CSE 601: DATA MINING
&
BIO INFORMATICS

FALL 2020

PROJECT 2: CLUSTERING ALGORITHMS

Team

Hemant Koti (50338178)

Sai Hari Charan Barla (50336868)

Shravya Pentaparthi (50337027)

Metrics used to evaluate clustering models:

In our experiment we are using Rand and Jaccard indices to compare the results of our clustering models.

The terms related to the Rand and Jaccard indices are:

- **true positive**: model *correctly* predicts the *positive* class.
- **true negative**: model *correctly* predicts the *negative* class.
- **false positive**: model *incorrectly* predicts the *positive* class.
- **false negative**: model *incorrectly* predicts the *negative* class.

Jaccard Index:

Jaccard similarity coefficient, is used in understanding the similarities between sample sets. The measurement emphasizes similarity between finite sample sets, and is formally defined as the size of the intersection divided by the size of the union of the sample sets. It is computed using the below formula:

Jaccard Index = (true positive) / (true positive + false negative + false positive)

Rand Index:

Rand index as a measure of the percentage of correct decisions made by the algorithm. Rand index is related to the accuracy. It is the metric used to evaluate the quality of clustering technique.

Random coefficient = (true positive + true negative) / (true positive + true negative + false positive + false negative)

Density Based Clustering (DBSCAN):

DBSCAN is a well-known density-based clustering algorithm (of Applications with noise) used in data mining and Machine Learning. The number of clusters need not be given as a parameter in case of DBSCAN, instead it infers the number of clusters based on the data and it discovers clusters of arbitrary shape.

The 2 parameters that are needed to be given in case of DBSCAN are:
1) **Eps ε:** The radius of our neighbours around a data point p.
2) **minPts**: The minimum number of data points needed in a neighbour to define a cluster.

Points are classified into the following using the above three parameters:

- **Core points**: Points having more than minimum number of points in its neighbourhood enough – within a distance of given radius – are present in the same cluster.
- **Border points**: Points having less than minimum number of points in its neighbourhood, but still present in neighbourhood of other core points. Any border point that is close enough to a core point is present in the same cluster as the core point.
- **Noise**: Any point that is not a core or border point and also having less than minimum number of points in its neighbourhood. Noise are not present in any cluster.

Density-reachable: An object q is directly density-reachable from object p if q is within the ε-neighborhood of p and p is a core object.

Density-connectivity: An object p is density-connected to object q with respect to ε and minimum number of points if there is an object o such that both p and q are density-reachable from o with respect to ε and minimum number of points.

Implementation of the code:

Step 1) The file is obtained from the command line from the user.

Step 2) Once the file is obtained, it is converted into a np array using np.asarray() function and stored in filledata. From this filedata, the first 2 columns are removed as they are Id and truthlabels,So only attributes are obtained and it is assigned to data

Step 3) The epsilon (radius) value and the minimum no. of points are taken as user input and passed as arguments to the dbscan function

Step 4) Two matrixes are created visited, finalCluster with dimensions (data.shape[0],1) and initialized to 0 using numpy zeros.

Step 5) Initially distance matrix for the points are computed by calling compute_distancematrix function. It computes a distance of a point with other points and stores it in the form of a matrix.

Step 6) **dbscan(data,epsilon,min_points)**

- The next step is to call the dbscan function. All the necessary arguments are passed to the function, that are mentioned above.
- For every point that is not visited yet, i.e by looping to the length of points, get the point's neighbours within the epsilon distance we obtained from the user.

Step 7) **neighbours (query_point_index):**

- This function is used to return the neighbour points from the distance matrix calculated above.
- Once the neighbour points are obtained, if the number of points is greater than the min_pts, we know it's a core point, add it to a cluster and call the expandCluster() function
- If it is not a core point, i.e. the number of neighbour points is less than the min_pts, it is classified as a noise.

Step 8) **expandCluster(corepoint_index,points, neighbour_pts,cluster):**

- For every neighbour point we obtained above, if the point is not in visited list, get it's neighbours by calling the regionQuery() function.
- If the number of points is greater than the min_pts, the neighbour points are merged with the neighbour points, i.e the points are added in the same cluster.
- Once all the points have been finised visiting, then we obtain the cluster list of all points, which are maintained in cluster_final.

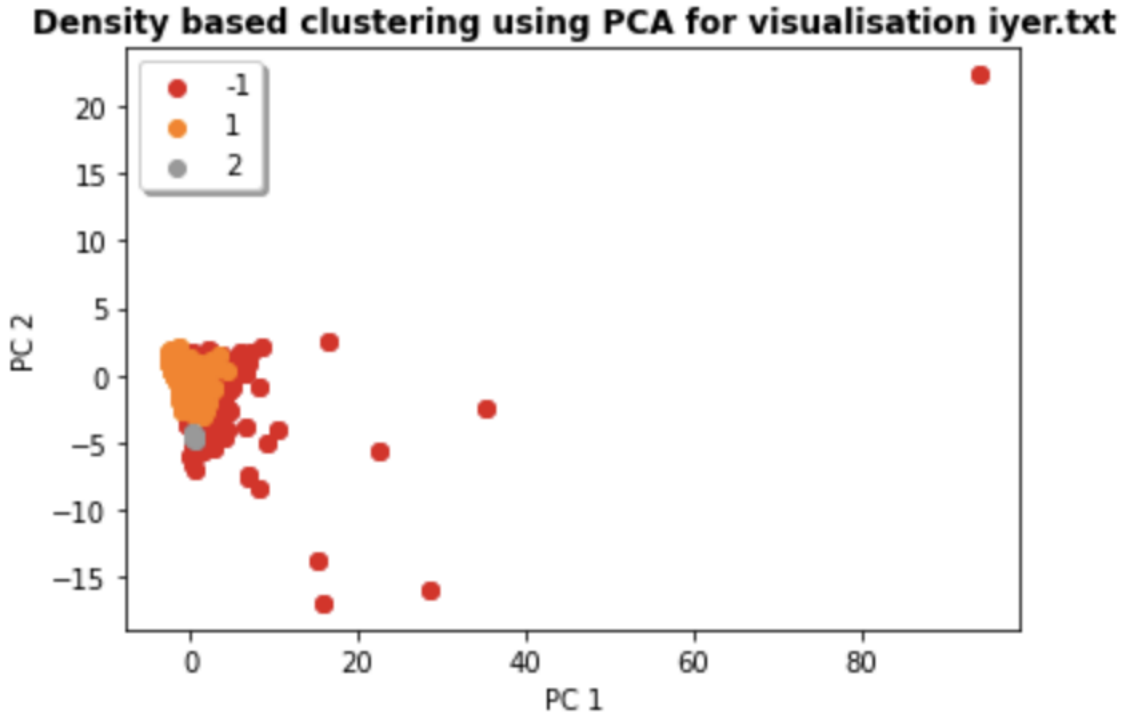
Step 9) Once the cluster_final is obtained, Those clusters are plotted by calling the PCA() method provided by skylearn. The plotting is done by using matplotlib.pyplot, by setting all the legends and attributes.

