CSE 601: Data Mining & Bioinformatics

Clustering Project Report

SUBMITTED BY:

MALINI ANBAZHAGAN (50289383)

MANDYAM BHARAT REDDY (50289569)

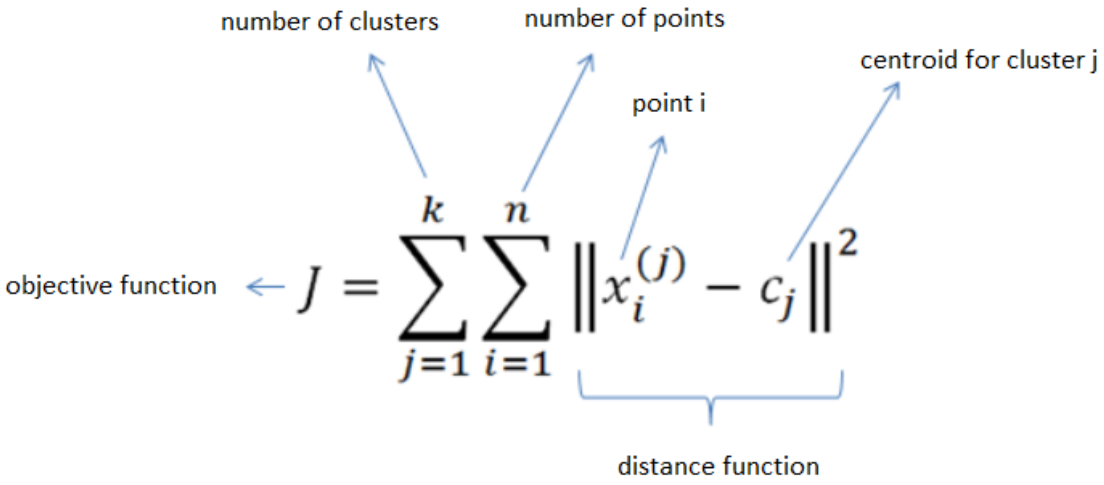
MONISHA BALAJI (50290962)

Table of Contents

K MEANS CLUSTERING

INTRODUCTION

K-Means clustering intends to partition n objects into k clusters in which each object belongs to the cluster with the minimum distance. This method produces exactly k different clusters of greatest possible distinction. The best number of clusters k leading to the greatest separation (distance) is not known as a priori and must be computed from the data. The objective of K-Means clustering is to minimize total intra-cluster variance, or, the squared error function



ALGORITHM

- Given: a data set of N points.
- Clusters the data points into k clusters.
- Initialize k points as cluster centers.
- Assign objects to their closest cluster center according to the Euclidean distance function.
- Calculate the centroid or mean of all objects in each cluster.
- Repeat steps 2, 3 and 4 until the centroid remains unchanged consecutively.

IMPLEMENTATION

- The algorithm is implemented using Python3.
- Dataset path, number of clusters and number of iterations are taken as input from the user.
- The user is given an option to enter k centroids or use k random centroids.
- The Euclidean distance of a point and centroid is calculated in eucl_dist() function.
- A dictionary is created to store initial cluster centroids.
- The method distance_matrix() calculates the euclidean distance of the points with each centroid using eucl_dist().
- The method calculate_centroids() updates the centroid by calculating the mean of the cluster.
- The method check_centroid() checks if there is a change in centroids from previous iteration.
- The method create_label() is used to assign cluster ID to each point that it belongs to and is stored in an array.
- Extract ground truth from the dataset and store it in an array.
- Calculate M00, M01, M10, & M11.
- Calculate Rand Index and Jaccard Coefficient using M00, M01, M10, & M11.
- Inbuilt PCA package is used from sklearn.decomposition to reduce the original data to 2 components for plotting.
- Data points are colored as per the cluster they belong to and cluster-ID is displayed as a legend in the plots.

