=5,aspect = 1.5,legend
    plt.title(file)
```

10. Make both ground truth values and assigned_cluster values into a matrix and calculated external index (rand index and jaccard coefficient) as shown below.

```
if kmeans_matrix[i][j] == 1 and ground_matrix[i][j] == 1:
    m11 += 1
elif kmeans_matrix[i][j] == 0 and ground_matrix[i][j] == 0:
    m00 += 1
elif kmeans_matrix[i][j] == 0 and ground_matrix[i][j] == 1:
    m01 += 1
elif kmeans_matrix[i][j] == 1 and ground_matrix[i][j] == 0:
    m10 += 1
```

```
#Rand Index
rand_index = (m11 + m00)/(m11 + m00 + m01 + m10)

#Jaccard Coefficient
j_coeff = (m11)/(m11 + m10 + m01)
```

Advantages

1. Efficient: O(*tkn*), where *n* is number of objects, *k* is the number of clusters, and *t* is number of iterations. Normally, *k, t* << *n*
2. Easy to implement.
3. If the clusters are globular, closed pack will be formed.

Disadvantages

1. Empty clusters may appear.
2. Need to specify *K,* the number of clusters.
3. Local minimum– Initialization matters.
4. K-means has problems when clusters are of differing sizes, densities, irregular shapes.

Hierarchial Agglomerative Clustering:

It is a bottom up clustering method where the cluster consists of sub-clusters and sub-clusters. It starts with the single cluster with every single object and with successive iterations it merges the closest pair using some similarity criteria. Here we have used Euclidian distance as the distance measure.

The generic steps include:

- Assign each point as a separate cluster.
- Evaluate Euclidian distance between clusters.
- Construct distance matrix.
- Check the shortest value.
- Remove the pair and merge them.
- Evaluate all the distance and check the minimum and update the merged cluster matrix.

Implementation

1. The file path is set to the variable ‘file’ and it is converted into a np array.
2. Take the ground truth values and gene data into separate variables.
3. Input the number of clusters as in input.
4. Distance matrix is calculated from a user defined function that computes Euclidean distance of a point with other points.
5. The new_cluster() method is called, which will return the final cluster list of all the points.
6. By looping to the number of clusters, we perform Hierarchical clustering with Min based approach.
    - Initially we find the minimum value from the distance matrix and find the cluster those points belong to.
    - assign_cluster(clusters, gene_index): This method is used to return the index of the cluster that point belongs to.

- Once the index of the cluster of the points whose distance was minimum from the distance matrix was found, the next step is to merge those clusters.
- Once the clusters are merged, the next step is to update the distance matrix.
- The matrix is updated based on min-based approach.
- Once the loop is finished to the number of clusters, it will return a return the new cluster list

7. Perform the PCA dimensionality reduction on the gene data using 2 components.
8. Plot the clusters using seaborn package by passing the pca_data and 'assigned_cluster' variable.

```
#Plotting the clusters
def plots(pca_data,assigned_cluster,file):
    df_new = pd.DataFrame({'y1':np.array(pca_data)[:,0], 'y2':np.array(pca_data)[:,1], 'clusters': assigned_cluster})
    sns.lmplot(x='y1', y='y2', data=df_new, fit_reg=False,hue='clusters',palette = 'dark', height=5,aspect = 1.5,legend
    plt.title(file)
```

9. Make both ground truth values and assigned_cluster values into a matrix and calculated external index (rand index and jaccard coefficient) as shown below.

```
for i in range(len(data)):
    for j in range(len(data)):
        if ground_truth[i]==ground_truth[j]:
            if final_cluster_vals[j]:
                tp = tp + 1
            else:
                fn = fn +1
        elif ground_truth[i]!=ground_truth[j]:
            if final_cluster_vals[i]==final_cluster_vals[j]:
                fp = fp +1
            else:
                tn = tn+1
return tp,tn,fp,fn
```

```
#External_indices
tp,tn,fp,fn = get_count(points,final_cluster_vals,ground_truth)

#Rand Index
rand_index = (tp+tn)/(tp+tn+fp+fn)

#Jaccard Coefficient
j_coeff=(tp)/(tp+fp+fn)
```

Advantages

1. Do not have to assume any particular number of clusters.
2. Meaningful taxonomies.
3. They are very well-developed ideas and have been in use for years over a varied type of data.

Disadvantages

1. Time complexity is quadratic O(n^2), so impractical for very big datasets.
2. Our metric for clustering, MIN is susceptible to noise.
3. Once a decision is made to combine 2 clusters, it cannot be undone. D.
4. No objective function is directly minimized.

Density Based Clustering

It is used to identify clusters of any shape in a data set containing noise and outliers. It is based on intuitive notion of cluster and noise and hence it is derived as human intuitive clustering. The key idea is of each point of a cluster, the neighborhood contains at least minimum number of

specified points. For minimum distance, we calculate the Euclidian matrix and compare the value with eps.