Step 10) Two coefficients(Jaccard and Rand index) are calculated by calling the calculateJacRand() function

Visualisation:

File Name: iyer.txt

Here, the colors correspond to the Cluster ID assigned by the algorithm. After experimenting with varied values of Epsilon and MinPts, we have clustered the data by fixing the value of Epsilon being 1.42 and the value of Minpts to 3.



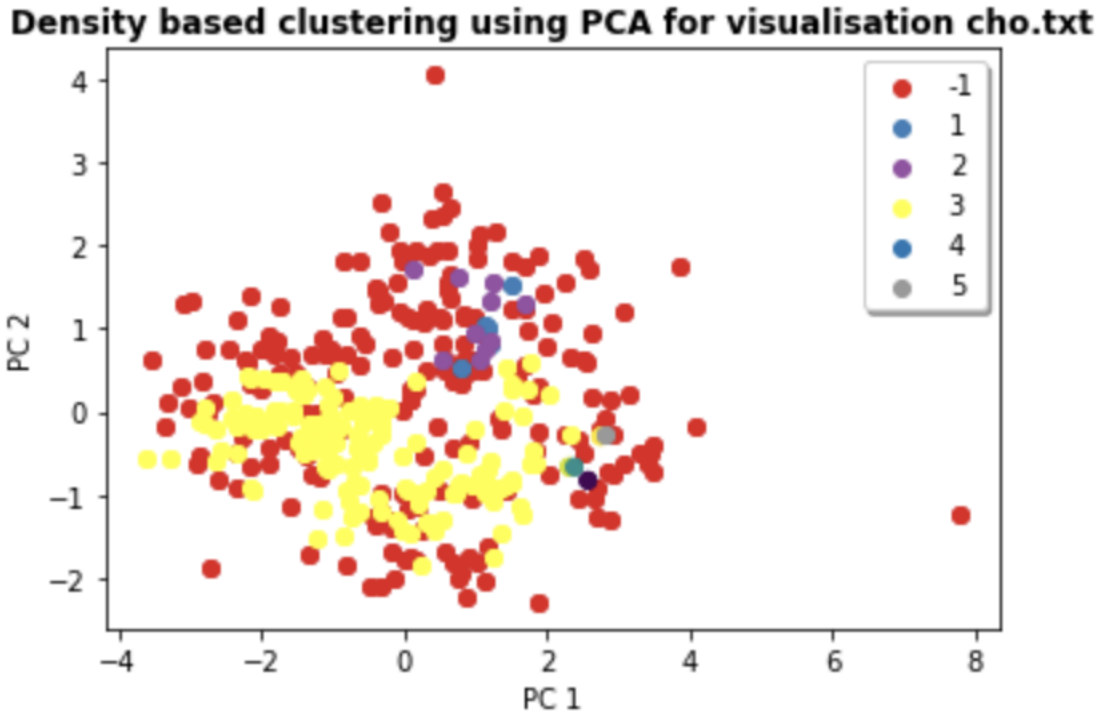
With the specified values of Epsilon and MinimunPoints, after the implementation of DBCSN algorithm, the data set is clustered.

The value of jaccard co-efficient and rand index for the current input parameters is:

Jaccard_coefficient: 0.20382462942723922
rand_index: 0.4576282600481127

File Name: cho.txt

Here, the colors correspond to the Cluster ID assigned by the algorithm. After experimenting with varied values of Epsilon and MinPts, we have clustered the data by fixing the value of Epsilon being 1.02 and the value of Minpts to 3.



With the specified values of Epsilon and MinimunPoints, after the implementation of DBCSN algorithm, the data set is clustered.

The value of jaccard co-efficient and rand index for the current input parameters is:

Jaccard_coefficient: 0.203937592867756
rand_index: 0.6460684876669074

RESULT EVALUATION:

- DBSCAN algorithm assigns the labels to each point based on the number of neighbours that reachable within the epsilon distance.
- Algorithm is capable of identifying the noise/outliers in the data.
- As we know that the DBSCAN classifies the points based on the number of neighbours that falls within a particular radius. Both of these are given as inputs to the algorithm minimumpoints and eps value.
- The time complexity of the algorithm is O(n2) where n is the number of data points when the dimensionality of data is low.
- We ran the code for different values of eps and min_pts and identified that the algorithm is sensitive to the parameters.
- n case of Cho.txt, visualisation of clusters is very good i.e. it was able to identify well-spaced clusters and we can differentiate it easily.

- In case of iyer.txt, the points look closer to each other along the principal axis, however in reality they could be farther apart from each other.
- From the graph we can see that it handles outliers better unlike hierarchical clustering.

ADVANTAGES:

- It detects arbitrary shaped clusters.
- It does not require to specify number of clusters in the beginning.
- It can handle clusters of different shapes and sizes
- It is robust towards outliers i.e it has high resistance towards noise.

DISADVANTAGES

- It is sensitive to parameters minimumpoints and eps.
- It fails to identify the cluster if the dataset is too parse.
- It is tricky when the density of the dataset varies.
- It cannot handle varying densities

Hierarchical Agglomerative clustering with Min approach:

- Agglomerative Clustering is a bottom-up approach starting with each point in a separate cluster, repeatedly joining the most similar pair of clusters and updating the similarity of the new cluster to other clusters until there is only one cluster
- We use single linkage i.e. we consider the distance of the closest pair of data objects belonging to different clusters
- In single-link clustering or single-linkage clustering, the similarity of two clusters is the similarity of their most similar members
- This single-link merge criterion is local

We used Single Linkage for the implementation,

Generic Algorithm:

1. Compute the proximity matrix.
2. Let each data point be a cluster.
3. Repeat: Merge two closest clusters and update the proximity matrix.
4. Until only a single cluster remains.

For the given dataset:

1. For the given objects and attributes in the input data, calculate the distance matrix using Euclidean distance.
2. On each iteration, Identify the least value i.e. minimum value from the distance matrix.
3. Once the minimum value is found, merge the object id's for which the value is minimum and update the distance matrix based on the new cluster.
4. Repeat the above steps, until it comes to only one cluster.

Implementaion:

We implemented Single-linkage i.e. the similarity of the closest pair ,

Step 1) The file is obtained from the command line from the user.

Step 2) Get the number of clusters from input from the command line user.

Step 3) Once the file is obtained, it is converted into a np array using np.asarray() function and stored in filledata. From this filedata, the first 2 columns are removed as they are Id and truthlabels,So only attributes are obtained and it is assigned to data.

Step 4) Initially distance matrix for the points are computed by calling compute_distancematrix function. It computes a distance of a point with other points and stores it in the form of a matrix.

Step 5) mergeCluster(), findCluster() functions has been used to assign the cluster of the data points after each iteration. The properties of those functions are very much like Union Find Data structure.

Step 6) Initially, each point is stored as single cluster. Then we start merging the clusters till only the given number of clusters remain.

Step 7) For merging, first the minimum distance that is greater than 0 is identified between two points from the distance matrix and then those points are merged by calling the mergeCluster method.

Step 8) The distance matrix is recomputed for remaining points using minimum distance between two of the merged points. For example if points 3 and 10 are merged in this step. Distance of all other points will be calculated w.r.t to cluster (3,10) using, d((3,10), P) = min(d(3,P),d(10,P). The distance matrix is recomputed and updated every time two points or clusters are merged.

