CSE 601

Project 2 Report

Submitted by

Paritosh Kumar Velalam (50295537/pvelalam)

Sampreeth Reddy B (50298591/sboddire)

1. K-Means Algorithm:

1.1 Introduction:

K-Means is one of the most popular "clustering" algorithms. K-means stores K centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid.

K-Means finds the best centroids by alternating between

(1) Assigning data points to clusters based on the current centroids

(2) Choosing centroids (points which are the center of a cluster) based on the current assignment of data points to clusters.

1.2 Algorithm:

In the clustering problem, we are given a training set {x1, x2, ... ,xm}, and want to group the data into a few cohesive "clusters." Here, we are given feature vectors for each data point x1 in R^n as usual; but no labels yi (making this an unsupervised learning problem). Our goal is to predict K centroids and a label ci for each data point. The k-means clustering algorithm is as follows:

1. Choose the number of clusters *k*
2. Select k random points from the data as centroids
3. Assign all the points to the closest cluster centroid

For every i, set

$$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

4. Recompute the centroids of newly formed clusters

For each j, set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\}x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$$

5. Repeat steps 3 and 4

1.3 Implementation:

1. Get the number of clusters, initial centroids and maximum number of iterations from input. If the initial centroids is specified by empty list, then random data points are chosen as initial centroids.

2. Assign each data point to one of the K cluster centroids by selecting the cluster centroid to which the Euclidean distance is smaller. Euclidean distance of each data point to cluster centroid is calculated using np.linalg.norm (datapoint - centroid).
3. Calculate the new Cluster centroids by calculating the geometric mean of all the points that belong to a cluster.
4. Repeat steps 2 and 3 until there is no change in old centroids and new centroids or the maximum number of iterations is reached.

5. Using the original data and cluster indices computed using K-Means are used to compute the Jaccard co-efficient and rand index.

- Jaccard co-efficient is computed as:

$$Jaccard\ coefficient = \frac{|M_{11}|}{|M_{11}| + |M_{10}| + |M_{01}|}$$

- Rand Index is computed as:

$$Rand = \frac{|Agree|}{|Agree| + |Disagree|} = \frac{|M_{11}| + |M_{00}|}{|M_{11}| + |M_{00}| + |M_{10}| + |M_{01}|}$$

Where, M_{11} : (agree, same cluster) M_{00} : (agree, different clusters)

M_{10} : (disagree, different clusters) M_{01}: (disagree, different clusters)

6. The original data is reduced to two dimensions by performing PCA and scatter plot is drawn with the cluster labels obtained by performing K-Means.

1.3 Visualization:

1.3.1 Ground Truth Scatter Plot for "cho.txt":

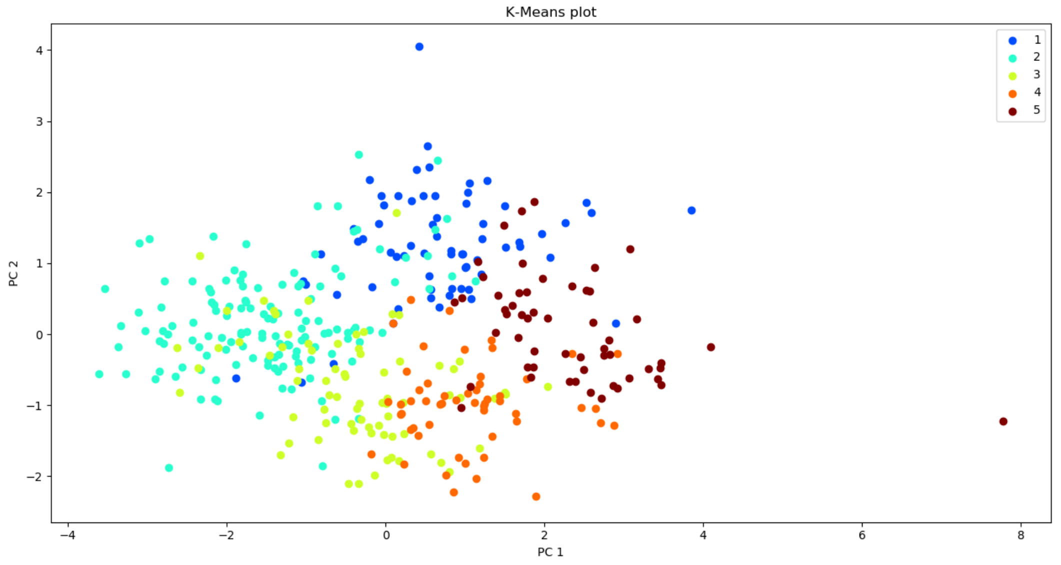


Figure 1.1 Ground Truth Scatter plot for "cho.txt"

1.3.2

Parameters:

File: “cho.txt”, Number of Clusters: 5, Initial Centroids: [ ], Maximum Iterations: 100

The obtained Jaccard Coefficient and Rand Index values are:
Jaccard Coefficient: 0.34425066755674233
Rand Index: 0.7890278933662649

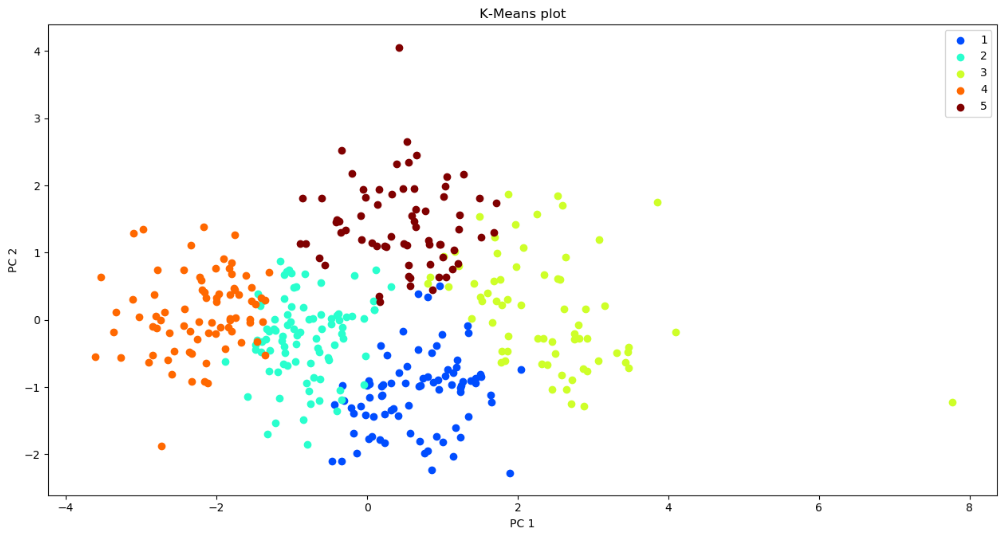


Figure 1.2. Scatter plot for “cho.txt after K-Means”

1.3.3

Parameters:

File: “cho.txt”, Number of Clusters: 5, Initial Centroids: [5, 25, 75, 125, 150], Maximum Iterations: 100

The obtained Jaccard Coefficient and Rand Index values are:
Jaccard Coefficient: 0.39166288706551766
Rand Index: 0.8019544148836211