The generic steps included:

- Specify minimum number of points and epsilon value (distance limit) for the clusters.
- For each point assign neighbors by checking the Euclidean distance with epsilon value.
- For each neighbor check if the points lie in the cluster by checking the min_points and hence expand the cluster until no new points discovered.
- Other points can be considered as noise.

Implementation

1. The file path is set to the variable 'file' and it is converted into a np array.
2. Take the ground truth values and gene data into separate variables.
3. Input the values user epsilon and minimum points.
4. Distance matrix is calculated from a user defined function that computes Euclidean distance of a point with other points.
5. Call density_based_scan(gene_data, eps,min_points,seen,end_point,dist_matrix):
   - For every point that is not visited yet, i.e. by looping to the length of points, get the point's neighbors within the epsilon distance we obtained from the user.
6. Function compute_region(qp_index,gene_data,eps,dist_matrix) will return the neighbor points from the distance matrix calculated in the second step.
   - Once the neighbor points are obtained, if the number of points is greater than the min_points, we know it's a core point, add it to a cluster and call the find_cluster() function
   - If it is not a core point, i.e. the number of neighbor points is less than the min_points, it is classified as a noise.
7. Function find_cluster(corepoint_index,gene_data, adj,cluster,eps, min_points end_point, seen, dist_matrix) will perform below tasks :
   - For every neighbor point obtained in the above function, if the point is not in visited list, get its neighbors by calling the compute_region() function.
   - If the number of points is greater than the min_points, the neighbor points are merged with the neighbor points, i.e the points are added in the same cluster.
   - Once all the points have been finished visiting, then we obtain the cluster list of all points, which are maintained in end_point.
8. Perform the PCA dimensionality reduction on the gene data using 2 components.
9. Plot the clusters using seaborn package by passing the pca_data and 'assigned_cluster' variable

```
#Plotting the clusters
def plots(pca_data,assigned_cluster,file):
    df_new = pd.DataFrame({'y1':np.array(pca_data)[:,0], 'y2':np.array(pca_data)[:,1], 'clusters': assigned_cluster})
    sns.lmplot(x='y1', y='y2', data=df_new, fit_reg=False,hue='clusters',palette = 'dark', height=5,aspect = 1.5,legend
    plt.title(file)
```

10. Make both ground truth values and assigned_cluster values into a matrix and calculated external index (rand index and jaccard coefficient) as shown below.

```
for i in range(len(data)):
    for j in range(len(data)):
        if ground_truth[i]==ground_truth[j]:
            if final_cluster_vals[j]:
                tp = tp + 1
            else:
                fn = fn +1
        elif ground_truth[i]!=ground_truth[j]:
            if final_cluster_vals[i]==final_cluster_vals[j]:
                fp = fp +1
            else:
                tn = tn+1
return tp,tn,fp,fn
```

```
#External_indices
tp,tn,fp,fn = get_count(points,final_cluster_vals,ground_truth)

#Rand Index
rand_index = (tp+tn)/(tp+tn+fp+fn)

#Jaccard Coefficient
j_coeff=(tp)/(tp+fp+fn)
```

Advantages

1. Resistant to noise.
2. Can handle clusters of different shapes and size.
3. Doesn't require to specify the number of clusters, as opposed to k-means.
4. It can even find a cluster completely surrounded by (but not connected to) a different cluster.
5. Requires just 2 parameters and is mostly insensitive to the ordering of the points in the database.

Disadvantages

1. Cannot handle varying densities.
2. Sensitive to parameters – hard to determine the correct set of parameters.
3. Algorithm is not entirely deterministic.
4. The quality of algorithm depends on the distance measure used in the compute_region function. Mostly Euclidean distance is used. But for high dimensional data, this measure is rendered almost useless due to curse of dimensionality.

Gaussian Mixture Model

Gaussian mixture models can be used to cluster unlabeled data in much the same way as k-mean clustering. It is a probabilistic grounded way of doing soft clustering. The parameters of the model are mean and covariance. Gaussian distribution parameters are estimated for each cluster and weight of a cluster. Once the parameters are estimated, probability of belonging to each cluster is calculated.