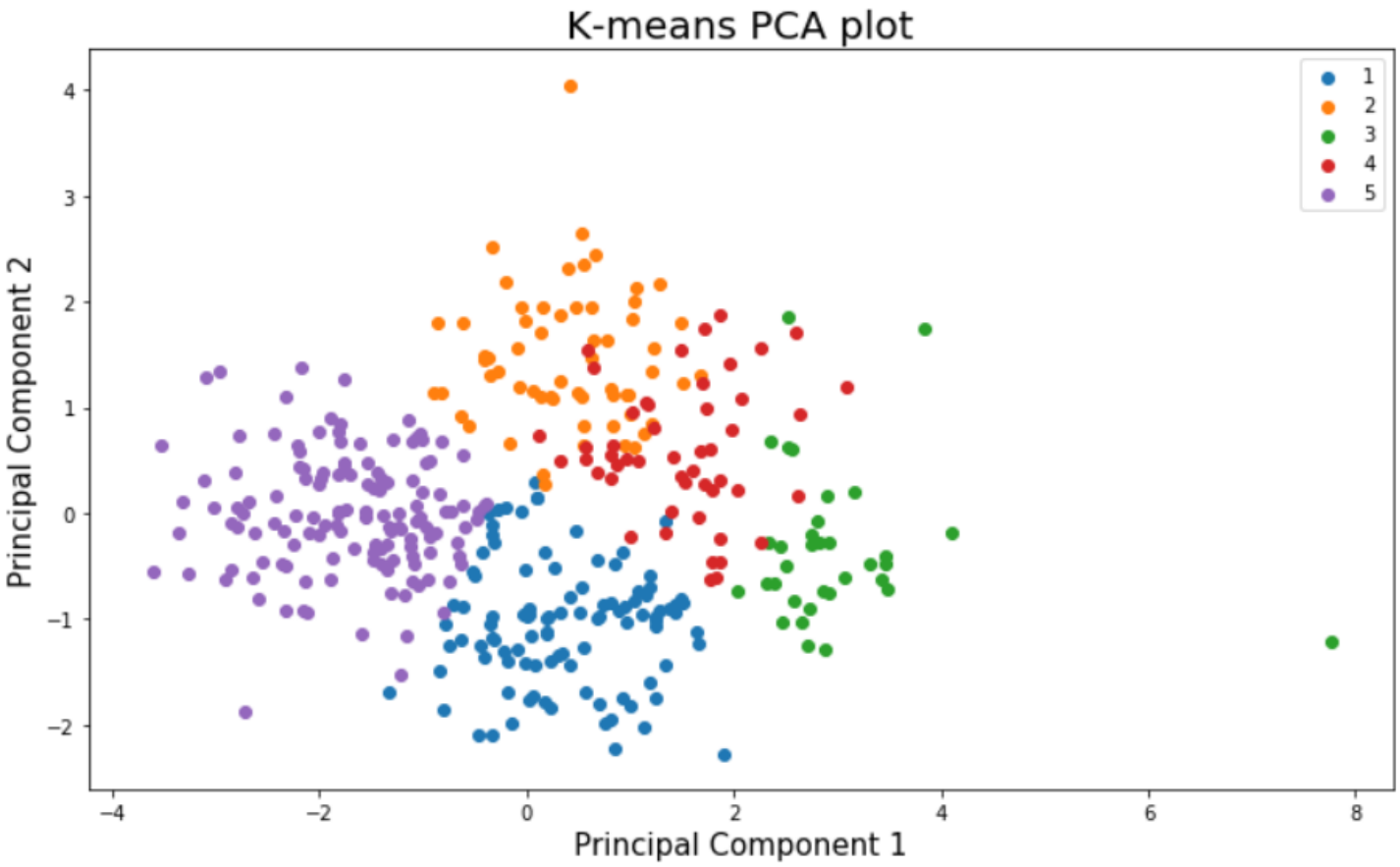
External Libraries used:

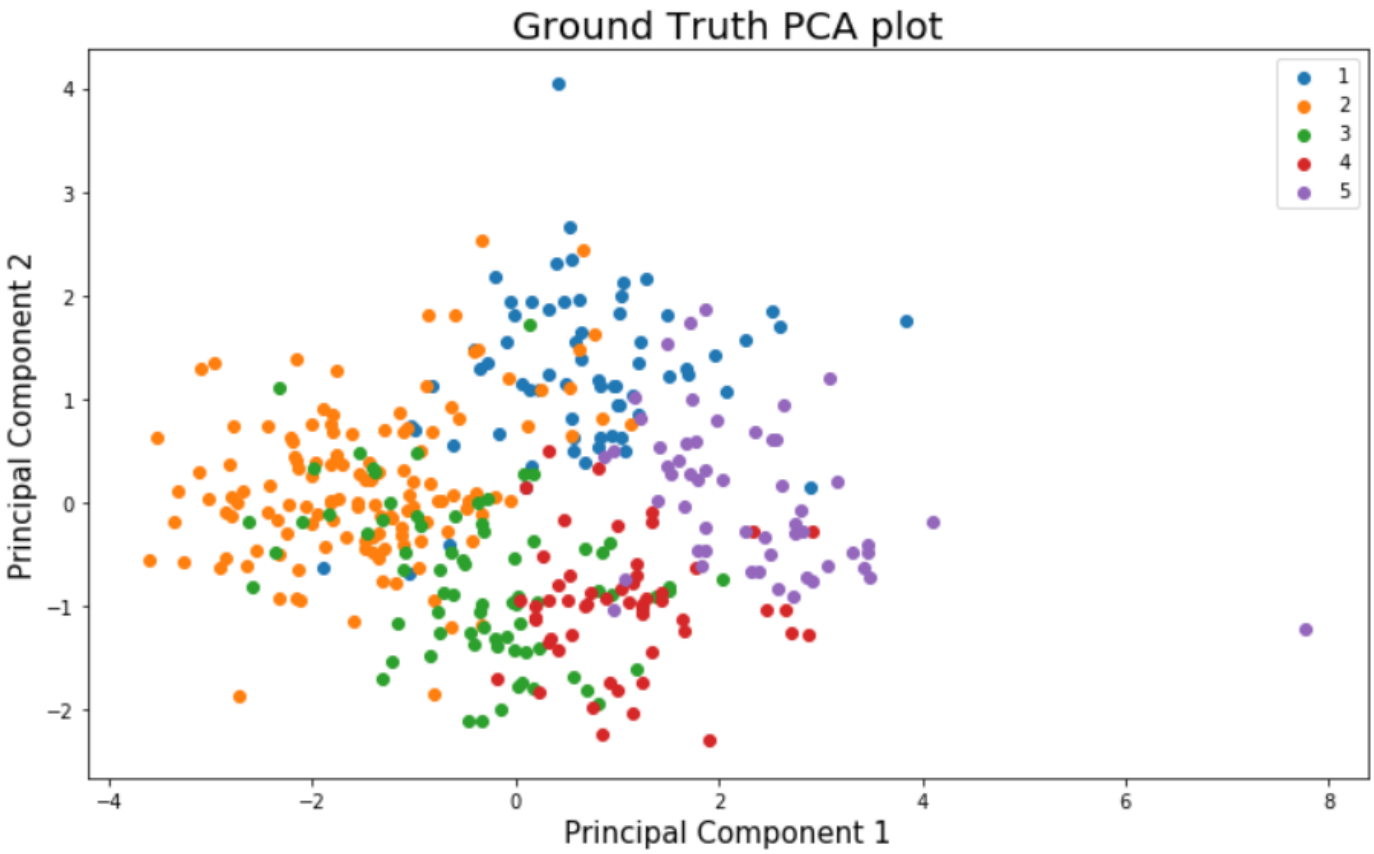
- pandas as pd: for reading dataset
- numpy as np: for arrays and Euclidian distance calculation
- PCA from sklearn.decomposition : to reduce the dimensions of data points
- matplotlib.pyplot as pl: for plotting clustering results
- Randint from random: to get random values