Step 9) Finally labels are assigned based on the cluster to which the data point belongs.

Step 10) Once the labels are obtained, Those clusters are plotted by calling the PCA() method provided by skylearn. The plotting is done by using matplotlib.pyplot, by setting all the legends and attributes.

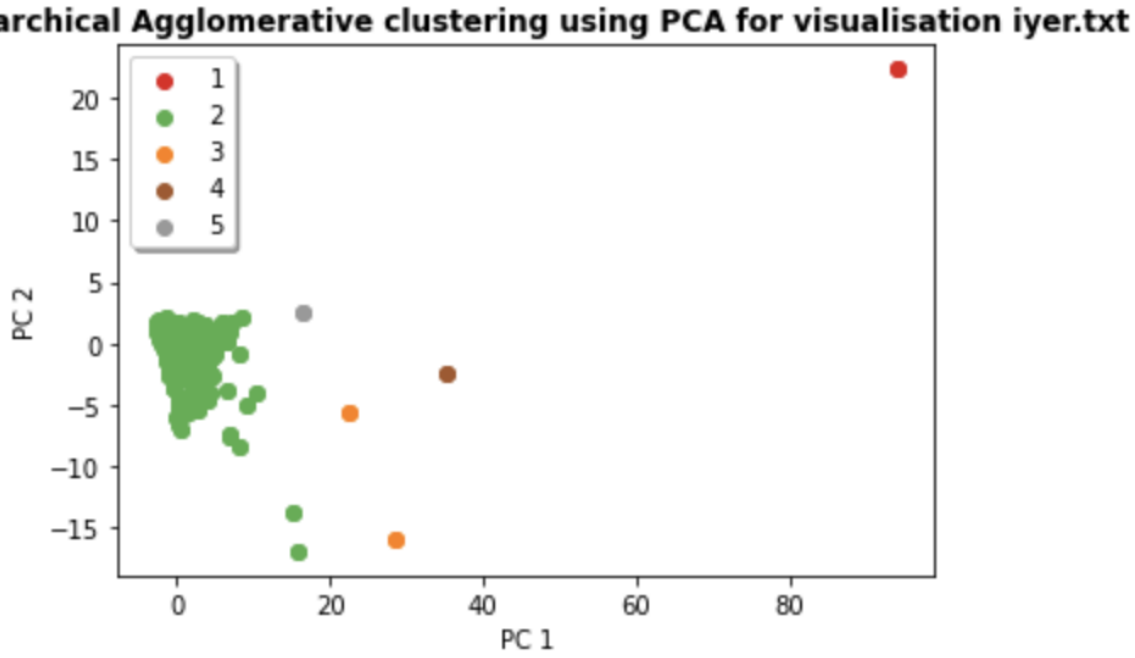
Step 11) Jaccard Coefficient and rand index are computed by calling calculateJacRand() function.

Visualisation:

File Name: iyer.txt
Clusters: 5

The obtained Jaccard Coefficient and Rand Index values are:

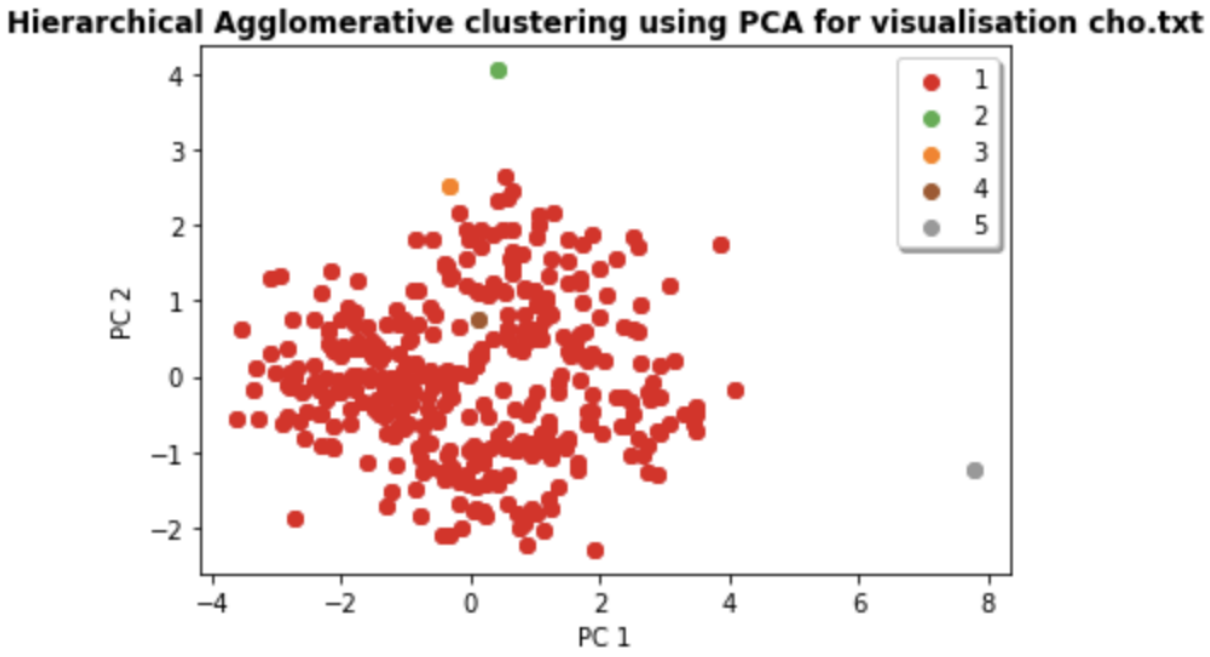
Jaccard Coefficient: 0.15647222380925174
Rand Index: 0.17144364339722173



File Name: cho.txt
Clusters: 5

The obtained Jaccard Coefficient and Rand Index values are:

Jaccard Coefficient: 0.22839497757358454
Rand Index: 0.24027490670890495



Result Evaluation:

- We used Single Linkage Hierarchical agglomerative clustering, which is used to group clusters in bottom up fashion, at each step combining two clusters that contain the closest pair of elements not yet belonging to the same cluster as each other. Thus points closest to each other first form a cluster.
- In visualisation, we can see the clusters are not evenly distributed. It looks like a one dominant cluster. The points which are farther apart falls in separate cluster.
- Hierarchical agglomerative clustering does not provide evenly distributed clusters for user defined number of clusters. HAC used together with visualization of the dendrogram can be used to decide how many clusters exist in the data
- We don't specify the clusters or eps or min value at the start. The clusters are determined by pair of points, just by a link in the proximity graph.
- By looking at the graph, we can identify it is more sensitive to outliers and noise. At times Jaccard coefficient is a lesser value when compared to other algorithms for the same reason.

Advantages:

- Any desired number of clusters can be obtained
- The min approach can handle non-elliptical shapes.

- It outputs dendogram, which can be visualized for the clusters present in data.