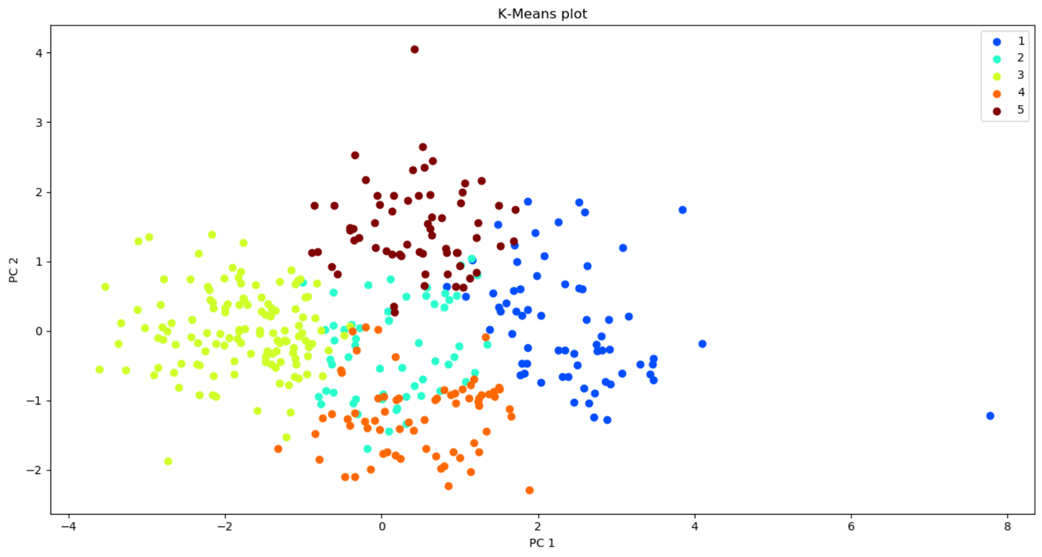


Figure 1.3. Scatter plot for “cho.txt after K-Means”

By selecting random values as initial centroids, we observe that the Jaccard co-efficient for 5 cluster varies in the range 0.33 to 0.391. Similarly, the rand index varies in the range 0.75 to 0.802.

1.3.4
Ground Truth Scatter Plot for “iyer.txt”

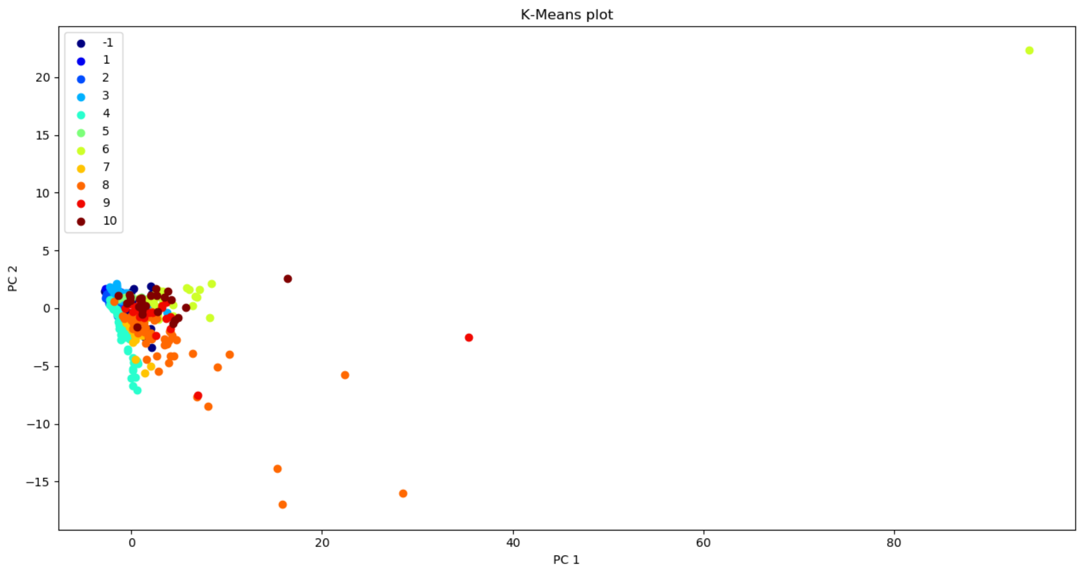


Figure 1.4. Ground Truth scatter plot for “iyer.txt”

1.3.5
Parameters:
File: “iyer.txt”, Number of Clusters: 11, Initial Centroids: [ ], Maximum Iterations: 100
The obtained Jaccard Coefficient and Rand Index values are:
Jaccard Coefficient: 0.3032080633128045
Rand Index: 0.8217958838856051

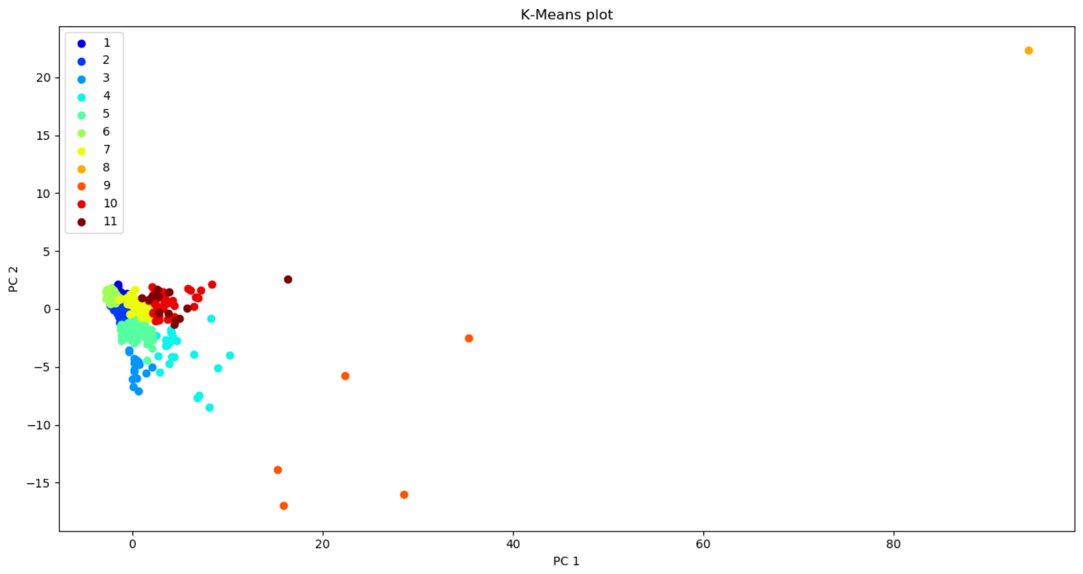


Figure 1.5. Scatter plot for “iyer.txt after K-Means”

1.3.6
Parameters:
File: “iyer.txt”, Number of Clusters: 11, Initial Centroids: [1,2,3,4,5,6,7,8,9,10,11],
Maximum Iterations: 100
The obtained Jaccard Coefficient and Rand Index values are:
Jaccard Coefficient: 0.40648677204543243
Rand Index: 0.8402777517967444

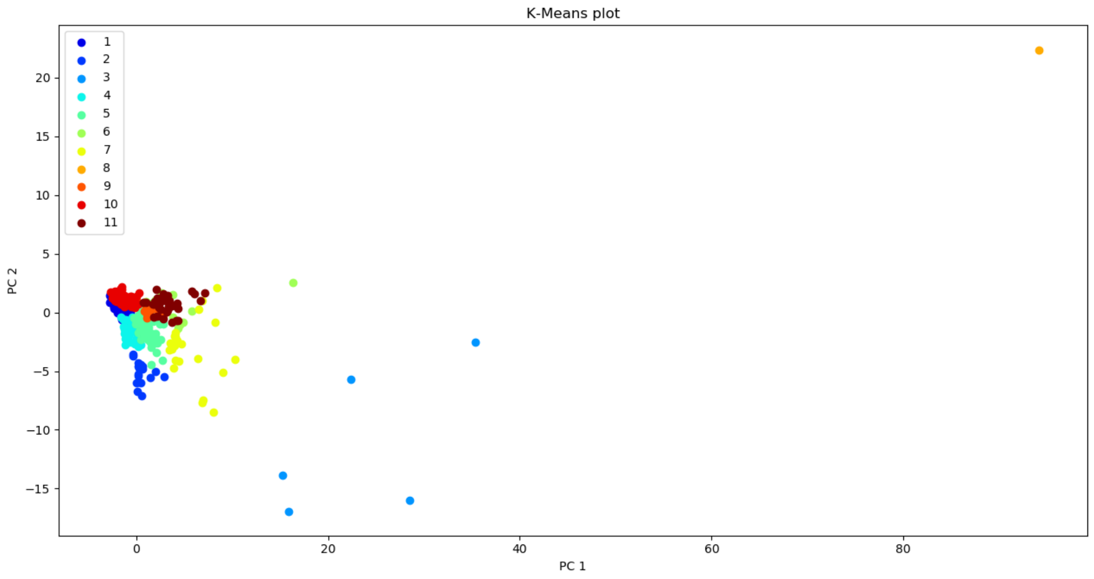


Figure 1.6. Scatter plot for “iyer.txt after K-Means”

By selecting random values as initial centroids, we observe that the jaccard co-efficient for 11 clusters varies in the range 0.30 to 0.4. Similarly, the rand index varies in the range 0.77 to 0.84.