The generic steps are:

1. The number of sources or clusters that has to fit the data had been decided.
2. Mean, fraction and covariance has been initialized for each cluster.
3. Expectation and maximization consist of two steps:
    - In the first step, the expectation step or E-step, it consists of calculating the posterior probability for each point belonging to each of the K clusters.

- In the second step, the maximization step, the parameters (mu, sigma and phi) is updated based on the posterior probability.
- The entire iterative process is repeated until the algorithm converges.

Implementation

1. In the first step, mean, covariance and prior probability for the K Gaussian distribution functions is initialized.
2. Two steps are performed in each iteration
   - E Step:
     In the estimation step, posterior probability for each point to each of the K clusters is calculated.
   - M Step:
     In the maximization step, parameters (mu, sigma and phi) are updated based on the posterior probability. The iteration is stopped, when the updated parameters goes below the convergence threshold. Convergence can be checked by using the log likelihood function.
3. For each point, the respective cluster is chosen which gives the maximum posterior probability at the end of all iterations.
4. After the clusters is found, we plotted these clusters by calling the PCA() method. The plotting is done by using a library seaborn.

Advantages

1. Gives probabilistic cluster assignments.
2. Have probabilistic interpretation.
3. Can handle clusters with varying sizes, variance

Disadvantages

1. Initialization matters.
2. Choose appropriate distribution
3. Overfitting issues

Spectral Clustering

Spectral clustering takes a graph-based approach to encode the information about the local neighborhood. It is an unsupervised machine learning algorithm. In this type pf clustering, what points fall under which cluster. The similarity graph is used to store the similarity distance between points where the weight of the edges represents similarity and each point is considered as a vertex of the graph.

Gaussian kernel is defined as follows:

$$w_{ij} = \exp(-\| x_i - x_j \|^2 / \sigma^2)$$

The generic steps are:

1. Construct a similarity graph. A similarity matrix is a n*n matrix, where n represents the number of data points, and the value in each cell I,j is the distance given by the gaussian kernel.
2. Determine the adjacency matrix W, Degree matrix D and the Laplacian matrix L.
   Degree matrix is an n*n diagonal matrix, where n represents the number of data points, where the value of each cell i, where i is the sum of row i.
   Laplacian matrix is an n*n symmetric matrix is derived by subtracting the similarity matrix from the degree matrix.
3. Computer the eigen vectors of the Laplacian matrix.
4. We train a K means model using the second smallest eigen vector as input and can be used to classify the data.

Implementation

1. Similarity matrix is constructed from the given dataset. Similarity is used as a metric that determines how close two points are in our space. Gaussian kernel is used to determine the similarity.
2. Degree matrix is obtained from similarity matrix. Degree matrix is a diagonal matrix consisting of the sum of weights into each vertex.
3. Laplacian matrix is determined, and it is obtained by subtracting similarity matrix from the degree matrix.
4. In the Laplacian matrix, eigenvalues and eigenvectors is calculated.
5. Taking the number of clusters to be K and by taking the smallest K eigenvalues and the corresponding eigenvectors, we reduce the number of points * number of features space to number of points * eigenvalues.
6. In the reduced space, apply KMeans Algorithm and label the points to their corresponding clusters.

Advantages

1. It can find clusters of arbitrary shapes.
2. Spectral clustering works well for anisotropic data
3. It is easy to implement
4. It gives good clustering results
5. It does not make strong assumption on the statistics of clusters

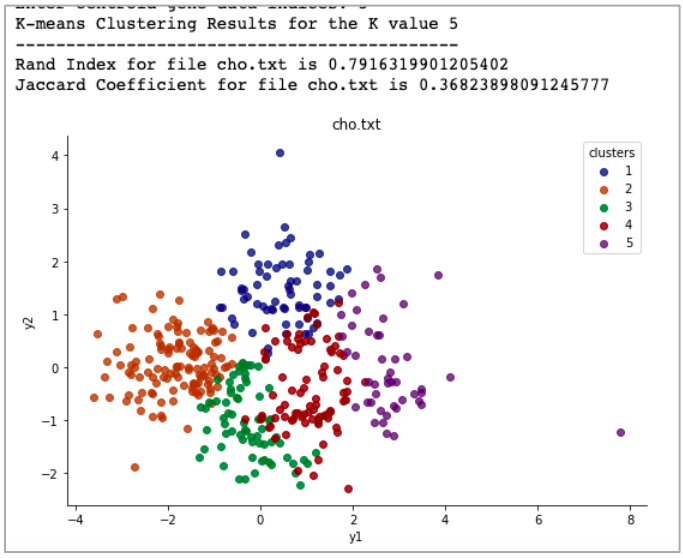
Disadvantages