Disadvantages:

- Greedy – less accurate and computationally expensive and has high time complexity
- Sensitive to outliers
- Ordering of data has impact on its results
- Our metric for clustering, MIN is susceptible to noise.
- Once a decision is made to combine 2 clusters, it cannot be undone

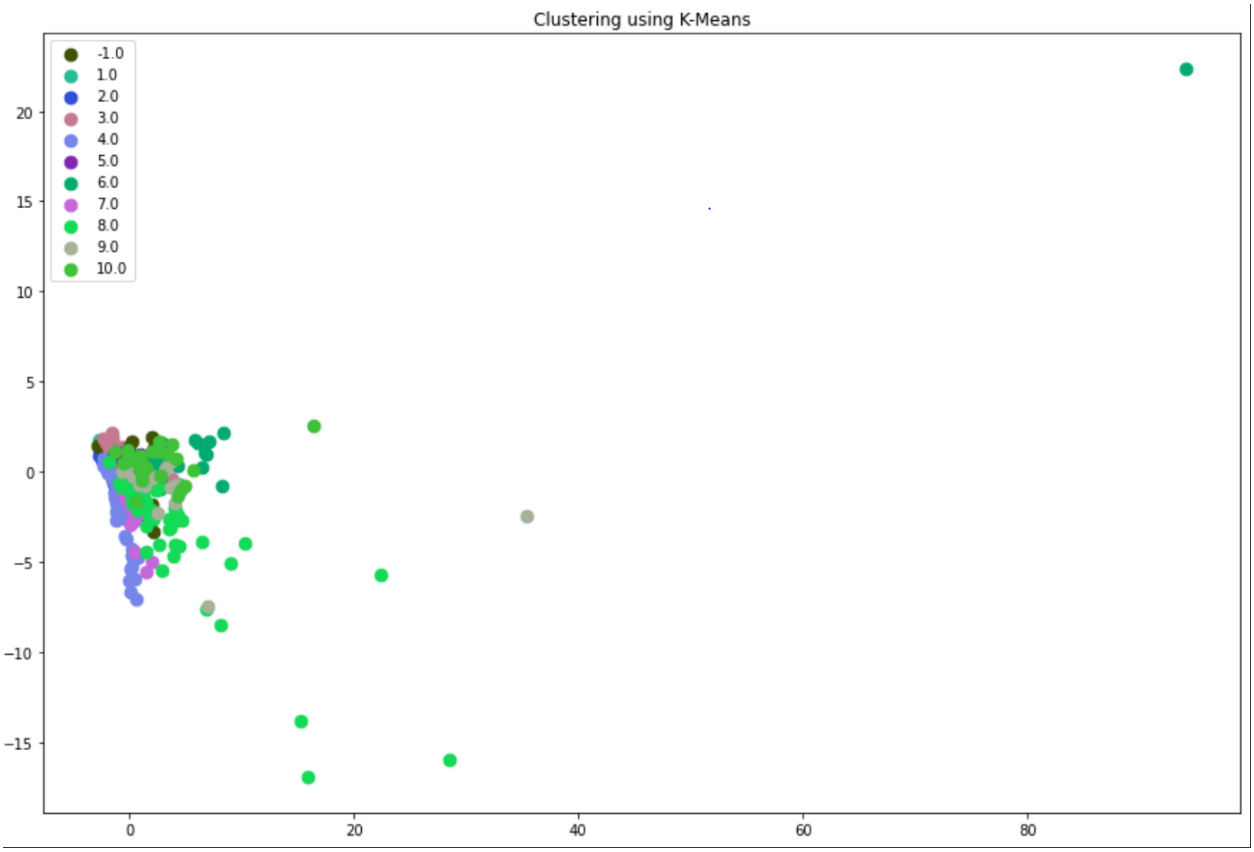
KMeans

KMeans is the center based clustering algorithm that tries to minimize the distance of the cluster with the centroid.

Following are the steps in the algorithm of kmeans :

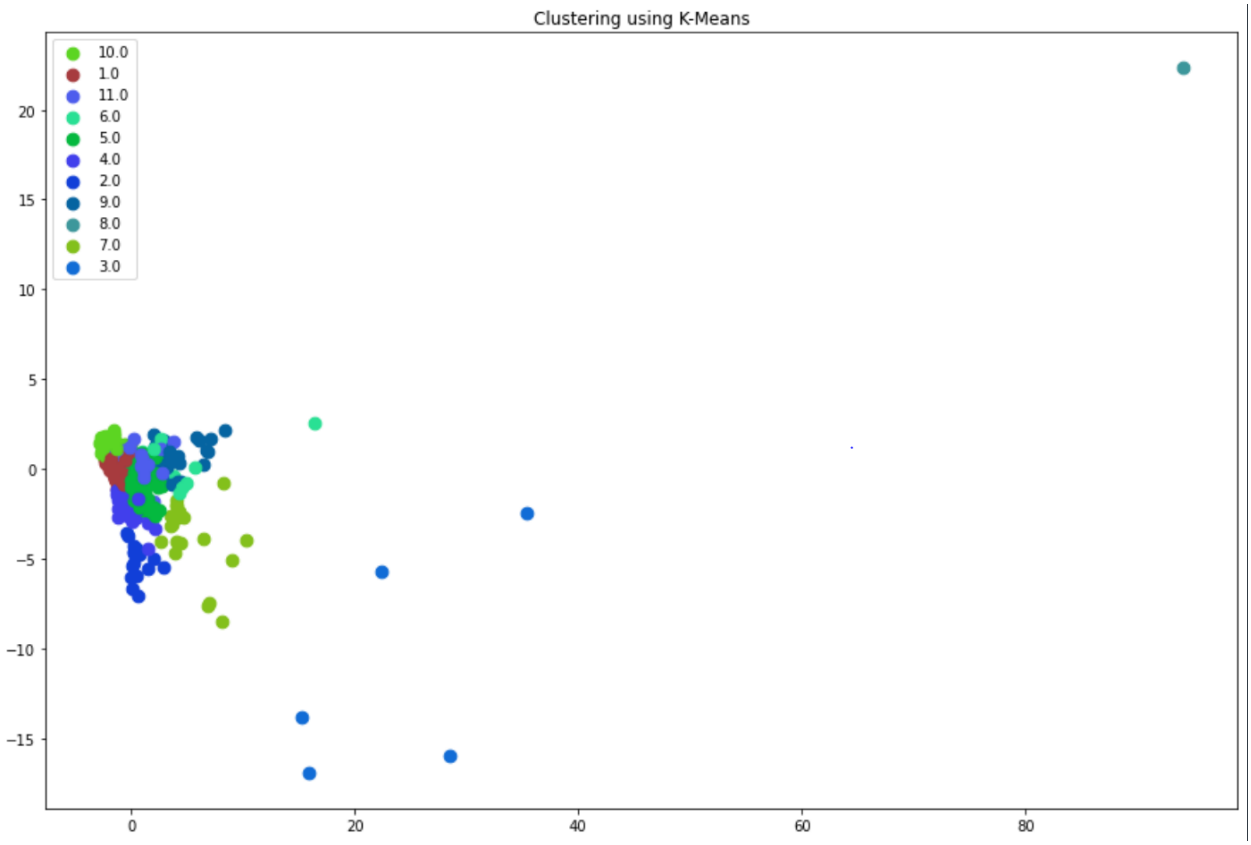
1. pick the number of clusters, k.
2. randomly select the centroid for each cluster
3. After selecting the centroids, we assign each point in the dataset to the closest cluster centroid.
4. After the centers have been assigned we compute the centroids of newly formed clusters
5. We repeat the steps 3 and 4 until we reach the stopping condition. The stopping conditions are:
    - Centroids of the newly formed clusters don't change.
    - Maximum number of iterations have reached.