1.4 Result Evaluation:

1. Higher jaccard similarity scores are obtained when initial centroids are chosen at random.
2. Sometimes the initial centroids will readjust themselves in ‘right’ way, and sometimes they don’t.

Advantages:

- For smaller values of k, k-means provides computationally faster results.
- K means provides compact clustering results.
- K-means clustering is usually more efficient run-time wise: O(tkn), where n is # objects, k is # clusters, and t is # iterations. Normally, k, t << n
- K-means algorithm is good for large dataset clustering

Disadvantages:

- Without initial ground truth values, it is difficult to predict the value of K.
- Selecting different initial centroids, provides different similarity results.
- Can lead to empty clusters. But most of the data points are usually clustered.
- Some pre and post processing may be required on the initial and final data for obtaining perfect results.
- Doesn’t perform well when the clusters differ in sizes, densities and shapes

2. Hierarchical Agglomerative Clustering Algorithm:

2.1 Introduction:

In hierarchical agglomerative clustering consists of a bottom up approach, we build a hierarchy of clusters represented using a dendogram. At the beginning each point is considered as a single cluster. Then each cluster is merged with their nearest neighbour (here based on minimum Euclidean distance) recursively till all points belong to a single cluster. The merging policy is greedy, each time, we only merge a pair of clusters that has the smallest inter-cluster distance. Other similarity functions can also be used in this algorithm to merge two clusters.

2.2 Algorithm:

Given a set of N items to be clustered, and an N*N distance (or similarity) matrix

1. Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item. Let the distances (similarities) between the clusters the same as the distances (similarities) between the items they contain.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one cluster less.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N.

2.3 Implementation:

1. Get the number of clusters from input.
2. Union Find data structure has been used to assign the cluster of the data points after each iteration.
3. The scipy.spatial package was used to compute the Euclidean distance for the points stored in the matrix.
4. At the start each point is stored as single cluster.
5. Then we start merging the clusters till only the given number of clusters remain.
6. We implemented Single-linkage: the similarity of the closest pair
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 union method.
8. The distance matrix is recomputed for remaining points using minimum distance between two of the merged points. For example if points 2 and 5 are merged in this step. Distance of all other points will be calculated w.r.t to cluster (1,2) using, d((1,2), P) = min(d(1,P),d(2,P). The distance matrix is recomputed every time two points or clusters are merged.

9. Finally labels are assigned based on the cluster to which the data point belongs.
10. Jaccard Coefficient and rand index are computed.
11. The data is visualized by reducing the data to two dimensions using PCA.

2.4 Visualization:

For cho.txt and 5 clusters

The obtained Jaccard Coefficient and Rand Index values are:

Jaccard Coefficient: 0.22839497757358454
Rand Index: 0.24027490670890495

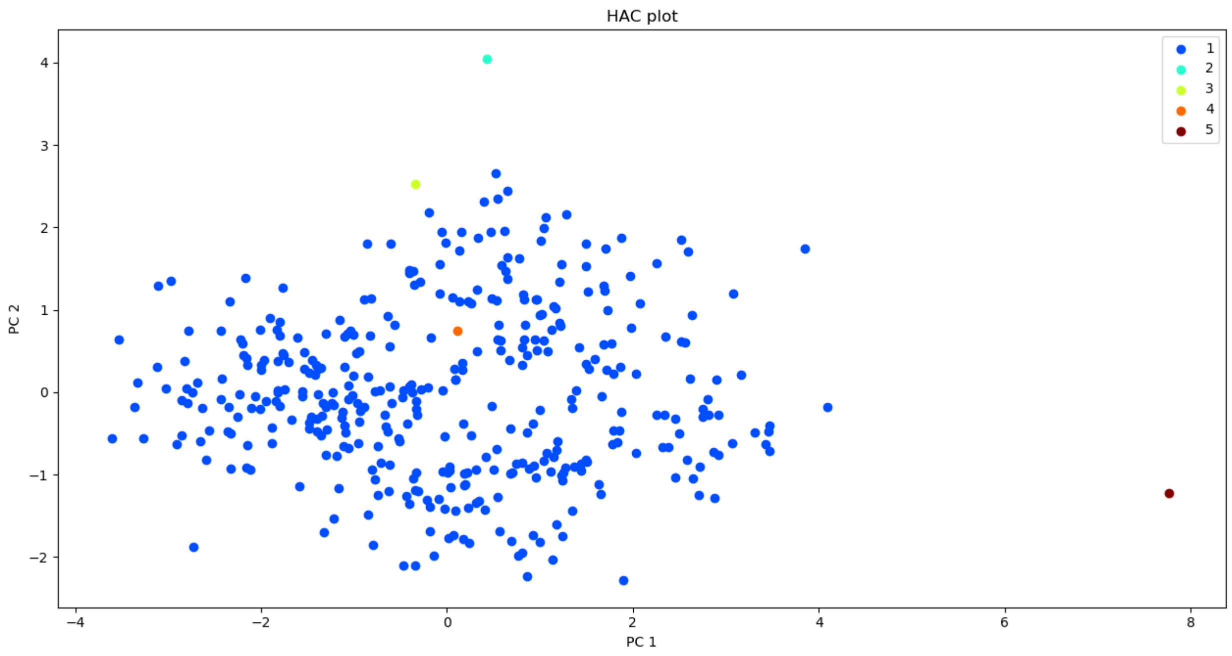


Figure 2.1 Scatter Plot for “cho.txt” after Hierarchical Agglomerative Clustering

Different combinations of number of clusters provide a range for external index. The Jaccard coefficient varies from 0.22 to 0.24.