1. Noisy dataset can affect the performance.
2. Computing eigen vectors is very expensive when dealing with large datasets.
3. Performance is slower than K-Means.

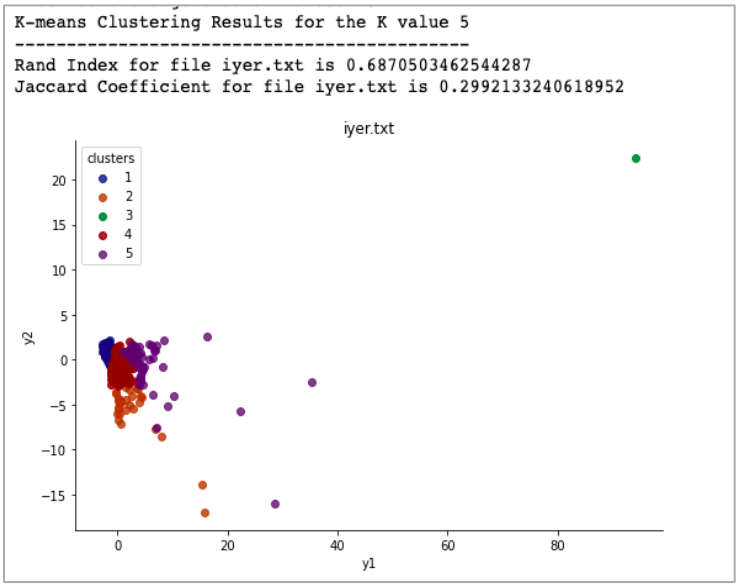
Scatter Plots

K-Means Results

1. File: ‘cho.txt’
2. K = 5, iterations = 10, initial centroids = (1,2,3,4,5)
3. Rand index = 0.79163, Jaccard Coefficient = 0.36823



1. File: ‘iyer.txt’
2. K = 5, iterations = 10, initial centroids = (1,2,3,4,5)
3. Rand Index = 0.63705, Jaccard Coefficient = 0.29921

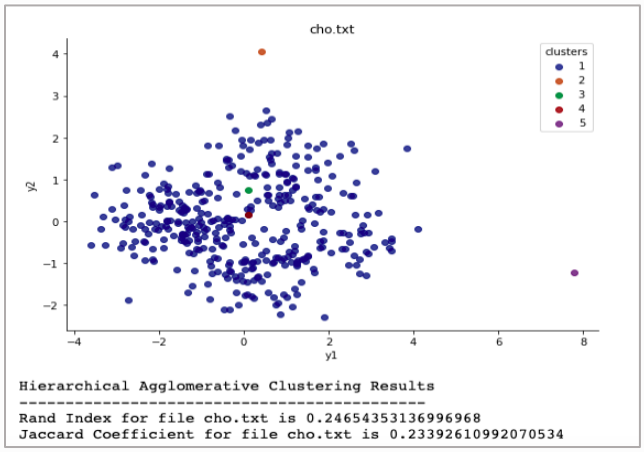


Analysis of K-Means:

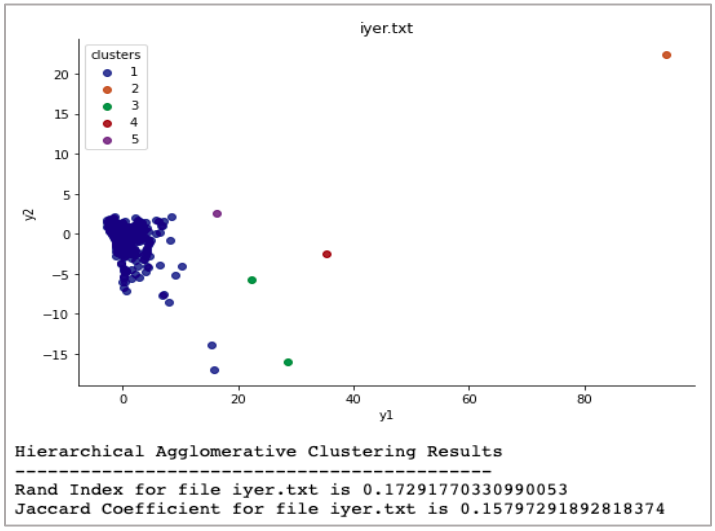
1. K-means has faster computation speed in comparison with other algorithms.
2. For globular dataset like 'iyer.txt', K-means doesn't seem to perform better clusters because of the nature of globular.
3. But, for the dataset 'cho.txt', K-Means seems to cluster and it can be seen through using the rand index and jaccard coefficient.