Ground truth scatter plot for *iyer.txt* file:



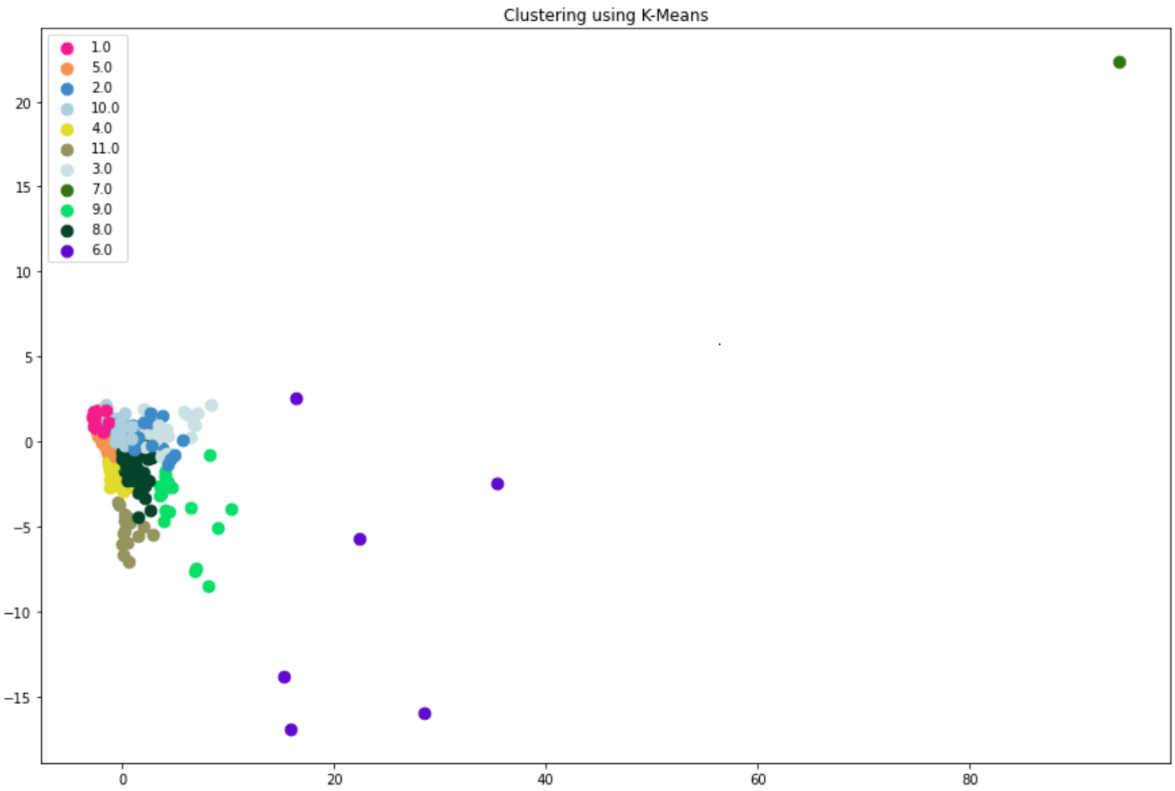
Select first k rows as centroids :

```
enter file path :C:/Users/svelpur/Desktop/601-Project2/iyer.txt

enter the number of clusters :11

enter maximum iterations :100

enter the list of centroids [1,2,3,..]:[]
first k values have been assigned as centroids
oldcentroids==newcentroids
jaccard value is  0.29821409059453424
random index value is  0.7837396974810037
```

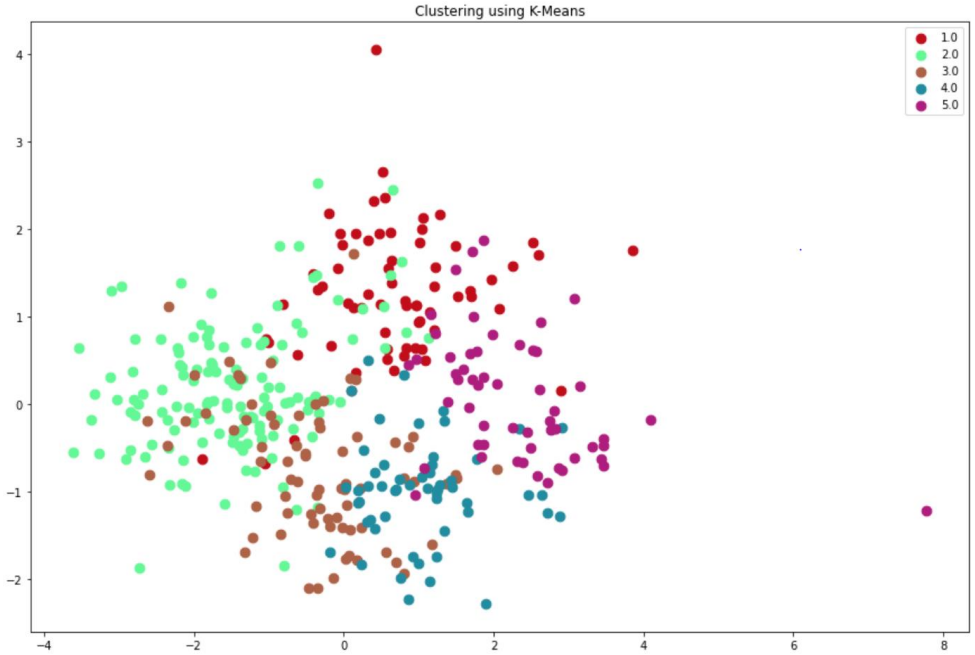


Randomly assigning centroids:

```
enter file path :C:/Users/svelpur/Desktop/601-Project2/iyer.txt

enter the number of clusters :11

enter maximum iterations :100

enter the list of centroids [1,2,3,..]:[1,56,90,45,78,100,8,99,27,66,34]
oldcentroids==newcentroids
jaccard value is  0.31607145140460574
random index value is  0.79988701368182
```