For iyer.txt and 11 clusters
The obtained Jaccard Coefficient and Rand Index values are:
Jaccard Coefficient: 0.1584602101134175
Rand Index: 0.1941381800223728

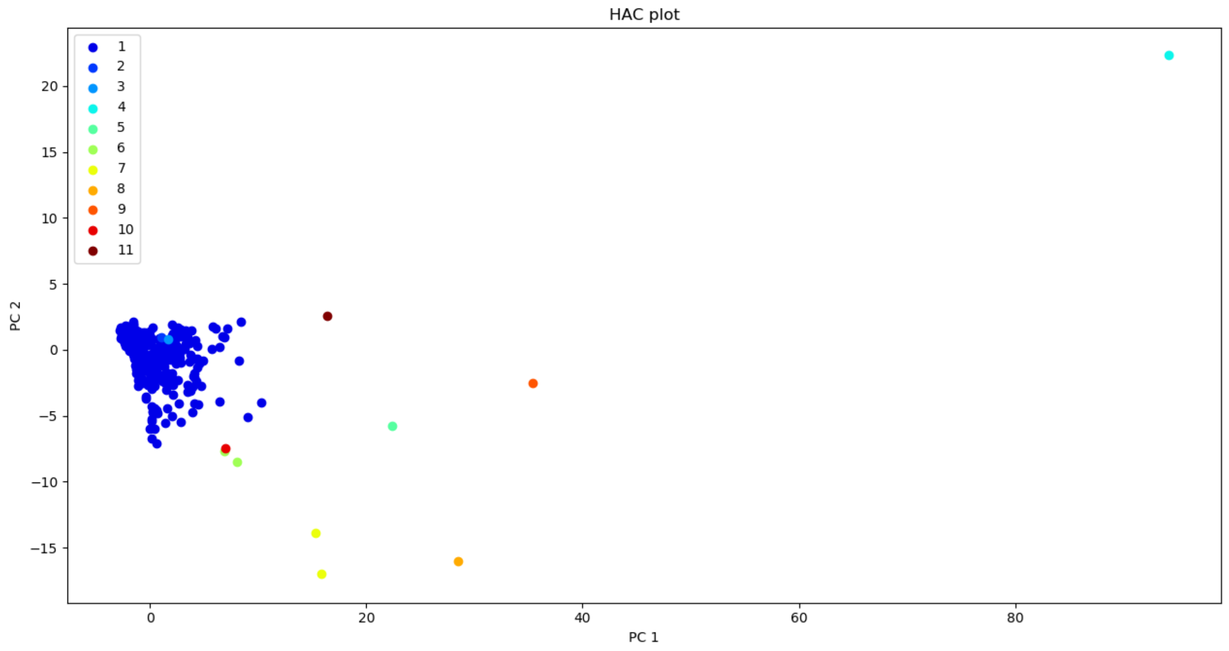


Figure 2.2 Scatter plot for “iyer.txt” after Hierarchical Agglomerative Clustering

Different combinations of number of clusters provide a range for external index. The Jaccard co efficient varies from 0.15 to 0.18

2.5 Result Evaluation:

- We were required to implement hierarchical clustering using single linkage. In single-linkage clustering, the similarity of two clusters is the similarity of their most similar members i.e. while recalculating distance matrix the minimum distance of a point from any point belonging to that cluster is stored.
- This single-link merge criterion is local. We pay attention solely to the area where the two clusters come closest to each other. Distant points of the cluster and the clusters' overall structure are not taken into account.
- Hierarchical agglomerative clustering does not identify outliers in the data as it merges all the points. Also it 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.

Advantages:

- No prior information about the number of clusters required
- It can produce an ordering of the objects, which is informative for data display (visualized mostly as dendograms)

- It is shown to capture concentric clusters
- It provides hierarchical relations between clusters
- We can create smaller clusters depending on where we cut the dendrograms which is helpful for discovery
- Meaningful taxonomies might be produced
- Allows anisotropic and non-convex shapes

Disadvantages:

- Greedy – less accurate and computationally expensive
- A single-link clustering can produce a chaining effect i.e. it produces straggling clusters. As the merge criterion is local, a chain of points can be extended for long distances without regard to the overall shape of the emerging cluster
- Sensitivity to noise and outliers
- hierarchical clustering is high in time complexity
- No objective function is directly minimized
- Algorithm can never undo what was done previously
- Sensitive to outliers
- Difficulty handling different sized clusters and convex shapes

3. Gaussian Mixture Model:

3.1 Description:

In real life, many datasets can be modeled by Gaussian Distribution (Univariate or Multivariate). So it is quite natural and intuitive to assume that the clusters come from different Gaussian Distributions. Or in other words, it is tried to model the dataset as a mixture of several Gaussian Distributions. This is the core idea of this model.
In one dimension the probability density function of a Gaussian Distribution is given by

$$G(X|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

where μ and σ^2 are respectively mean and variance of the distribution.
For Multivariate Gaussian Distribution, the probability density function is given by

$$G(X|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(X-\mu)^T\boldsymbol{\Sigma}^{-1}(X-\mu)\right)$$

Here μ is a d dimensional vector denoting the mean of the distribution and ∑ is the d X d covariance matrix.
Suppose there are K clusters (For the sake of simplicity here it is assumed that the number of clusters is known and it is K). So μ and ∑ is also estimated for each k. Had it been only one distribution, they would have been estimated by **maximum-likelihood method**. But since there are K such clusters and the probability density is defined as a linear function of densities of all these K distributions, i.e.

$$p(X) = \sum_{k=1}^{K} \pi_k G(X|\mu_k, \Sigma_k)$$

where π_k is the mixing coefficient for k-th distribution.

We want to maximize Loglikelihood

$$\ln p(x|\pi,\mu,\Sigma) = \sum_{i=1}^{n} \ln\{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k,\Sigma_k)\}$$

Each data point is generated by one of *K* clusters, a latent variable $z_i = (z_{i1}, \ldots, z_{iK})$ is associated with each x_i

$$\sum_{k=1}^{K} z_{ik} = 1 \text{ and } p(z_{ik}=1) = \pi_k$$

We Regard the values of latent variables as missing

Expectation-Maximization Algorithm

The Expectation-Maximization (EM) algorithm is an iterative way to find maximum-likelihood estimates for model parameters when the data is incomplete or has some missing data points or has some hidden variables. EM chooses some random values for the missing data points and estimates a new set of data. These new values are then recursively used to estimate a better first data, by filling up missing points, until the values get fixed.

These are the two basic steps of the EM algorithm, namely **E Step or Expectation Step or Estimation Step** and **M Step or Maximization Step**.

- Initialize μk, $\sum_k$ and πk by some random values, or by K means clustering results or by hierarchical clustering results.

Estimation step:

- For given parameter values we can compute the expected values of the latent variables

$$\begin{aligned} r_{ik} \equiv E(z_{ik}) &= p(z_{ik} = 1 | x_i, \pi, \mu, \Sigma) \\ &= \frac{p(z_{ik} = 1) p(x_i | z_{ik} = 1, \pi, \mu, \Sigma)}{\sum_{k=1}^{K} p(z_{ik} = 1) p(x_i | z_{ik} = 1, \pi, \mu, \Sigma)} \\ &= \frac{\pi_k \mathcal{N}(x_i | u_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i | u_k, \Sigma_k)} \end{aligned}$$

Note that $r_{ik} \in [0, 1]$ instead of $\{0, 1\}$ but we still have $\sum_{k=1}^{K} r_{ik} = 1 \text{ for all } i$

Maximization step:

Maximize the expected complete log likelihood

$$E[\ln p(x, z | \pi, \mu, \Sigma)] = \sum_{i=1}^{n} \sum_{k=1}^{K} r_{ik} \{\ln \pi_k + \ln \mathcal{N}(x_i | \mu_k, \Sigma_k)\}$$

Parameter update:

$$\pi_k = \frac{\sum_i r_{ik}}{n} \qquad \mu_k = \frac{\sum_i r_{ik} x_i}{\sum_i r_{ik}}$$

$$\Sigma_k = \frac{\sum_i r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i r_{ik}}$$

3.2 Algorithm:

1. Initialize the mean μ_k, the covariance matrix $\sum_k$ and the mixing coefficients π_k by some random values. (or other values)
2. Compute the r_{ik} values for all k.
3. Again Estimate all the parameters using the current r_{ik} values.
4. Compute log-likelihood function. $\sum_{i=1}^{n} \log\{ \sum_{k=1}^{k} \pi_k N(x/\mu_k, \sum_k)\}$.
5. Put some convergence criterion
6. If the log-likelihood value converges to some value (or if all the parameters converge to some values) then stop, else return to Step 2.