Hierarchical Clustering Results

1. File: 'cho.txt'.
2. K = 5
3. Rand Index = 0.24654, Jaccard Coefficient = 0.23392



1. File: 'iyer.txt'.
2. K = 5
3. Rand Index = 0.172917, Jaccard Coefficient = 0.1579729

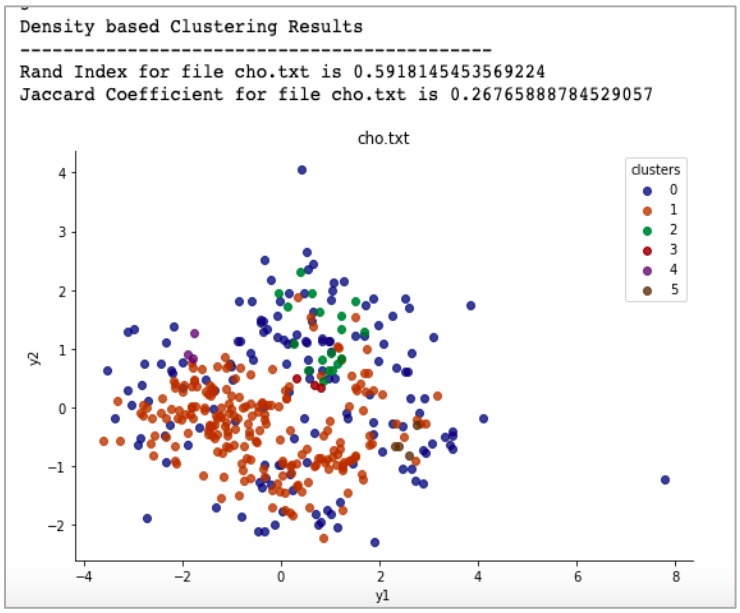


Analysis of Hierarchical Clustering:

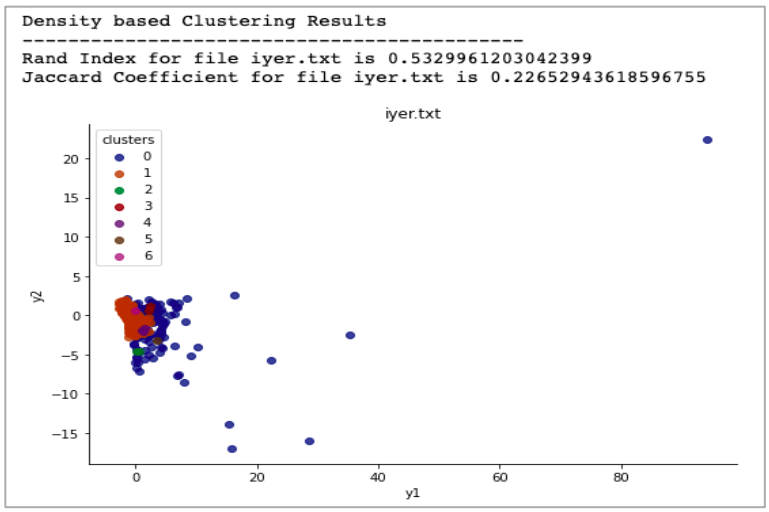
1. From the plot for dataset "iyer.txt", we can observe that it is capable of clustering points belonging to non-elliptical shaped distributions. This occurs as a result of using min approach to determine the inter-cluster distance
2. A few points in the central region of the large cluster do not belong to the large cluster as a result of dimensionality reduction. These points may be at a large distance from the large cluster which aren't primarily represented after PCA