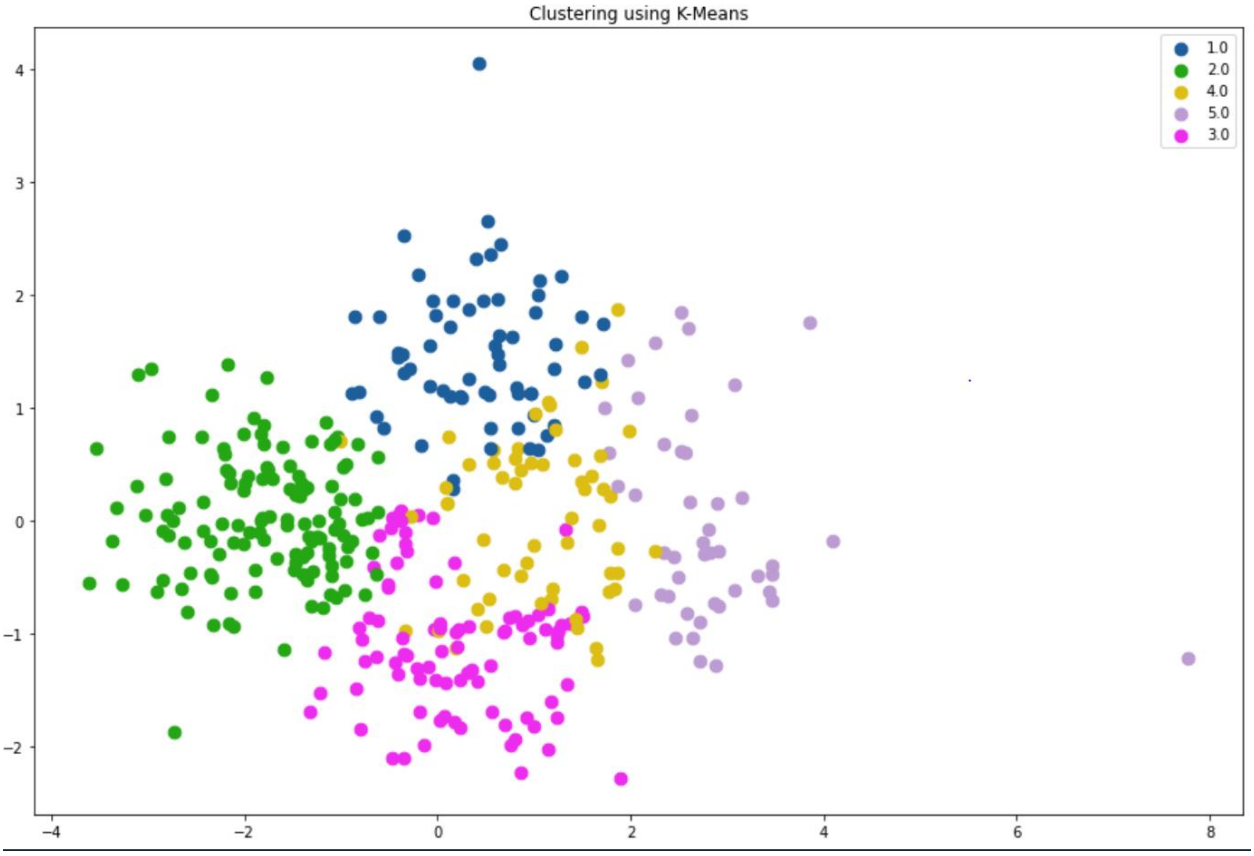


Ground truth plot for cho.txt file:



Selecting the first five values as the initial centroids :

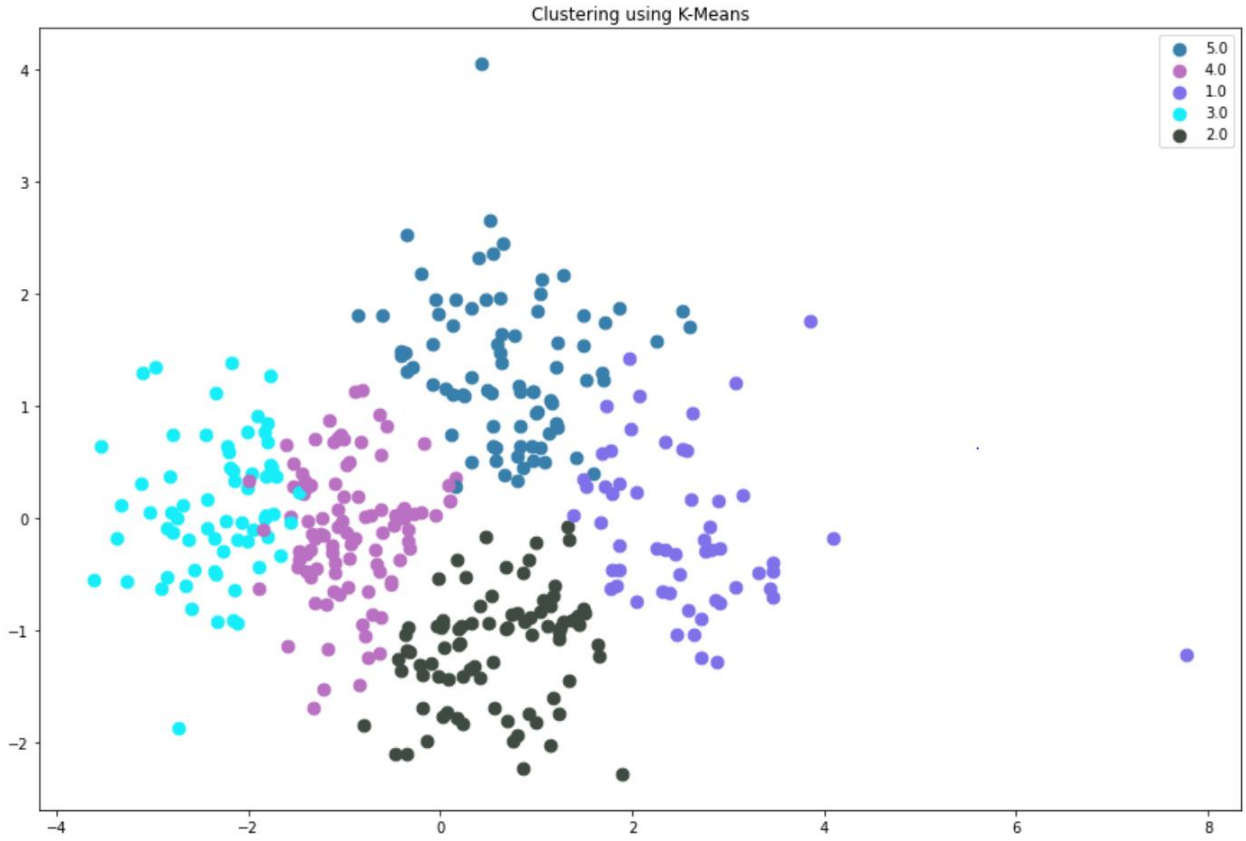
```
enter file path :C:/Users/svelpur/Desktop/601-Project2/cho.txt

enter the number of clusters :5

enter maximum iterations :100

enter the list of centroids [1,2,3,..]:[]
first k values have been assigned as centroids
oldcentroids==newcentroids
jaccard value is  0.3778081971173202
random index value is  0.7925581894815968
```



Selecting centroids randomly:

```
enter file path :C:/Users/svelpur/Desktop/601-Project2/cho.txt

enter the number of clusters :5

enter maximum iterations :100

enter the list of centroids [1,2,3,..]:[350,210,199,50,5]
oldcentroids==newcentroids
jaccard value is  0.33673720472440943
random index value is  0.7829069236758034
```

The value of Jaccard varies between 0.33 to 0.37 when the centroids are chosen continuously and randomly respectively. On the other hand, random index varies between 0.78 and 0.79 respectively for the data in *cho.txt*.



By randomly selecting the centroids, we can see that the Jaccard value varies between 0.29 and 0.31. Similarly, the rand index varies between 0.799 and 0.78.

Observations:

- Higher Jaccard similarity value is obtained when random initial centroids are chosen.
- Based on the initialization of centroids and value of k the clusters converge in thev right way and sometimes they don't.