3.3. Implementation:

- Get the number of clusters, initial priors,means and covariance matrices from input or else get these using K-Means.
- Perform the E-Step:
  - The likelihood is computed using multivariate_normal.pdf(data, means[i], covar[i])
  - Then likelihood is multiplied with prior. Then each value in the row is divided by the sum of values of that row to get R matrix where r_{ik} represents i^{th} datapoint and k^{th} cluster.
- Perform the M-Step:
  - Each value in the R matrix is divided by the sum of values along that column to get $N_k = \sum_i r_{ik}$. Then this result is divided with n to get π matrix.
  - Dot product is computed between transpose of R matrix and data and each row values is divided with corresponding row values of transpose of N_k to get μ matrix.
  - Then for each cluster i, data and means matrices are subtracted and transpose is taken which is denoted as diff. Then np.dot((R[:,i]*diff), diff.T)/($\mathrm{Nk}[i]$) is performed to covariance matrix of cluster i.
  - Then loglikelihood is computed

4. Steps 2 and 3 are repeated till convergence threshold or maximum iterations is achieved.

3.4 Visualization:

Parameters:
File Name: “cho.txt”, No of Clusters: 5, Means: [ ], Covariance: [ ], Prior: [ ], No. of iterations: 600, Convergence Threshold: 1e-9, Smoothing value: 1e-9

The obtained Jaccard Coefficient and Rand Index values are:

Jaccard Coefficient: 0.3881046489742142
Rand Index: 0.781806222985852

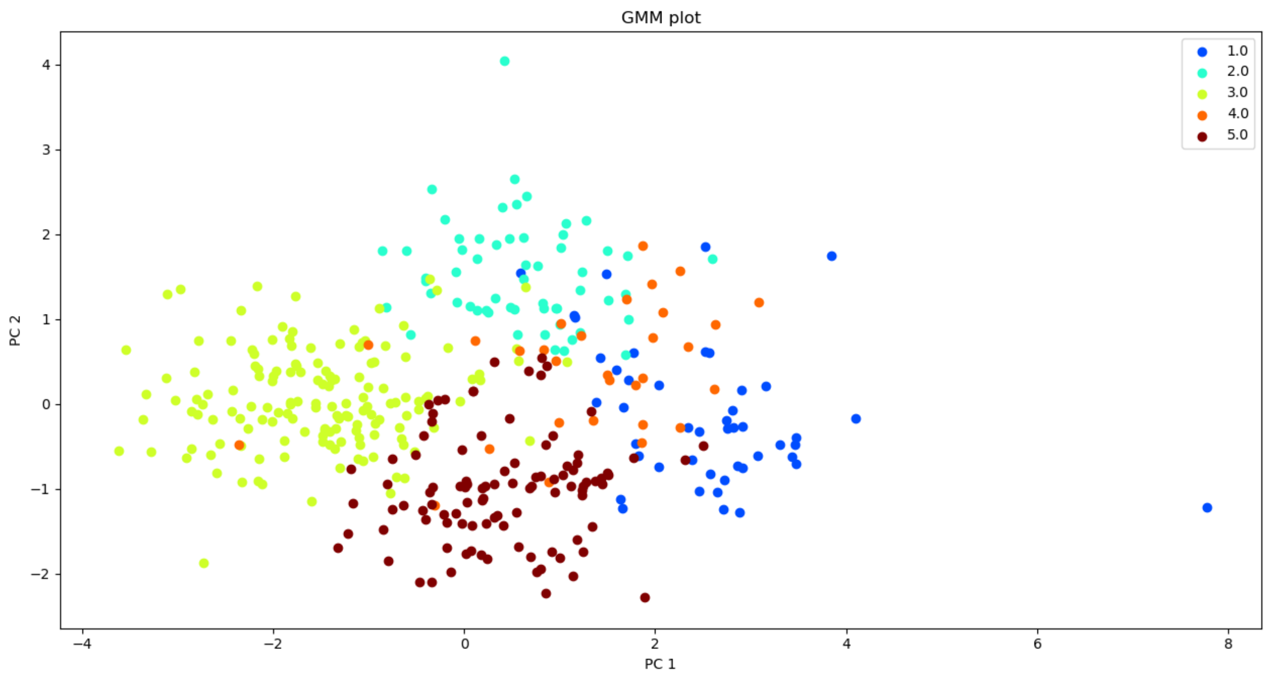


Figure 3.1 Scatter plot for “cho.txt” after Gaussian Mixture Model Clustering

Different combinations of number of clusters provide a range for external index. The Jaccard co efficient varies from 0.3 to 0.388

Parameters:
File Name: “iyer.txt”, No of Clusters: 11, Means: [ ], Covariance: [ ], Prior: [ ], No. of iterations: 600, Convergence Threshold: 1e-9, Smoothing value: 1e-9

The obtained Jaccard Coefficient and Rand Index values are:

Jaccard Coefficient: 0.37225728355016513
Rand Index: 0.8001638675740491

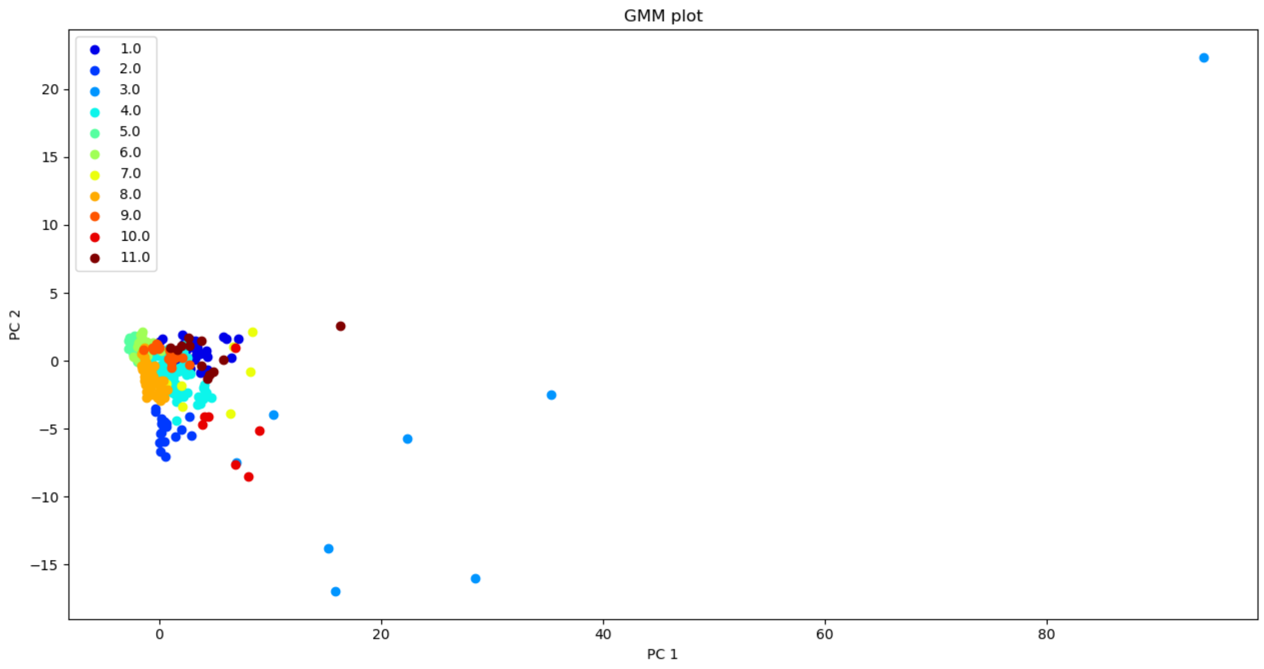


Figure 3.2 Scatter plot for “iyer.txt” after Gaussian Mixture Model Clustering

Different combinations of number of clusters provide a range for external index. The Jaccard co efficient varies from 0.28 to 0.37

3.5 Result Evaluation:

1. The results are highly dependent on initialization as singular matrix errors are observed during certain initializations.