3. The plots are adversely influenced by outliers which cause the Jaccard of HAC to be lower than that of K-Means.

Density based Clustering Results

1. File: 'cho.txt'
2. Epsilon = 1.13
3. Min_points = 3
4. Rand Index = 0.5918145, Jaccard Coefficient = 0.2676588



1. File: 'iyer.txt'
2. Epsilon = 1.13
3. Min_points = 3
4. Rand Index = 0.5329961, Jaccard Coefficient = 0.2265294

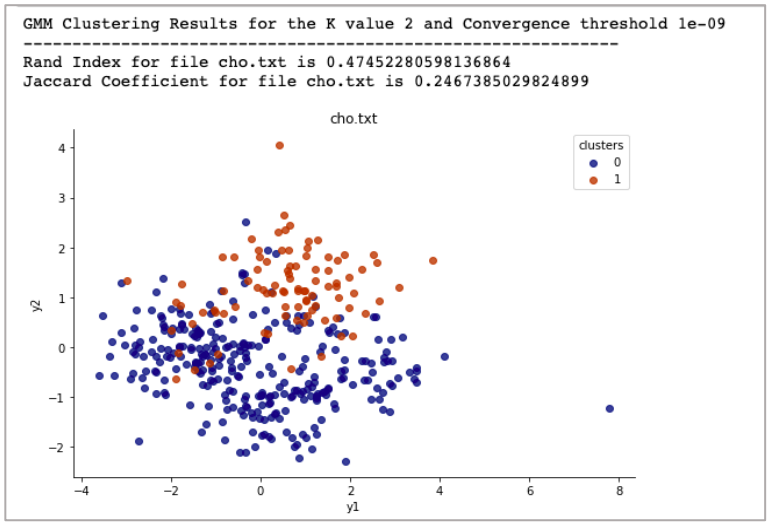


Analysis of Density based clustering

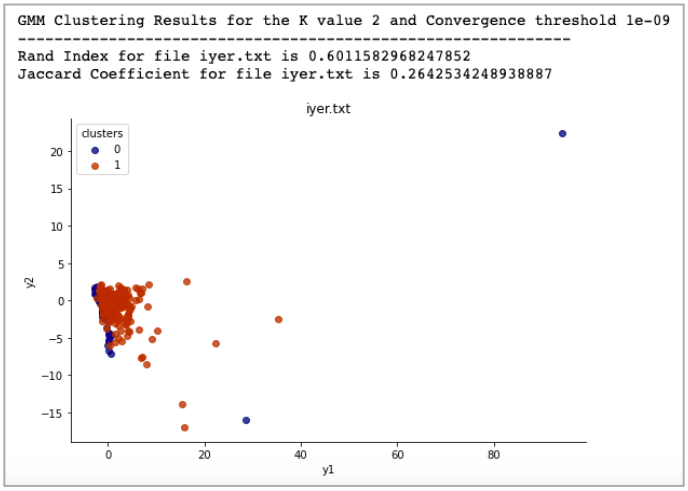
1. Dbscan is the only one which explicitly labels a set of points as noise based on their distance from the data distribution.
2. It is observed that the clusters formed are of various shapes and sizes, which is advantageous in many cases where the data doesn't belong to any regularly shaped distributions.
3. Dbscan performs better than agglomerative clustering in the dataset "iyer.txt", which is intuitive as the distribution of the data points is irregular for "iyer.txt".

GMM Clustering results

1. File: 'cho.txt'
2. K = 2, convergence threshold = 1e-9
3. Rand Index = 0.4745, Jaccard Coefficient = 0.246738



1. File: ‘iyer.txt’
2. K = 2, convergence threshold = 1e-9
3. Rand Index = 0.60115, Jaccard Coefficient = 0.264253

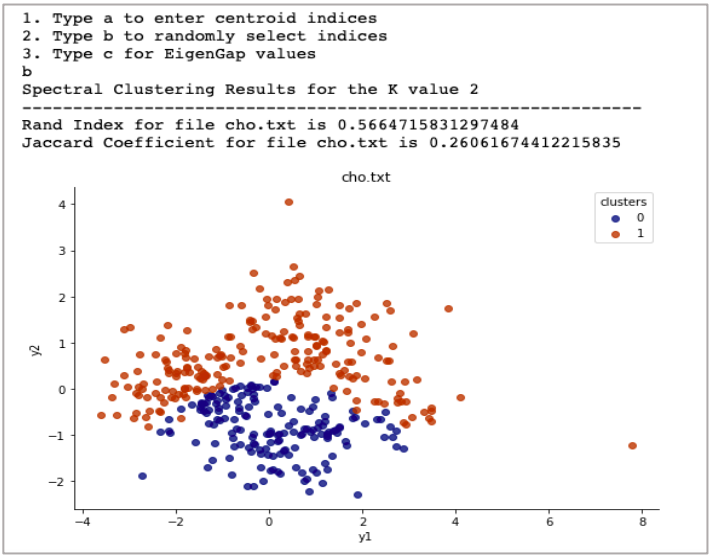


Analysis of GMM Clustering:

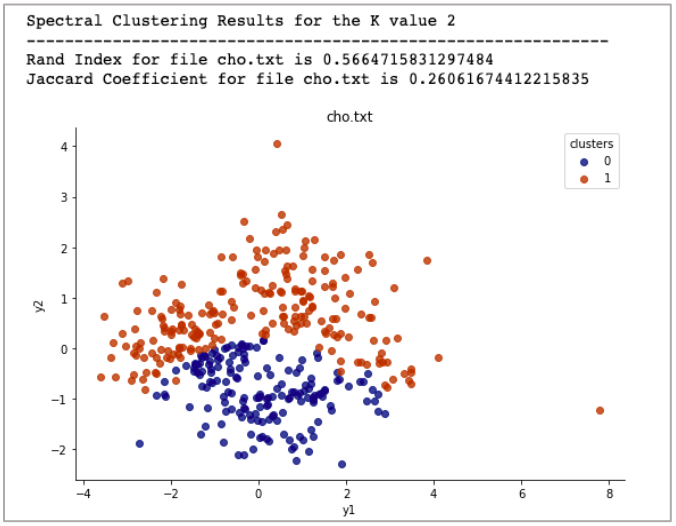
1. The result of the clustering algorithm is that it is associated with the Gaussian model.
2. In GMM, binary assignment of each point to a particular cluster is not followed. Instead Gaussian distribution is used to determine the likelihood of a point that is belonging to a particular cluster.

Spectral Clustering Results

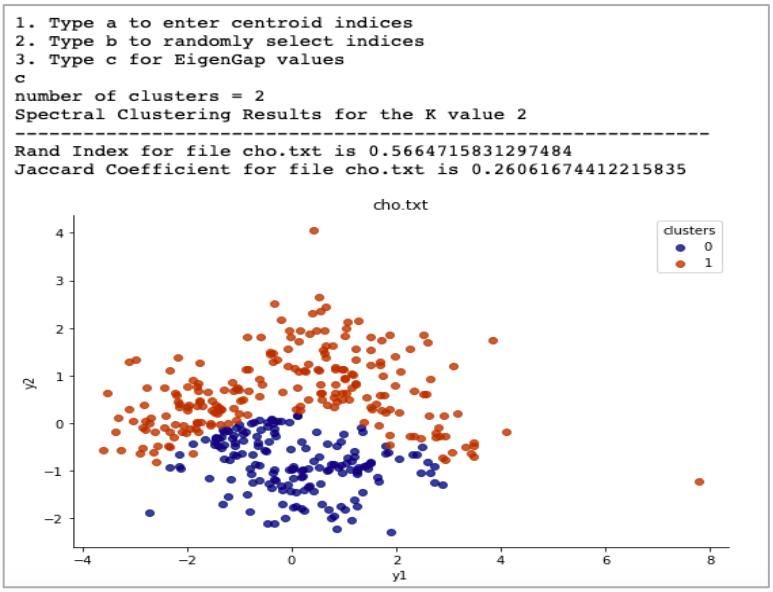
1. File: ‘cho.txt’
2. For k = 2, sigma = 2, for the randomly chosen centroids using K-Means
3. Rand Index = 0.56647, Jaccard Coefficient = 0.260616



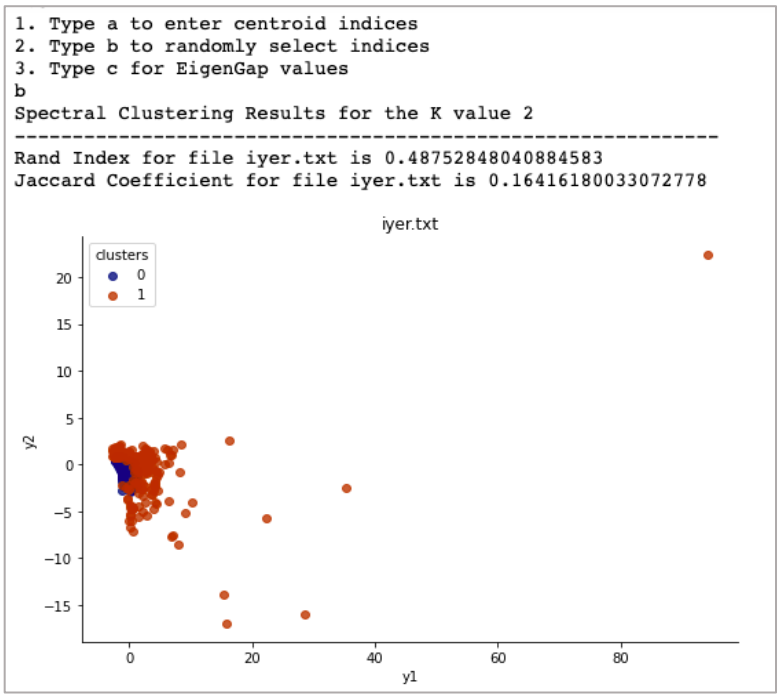
1. File: ‘cho.txt’
2. For k = 2, sigma = 2 and the initial data Ids = (2,3)
3. Rand Index = 0.56647,
4. Jaccard Coefficient = 0.260616



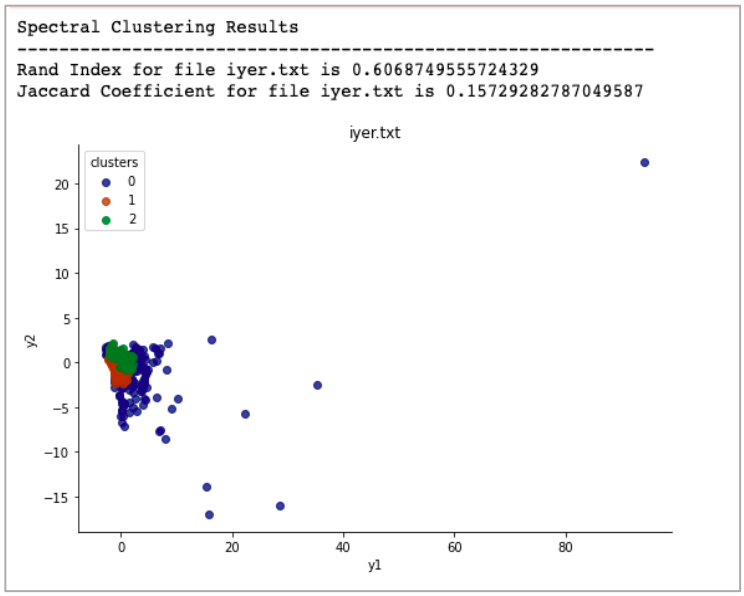
1. File: ‘cho.txt’
2. For k = 2, sigma = 2 and using the values of eigen vectors
3. Rand Index = 0.5664715,
4. Jaccard Coefficient = 0.260616



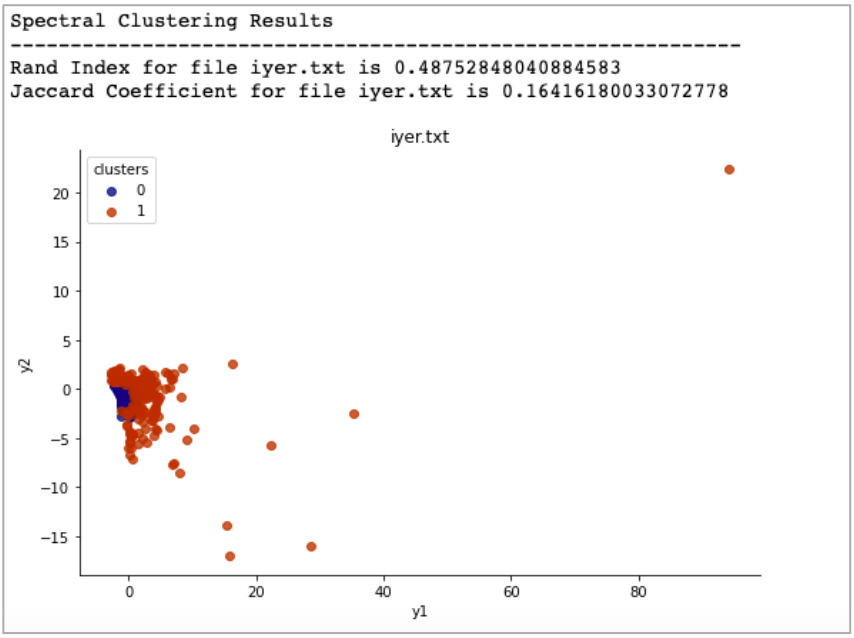
1. File: ‘iyer.txt’
2. For k = 2, sigma = 1.5, for the randomly chosen centroids using K-Means
3. Rand Index = 0.48752
4. Jaccard Coefficient = 0.16416



1. File: ‘iyer.txt’
2. For k = 3, sigma = 1.5 and the initial data Ids = (1,2,3)
3. Rand Index = 0.60687
4. Jaccard Coefficient = 0.15729



1. File: ‘iyer.txt’
2. For k = 2, sigma = 1.5 and using the values of eigen vectors
3. Rand Index = 0.48752
4. Jaccard Coefficient = 0.16416



Analysis of Spectral Clustering

Using Gaussian kernel, we are calculating the similarity of points. Value of sigma is chosen that will generate good clustering.

1. The proximity of the two data points can be exponentially weighed and decays exponentially with the two points distance.
2. Better clustering and better Random index and Jaccard value can be noticed as we increase the sigma.
3. Similarity matrix can influence the result of spectral clustering.
4. Accurate clustering result can be determined by choosing proper similarity function and choosing proper centroids.