VISUALIZATION

a.) Cho.txt

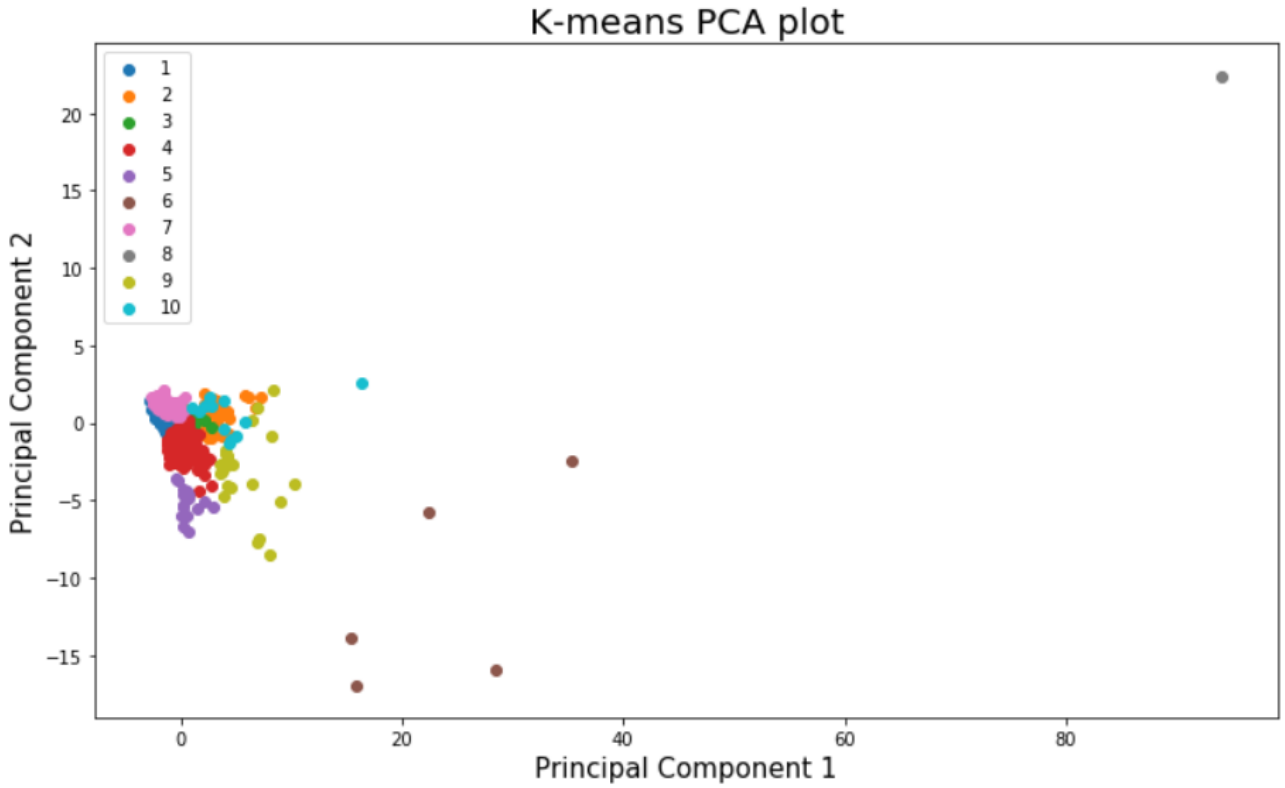
Rand Index: 0.7993100485919085
Jaccard Coefficient: 0.4095414873030291