Advantages:

- Simple to implement
- Scales large datasets
- Guarantees convergence
- Easily adapts to new examples

Disadvantages:

- We have to manually decide the value of k.
- Sometimes, ordering of data has impact on the clusters.
- Depends on the initial values of clusters and could result in local minima.
- Outliers might get their own clusters instead of being ignored.
- Cannot handle non-spherical shaped clusters.

Spectral Clustering:

Spectral clustering is a technique with roots in graph theory, where the approach is used to identify communities of nodes in a graph based on the edges connecting them. The method is flexible and allows us to cluster non-graph data as well.

Below are the steps for computing the Spectral Clustering:

1. We first construct the similarity matrix from the given data set. Similarity is the metric that defines how close two points in the space are. We use Gaussian Kernel to determine our similarity.
2. We construct the degree matrix from the similarity matrix. Degree matrix is the diagonal matrix consisting of the sum of the weights into each vertex.
3. Laplacian matrix is computed in this step. Let us assume degree matrix is D and similarity matrix is W. Then Laplacian matrix L=D-W.
4. We compute the eigen values and their corresponding eigen vectors of Laplacian Matrix.
5. Let the number of clusters be K. We take the smallest K eigen values and their corresponding eigen vectors. Reduce the N*M space vector to N*K space vector (N= number of points, M = number of features).

Following are the results obtained for *cho.txt* file:

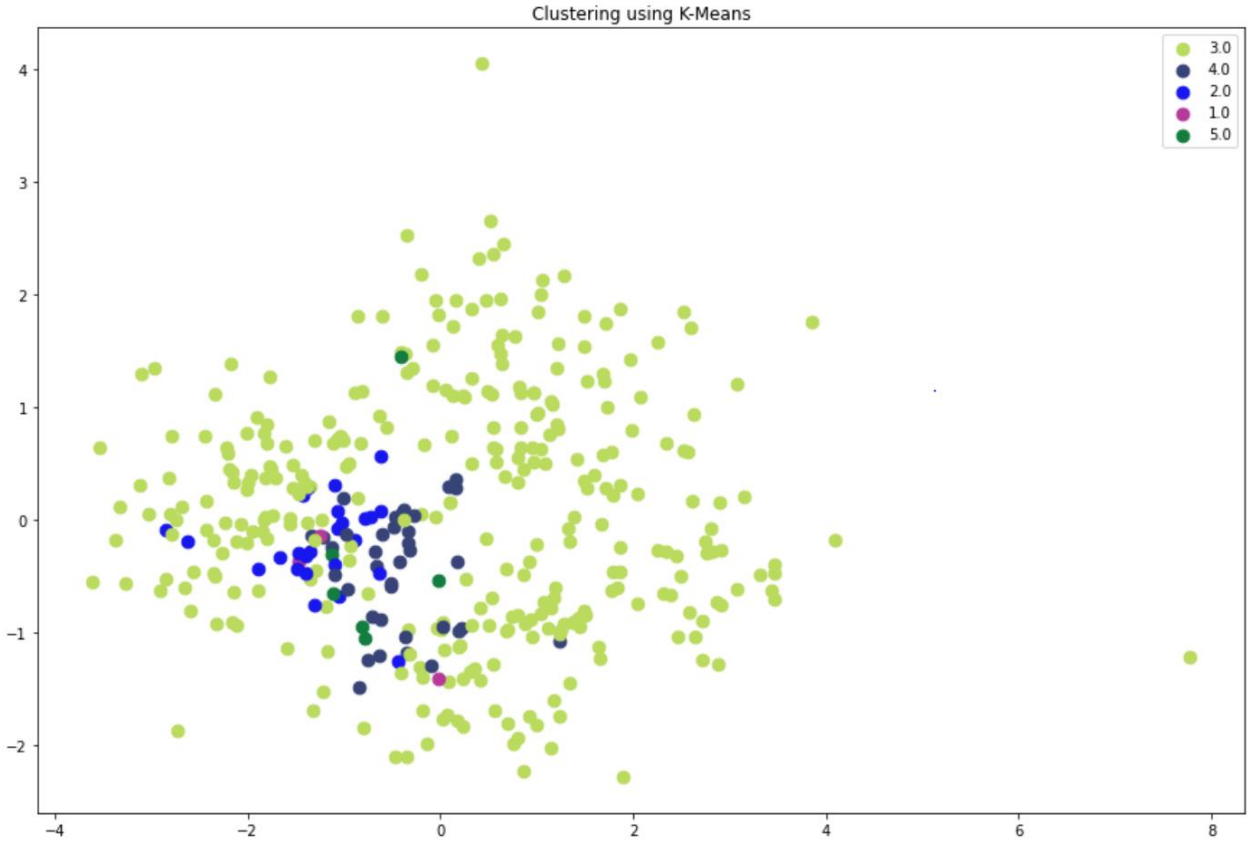
Sigma = 0.8

K=5

Centroids = Random initial centroids

The values of Jaccard and random index are as below:

```
jaccard value is  0.19736400416953764
random index value is  0.3901849714086284
```



Sigma	Jaccard	Random
0.5 k=5	0.2303186667596402	0.25917474294611936
2 k=5	0.23387386823875275	0.34235818411232516
2 K=10	0.17434602200974203	0.6928373916078284
.5 k=10	0.21150487017793174	0.6425004698112701

With the increase in k for same sigma value the random value increases. With the increase in sigma value the there is a slight change in Jaccard and Random Index.

5.4 Result Evaluation:

1. Unlike Kmeans, Spectral Clustering does not assume any prior shape of input data
2. It works good for non-convex shapes.

Advantages:

- Does not make strong assumptions on the statistics of the clusters hence helps create more accurate clusters.
- Easy to implement and gives good clustering results. It can correctly cluster observations that actually belong to the same cluster but are farther off than observations in other clusters due to dimension reduction.

- Reasonably fast for sparse data sets of several thousand elements.

Disadvantages:

- Use of K-Means clustering in the final step implies that the clusters are not always the same. They may vary depending on the choice of initial centroids.

- Computationally expensive for large datasets — This is because eigenvalues and eigenvectors need to be computed and then we have to do clustering on these vectors. For large, dense datasets, this may increase time complexity quite a bit.

Gaussian Mixture models

- It is a probabilistic grounded way of doing soft clustering.
- Each cluster is a generative model (Gaussian in this case)
- The parameters of the model are mean and covariance which are unknown.
- We estimate Gaussian distribution parameters mentioned above for each cluster and weight of a cluster.
- Once the parameters are learnt, we can calculate probability of it belonging to each of the cluster.

Mathematically we can define Gaussian mixture model as mixture of K Gaussian distribution that means it's a weighted average of K Gaussian distribution.

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

where N(x|mu_k,sigma_k) represents cluster in data with mean mu_k and co variance epsilon_k and weight pi_k.

MATH BEHIND THE MODEL:

- Decide on how many sources or clusters you want to fit the data in.
- Initialize the mean, covariance, fraction per class for each cluster. Expectation maximization for mixture models consists of two steps.