Advantages:

- It gives probabilistic cluster assignments
- Can have probabilistic interpretation

- It can handle clusters with varying sizes, variance etc
- It is the fastest algorithm for learning mixture models
- As this algorithm maximizes only the likelihood, it will not bias the means towards zero, or bias the cluster sizes to have specific structures that might or might not apply.

Disadvantages:
- When one has insufficiently many points per mixture, estimating the covariance matrices becomes difficult, and the algorithm is known to diverge and find solutions with infinite likelihood unless one regularizes the covariances artificially.
- This algorithm will always use all the components it has access to, needing held-out data or information theoretical criteria to decide how many components to use in the absence of external cues.
- In, GMM algorithm user must set the number of mixture models that the algorithm will try and fit to the training dataset. In many instances the user will not know how many mixture models should be used and may have to experiment.
- Can cause Overfitting.
- Choose appropriate distributions.

4. DBSCAN:

4.1 INPUT PARAMETERS:

Eps: specifies how close points should be to each other to be considered a part of a clusters

MinPoints: Defines Minimum number of points to form a dense region

• DBSCAN is a density-based algorithm.

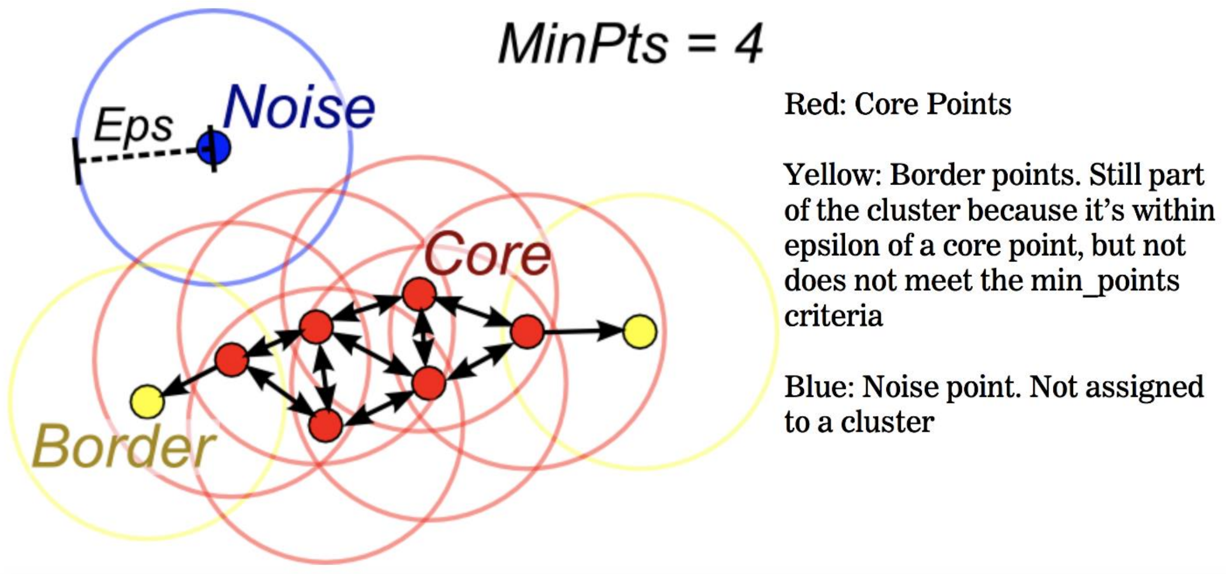
• DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise.

• It locates regions of high density that are separated from other regions of low density, where density = number of points within a specified radius.

• Points are classified into the following:

1. **Core points**: Points having more than minimum number of points in its neighborhood. These points are present at the interior of a cluster. Any core points that are close enough – within a distance of given radius – are present in the same cluster.
2. **Border points:** Points having less than minimum number of points in its neighborhood, but still present in neighborhood 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.
3. **Noise**: Any point that is not a core or border point and also having less than minimum number of points in its neighborhood. Noise are not present in any cluster.

Example demonstrating above points:


MinPts = 4
Eps
Noise
Core
Border
Red: Core Points
Yellow: Border points. Still part of the cluster because it's within epsilon of a core point, but not does not meet the min_points criteria
Blue: Noise point. Not assigned to a 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.

4.2 DBSCAN Clustering Algorithm

1. Arbitrary select a point p
2. Retrieve all points density-reachable from p wrt Eps and MinPts.
3. If p is a core point, a cluster is formed.
4. If p is a border point, no points are density-reachable from p and DBSCAN visits the next point of the database.
5. Continue the process until all of the points have been processed

4.3 Implemented Algorithm:

1. The data is loaded as using input file path and then cast as a numpy array in to data. From this data, the first 2 columns are removed to get only the feature matrix and assign it to data_x.
2. The epsilon (radius) value and the minimum no. of points are taken as user input and passed as arguments to the dbscan function.
3. In the dbscan function, cluster ID is initially set to 0 and a numpy zeros matrix of shape no. of (rows of data_x , 1) is created to assign the new labels.
4. For each unvisited point pt in the matrix, find the neighbors of the point using neighbors function which returns the points within the distance eps of the point using Euclidean distance.
5. If the number of neighbors is less than the minimum number of points, classify the label of that point pt to be -1 and the next point is checked.
6. If the number of neighbors is more than the minimum number of points, classify the label of
that point pt to the corresponding cluster ID, which is incremented every time when a new
cluster is formed.
7. Then for every unvisited point pt in the neighborhood of point pt, it is assigned to the same cluster id as pt’s cluster id. The point’s neighbors are calculated by calling neighbors function and then added on to the existing neighboring points.
8. For every point in the neighborhood of point pt which has been assigned as noise earlier are
classified as border points to the current cluster and thus, labeled as same cluster id.
9. At the end of the loop, after every point pt in the matrix are visited, label gives cluster Id of every point in which points are identified using the index of the cluster Id.

4.4 Estimation of Parameters:

Epsilon:

If the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don't satisfy the number of points to create a dense region. On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster. The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.

MinPoints:

As a general rule, a minimum minPoints can be derived from a number of dimensions (D) in the data set, as minPoints ≥ D + 1. Larger values are usually better for data sets with noise and will form more significant clusters. X

4.5 VISUALIZATION:

File Name: cho.txt

Here, the colors correspond to the Cluster ID assigned by the algorithm.

After experimenting with varied values of ε and MinPts, we have clustered the data by fixing the value of ε 1.03 and the value of Minpts to 3

Resulting clusters are as shown:

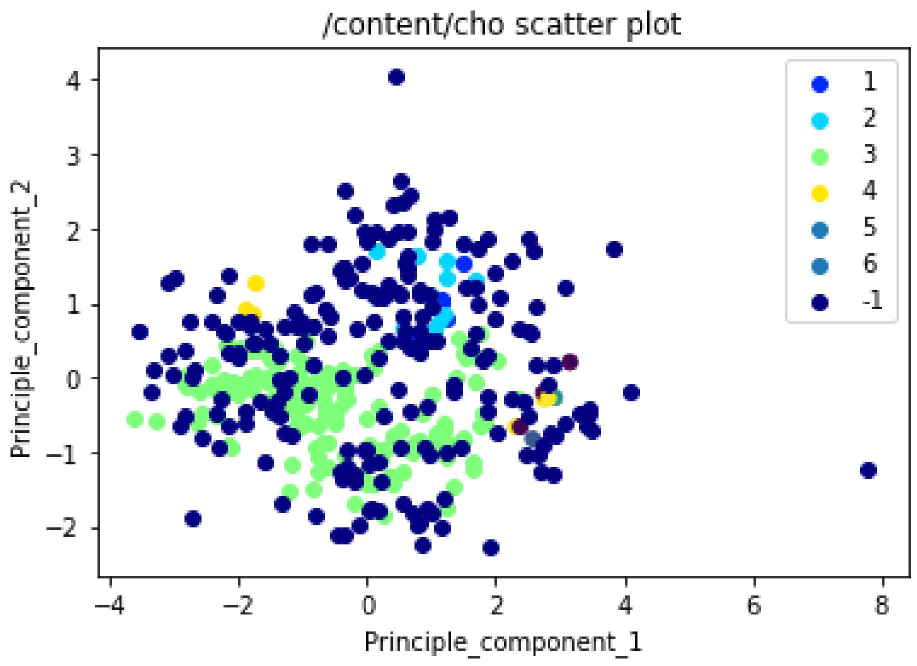


Figure 4.1 Scatter plot for "cho.txt after DBScan clustering"

As we can see, the algorithm has efficiently identified some of the data points as noise by clustering them with a cluster ID as -1 and the remaining data points are represented by their cluster number. The dimensions of the data points have been reduced and we have used PCA for effective visualization.