- The first step, known as the **expectation** step or **E** step, consists of calculating the expectation of the component assignments C_k*Ck* for each data point x_i \in X*xi*∈*X* given the model parameters phi, mu, and sigma.
- The second step is known as the **maximization** step or **M** step, which consists of maximizing the expectations calculated in the E step with respect to the model parameters. This step consists of updating the values phi, mu, and sigma.
- The entire iterative process repeats until the algorithm converges, giving a maximum likelihood estimate. By alternating between which values are assumed fixed, or known, maximum likelihood estimates of the non-fixed values can be calculated in an efficient manner

Implementation

1. Initialize the mean, covariance, and prior probability for the K Gaussian distribution functions.
2. Try with lesser values for the covariance matrix diagonal, this will avoid singular matrix errors.
3. Now in each iteration, we will perform two operations
    a. E Step
        i. Here we calculate the posterior probability for each point belongs to each of the K clusters.
    b. M Step
        i. In this step, we would update the parameters (mu, sigma, and phi) based on the posterior probability.
        ii. If the parameter updating is less than the convergence threshold, we would stop the iteration.
        iii. We can also use the log-likelihood function for checking the convergence
4. At the end of all iterations, for each point, we would choose the cluster which gives the maximum posterior probability.
5. Once the final set of clusters are obtained, we will plot those clusters by calling the PCA() method provided by the sklearn package.
6. pca_plot_matrix = PCA(n_components=2).fit_transform(points)

Visualization

1. Cho.txt dataset

    python gmm.py –filepath cho.txt

    Enter the number of clusters: 5

    Enter the maximum number of iterations: 15

    Enter the convergence threshold: 0.000000001

    Enter the smoothing value: 0.000000001

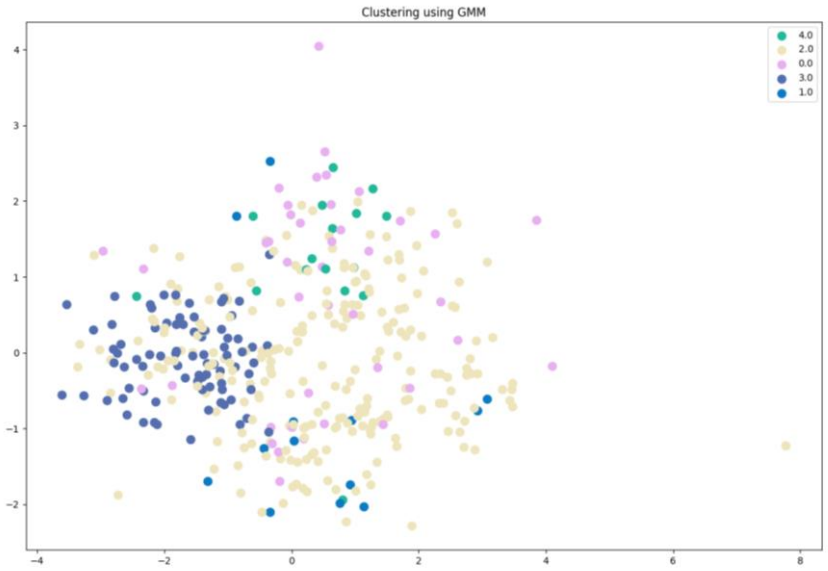
Enter mu value: []

Enter sigma value: []

Enter pi value: []

Jaccard index value: **0.23022439646921194**

Rand index value: **0.6142017235362023**



Clustering using GMM on cho dataset

2. Iyer.txt dataset

python gmm.py –filepath iyer.txt

Enter the number of clusters: 10

Enter the maximum number of iterations: 15

Enter the convergence threshold: 0.0001

Enter the smoothing value: 0.0001

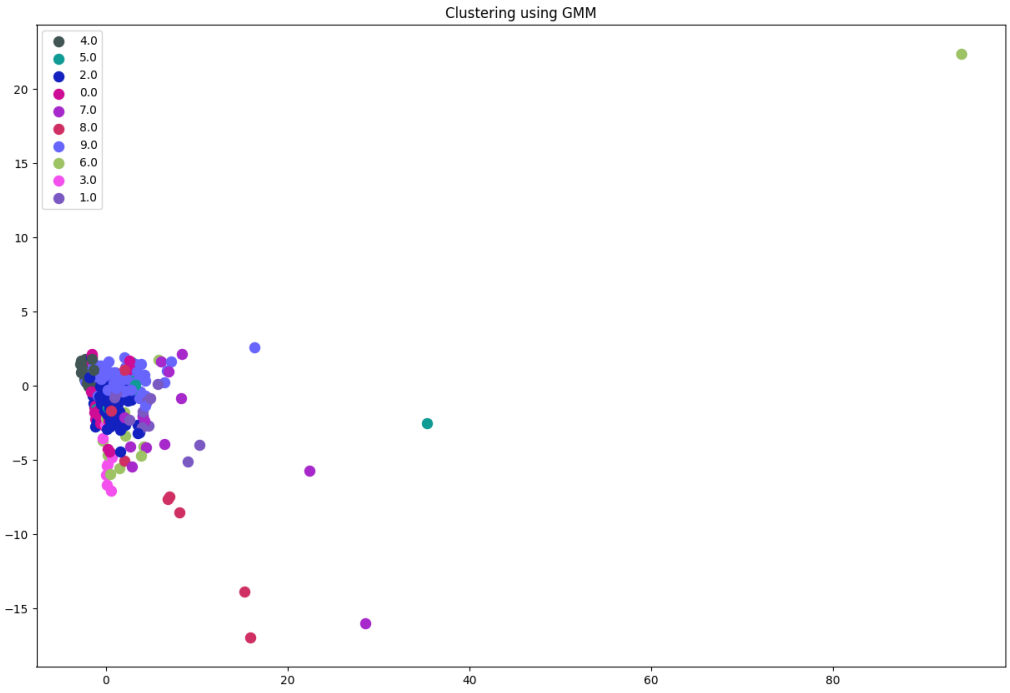
Enter mu value: []

Enter sigma value: []

Enter pi value: []

Jaccard index value: **0.34167594973608323**

Rand index value: **0.7634246078214966**



Clustering using GMM on iyer dataset

Evaluation: The results are varying based on initialization as singular matrix errors are observed during certain initializations. In GMM each cluster is associated with a smooth Gaussian model where multiple random initializations are used.

ADVANTAGES:

- It can handle more complex and ellipsoid shape cluster compared to other models.
- It can handle clusters with varying sizes, variance etc
- It is the fastest algorithm for learning mixture models
- It is based on likelihood of a point falling on a cluster and this approach is more flexible.

DISADVANTAGES:

- It has strong assumptions about the data
- Can cause Overfitting.
- Choose appropriate distributions

References:

- https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages
- https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/
- https://developers.google.com/machine-learning/crash-course/classification/true-false-positivenegative#:~:text=A%20true%20positive%20is%20an,incorrectly%20predicts%20the%20positive%20class.
- https://en.wikipedia.org/wiki/Rand_index#:~:text=The%20Rand%20index%20or%20Rand,similarity%20between%20two%20data%20clusterings.&text=From%20a%20mathematical%20standpoint%2C%20Rand,class%20labels%20are%20not%20used.
- https://towardsdatascience.com/spectral-clustering-aba2640c0d5b