The value of jaccard co-efficient and rand index is:

Jaccard Coefficient: 0.20164510077444803
randIndex: 0.5544309914360117

Different combinations of ε and MinPts provide a range for external index. The data is clustered into noise or with cluster IDs accordingly depending upon these input parameters. The Jaccard coefficient varies from 0.20 to 0.23 Similarly, the rand index is noted to vary from 0.23 to 0.55

File Name: iyer.txt:

Here, the colors correspond to the Cluster ID assigned by the algorithm. After experimenting with varied values of ε and MinPts, we have clustered the data by fixing the value of ε 0.8 and the value of Minpts to 3.

Resulting clusters are as shown:

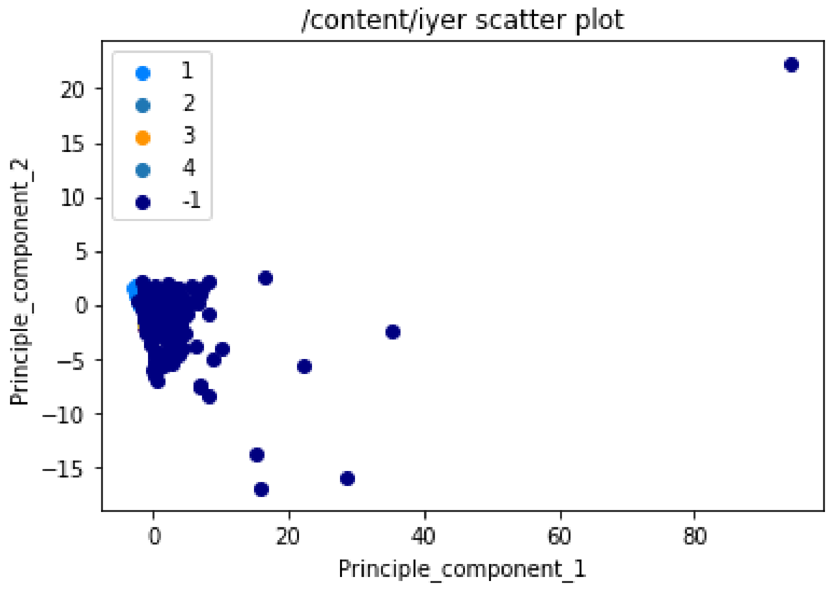


Figure 4.2 Scatter Plot for “iyer.txt” after DBScan clustering

With the specified values of ε and MinPts, 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: 0.2845701839961885
rand_index: 0.6460684876669074

The algorithm effectively clusters all the data points and identifies the non-density reachable data points as noise, which is represented by -1.

Different combinations of ε and MinPts provide a range for external index. The data is clustered into noise or with cluster IDs accordingly depending upon these input parameters.

The Jaccard co efficient varies from 0.17 to 0.28
Similarly, the rand index is noted to vary from 0.31 to 0.59

It can thus be observed that the similarity measure is highly affected on the neighborhood distance and minimum number of points in neighborhood. The data points that are more closely placed are clustered together and thus density along with the input parameter setting accounts as an important measure for detecting similarities using DBSCAN algorithm.

4.5 Result Evaluation:

1. DBScan algorithm assigns the labels to each point based on the number of neighbors that reachable within the epsilon distance.
2. Algorithm is capable of identifying the noise/outliers in the data.

Advantages:

• DBSCAN has a time complexity of O(n**2) and space complexity of O(n), where n is the number of data points.
• DBSCAN doesn’t require to specify the number of clusters, as opposed to k-means.
• DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster.
• DBSCAN is resistant to noise.
• DBSCAN can handle clusters of different shapes and sizes.
• DBSCAN requires just 2 parameters and is mostly insensitive to the ordering of the points in the database.

Disadvantages:

• DBSCAN cannot handle varying densities because then choosing a meaningful distance measure (ε) can be difficult.
• DBSCAN cannot handle cluster data sets well with large differences in densities, since the minimum points – ε combination cannot then be chosen appropriately for all clusters.
• Due to the above 2 points, it is hard to determine the correct set of parameters, and thus it is highly sensitive to the parameters.

• DBSCAN is not entirely deterministic. Border points that are reachable from more than one cluster can be part of either cluster, depending on the order the data is processed.
• The quality of DBSCAN depends on the distance measure used in the neighbours function. Mostly Euclidean distance is used. But for high dimensional data, this measure is rendered almost useless due to curse of dimensionality.

5. Spectral clustering:

- Spectral clustering is a technique where points which are connected or immediately next to each other are put in same cluster
- In spectral clustering, the data points are treated as nodes of a graph. Thus, clustering is treated as a graph partitioning problem.
- The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters.
- An important point to note is that no assumption is made about the shape/form of the clusters

5.1 steps for Spectral Clustering:

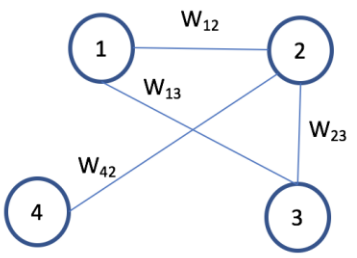
Step 1 — Compute a similarity graph:

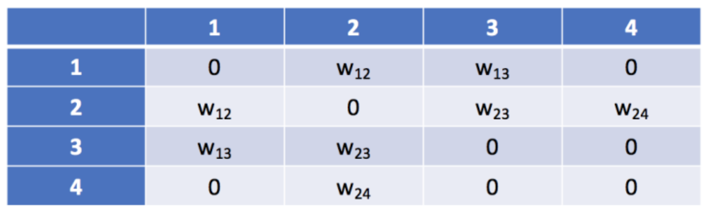
- We first create an undirected graph G = (V, E) with vertex set V = {*v1, v2, …, vn*} for n observations in the data. This can be represented by an adjacency matrix which has the similarity between each vertex as its elements
- We can compute this Similarity matrix in 3 ways:
  1) The ε-neighborhood graph:
  2) KNN Graph:
  3) Fully connected graph:
- For this project we have used Fully connected graph to find the similarity matrix:
- To construct this graph, we simply connect all points with each other, and we weight all edges by similarity *sij*. This graph should model the local neighborhood relationships, thus similarity functions such as Gaussian similarity function are used.

$$s(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Here the parameter σ controls the width of the neighborhoods

Example for creation of similarity or adjacency matrix;





	1	2	3	4
1	0	w_{12}	w_{13}	0
2	w_{12}	0	w_{23}	w_{24}
3	w_{13}	w_{23}	0	0
4	0	w_{24}	0	0

Step 2 — Project the data onto a low-dimensional space:

- Our goal then is to transform the space so that when the 2 points are close, they are always in same cluster, and when they are far apart, they are in different clusters.
- We need to project our observations into a low-dimensional space.
- For this, we compute the Graph Laplacian, which is just another matrix representation of a graph and can be useful in finding interesting properties of a graph. This can be computed as below:

$$L = D - A,$$

$$\text{where } A \text{ is the adjacency matrix and } D \text{ is the degree matrix such that}$$

$$d_i = \sum_{\{j|(i,j)\epsilon\, E\}} w_{ij} \qquad Thus, L_{ij} = \begin{cases} d_i & if\ i = j \\ -w_{ij} & if\ (i,j)\ \epsilon\ E \\ 0 & if\ (i,j) \notin E \end{cases}$$

Step 3 — Create clusters:

- For K clusters, compute the first K eigenvectors {v1,v2,v3,……..vk}
- Stack the vectors vertically to form a matrix with the vectors as columns.
- Represent every node by the corresponding row of this new matrix. These rows form the feature vector of the nodes
- Use K-means Clustering to now cluster these points into k clusters {c1,c2,c3……ck}

5.2 Implemented Algorithm:

1. The data is loaded as using input file path and then cast as a numpy array in to data. From this data, the first 2 columns are removed to get only the feature matrix and assign it to data_x.
2. Build the similarity matrix by calculating the weights using squared Euclidean distance for each i and j.
3. Find the degree matrix by summing all the row values and place in the respective diagonal position.
4. Find the Laplacian matrix by subtracting similarity and the Degree matrix.
5. Now find the Eigen values and Eigen vectors for the Laplacian matrix.
6. Find the maximum difference Eigen Values and consider Eigen vectors up to this point.
7. Now give this input to K-means algorithm and cluster them based on the input no of clusters.

5.3 VISUALIZATION:

File Name: cho.txt:

Here, the colors correspond to the Cluster ID assigned by the algorithm.
After experimenting with varied values of sigma and K, we have clustered the data by fixing the value of sigma 0.8 and the value of K to 5

Resulting clusters are as shown:

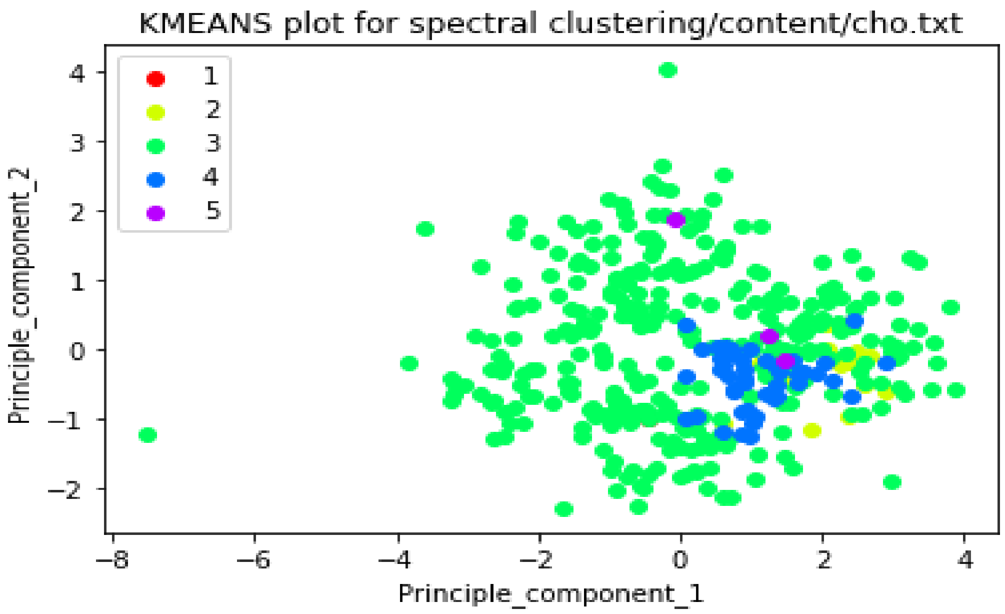


Figure 5.1 Scatter ploy for "cho.txt" after Spectral Clustering

The value of jaccard co-efficient and rand index is:

JaccardIndex: 0.19700140666277832

randIndex: 0.371660984187495

Different combinations of sigma and K provide a range for external index. The data is clustered based on the provided sigma value. The Jaccard co efficient varies from 0.20 to 0.23 Similarly, the rand index is noted to vary from 0.19 to 0.41 for sigma (0.7 to 1.1) and K = 5.

File Name: iyer.txt:

Here, the colors correspond to the Cluster ID assigned by the algorithm.
After experimenting with varied values of sigma and K, we have clustered the data by fixing the value of sigma 0.8 and the value of K to 5

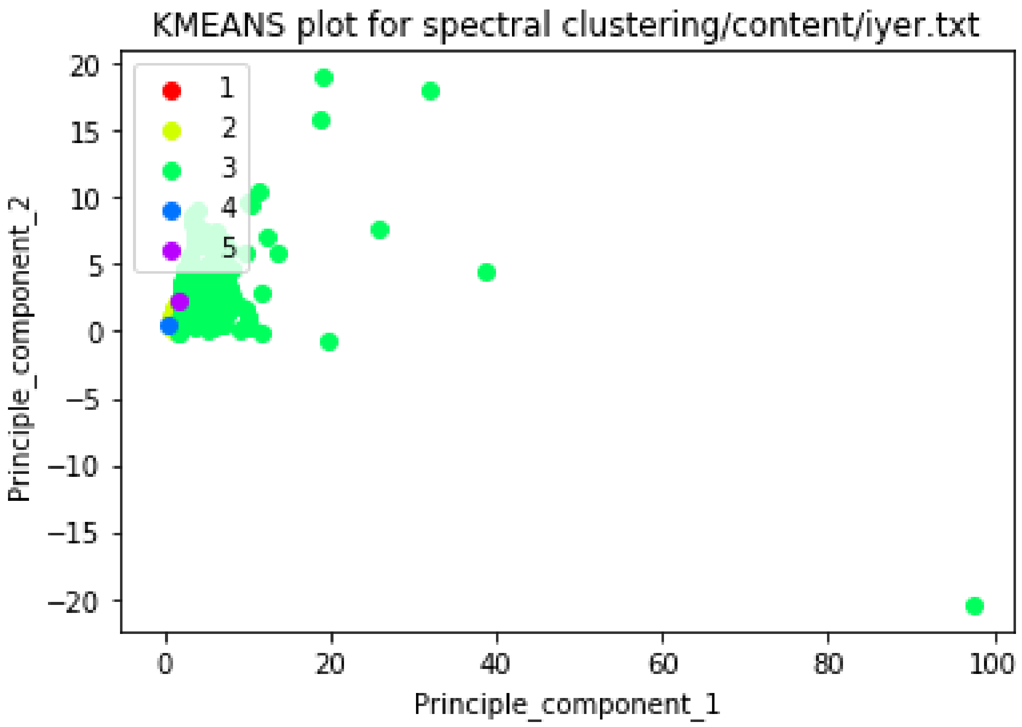


Figure 5.2 Scatter plot for "iyer.txt" after Spectral Clustering

The value of jaccard co-efficient and rand index is:

JaccardIndex: 0.28432979352357

randIndex: 0.6422224633262125

Different combinations of sigma and K provide a range for external index. The data is clustered based on the provided sigma value. The Jaccard co efficient varies from 0.15 to 0.28. Similarly, the rand index is noted to vary from 0.19 to 0.64 for sigma (0.7 to 1.1) and K = 5.

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:

1. Does not make strong assumptions on the statistics of the clusters — Clustering techniques like K-Means Clustering assume that the points assigned to a cluster are spherical about the cluster center. This is a strong assumption to make, and may not always be relevant. In such cases, spectral clustering helps create more accurate clusters.

2. 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.

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

Disadvantages:

1. 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.

2. 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.