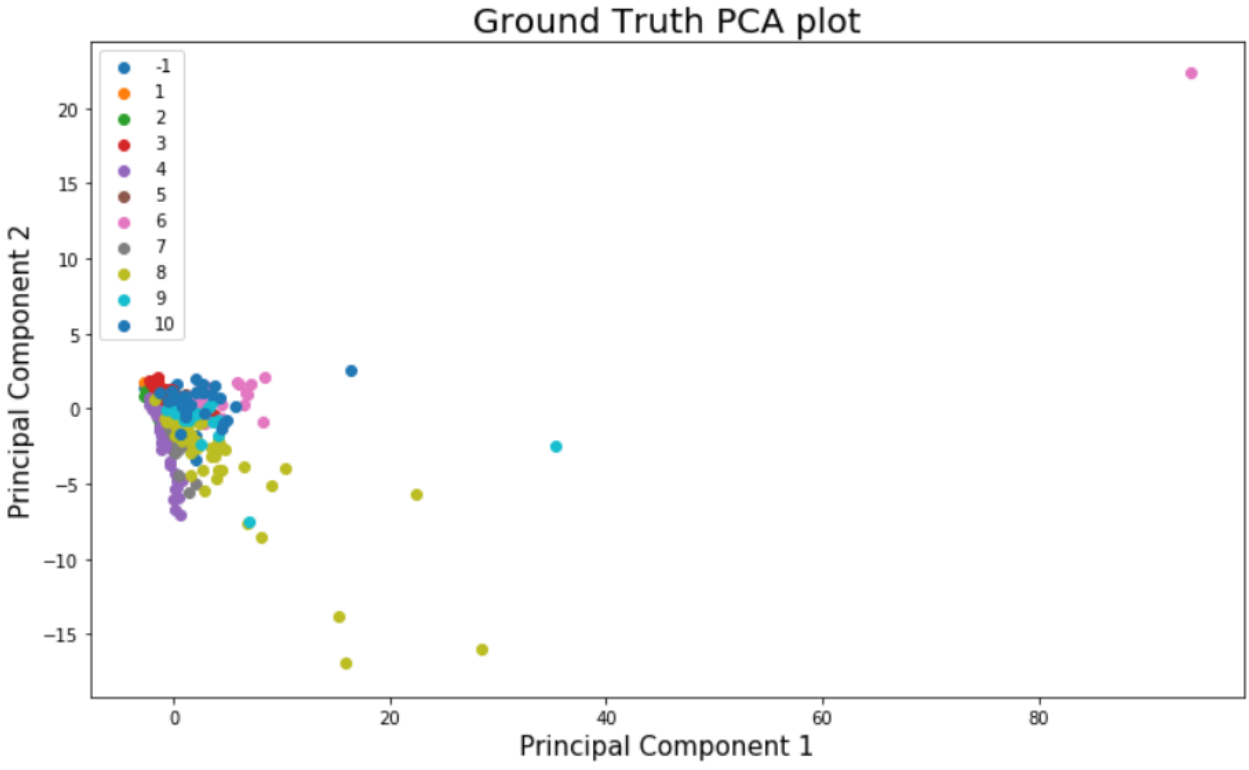

K-means PCA plot
Principal Component 2
Principal Component 1
1
2
3
4
5


Ground Truth PCA plot
Principal Component 2
Principal Component 1
1
2
3
4
5

b.) iyer.txt

Rand Index: 0.8285900280221035
Jaccard Coefficient: 0.3918848966698079





Pros for Kmeans:

- Simple: It is easy to implement k-means and identify unknown groups of data from complex data sets. The results are presented in an easy and simple manner.
- Flexible: K-means algorithm can easily adjust to the changes. If there are any problems, adjusting the cluster segment will allow changes to easily occur on the algorithm.
- Suitable in a large dataset: K-means is suitable for a large number of datasets and it's computed much faster. It can also produce higher clusters.
- Efficient: The algorithm used is good at segmenting the large data set. Its efficiency depends on the shape of the clusters. K-means work well in hyper-spherical clusters.
- Time complexity: K-means segmentation is linear in the number of data objects thus increasing execution time. It doesn't take more time in classifying similar characteristics in data like hierarchical algorithms.
- Spherical clusters: This mode of clustering works great when dealing with spherical clusters. It operates with an assumption of joint distributions of features since each cluster is spherical.

Cons for Kmeans:

- No-optimal set of clusters: K-means doesn't allow development of an optimal set of clusters and for effective results, you should decide on the clusters before.
- Sensitivity to scale: Changing or rescaling the dataset either through normalization or standardization will completely change the final results.
- Handle numerical data: K-means algorithm can be performed on numerical data only.
- Specify K-values: For K-means clustering to be effective, one has to specify the number of clusters (K) at the beginning of the algorithm.
- Prediction issues: It is difficult to predict the k-values or the number of clusters. It is also difficult to compare the quality of the produced clusters.
- Problem: K-means can't cluster effectively if the clusters vary in density, size, or has irregular shape.

HIERARCHIAL AGGLOMERATIVE CLUSTERTING

INTRODUCTION

Hierarchical Agglomerative Clustering is a clustering technique used to build a hierarchy of clusters. It follows a bottom-up approach, wherein, each point in the dataset is initially considered as a separate cluster. At each step, the nearest pair of clusters are merged until one cluster remains. Hierarchical clustering can be visualized using Dendrograms. Dendrograms are tree-like structures that depict the relationship of a cluster and its member points/sub-clusters as well as the order in which the clusters were merged.
Pair of clusters are merged using three different techniques – MIN, MAX, AVERAGE. In this project, we will be using the MIN approach which is also called a Single-Link approach. The Distance matrix is calculated using Euclidian distance.

ALGORITHM

- Given: a data set of N points.
- Compute Distance Matrix using Euclidian distance formula
- Let each point be a cluster
- Repeat Steps 4 and 5 until only a single cluster of size N remains
- Merge the two closest clusters (min distance in distance matrix)
- Update the distance matrix by calculating distances between the new cluster and each of the old clusters

IMPLEMENTATION

- The algorithm is implemented using Python3.
- Dataset path and number of clusters to be formed are taken as input from the user.
- All attributes of each point are extracted and stored in an array.
- The distance matrix is calculated using np.linalg.norm package. Function eucl_dist() is used to calculate Euclidean distance between two points.
- Function min_value() is used to retrieve the index of the point which has the least distance stored in the distance matrix.
- The cluster dictionary is created to maintain the mapping of each point to its cluster.
- Run a loop until the number of clusters is not equal to users' input for the number of clusters.
    - Merge the points which have the least distance and update the distance matrix by recalculating distances between the points/clusters.
- Function create_label() is used to assign cluster ID to each point that it belongs to and is stored in an array.
- Extract ground truth from the dataset and store it in an array.

- Calculate M00, M01, M10, & M11.
- Calculate Rand Index and Jaccard Coefficient using M00, M01, M10, & M11.
- Inbuilt PCA package is used from sklearn.decomposition to reduce the original data to 2 components for plotting.
- Data points are colored as per the cluster they belong to and cluster-ID is displayed as a legend in the plots.

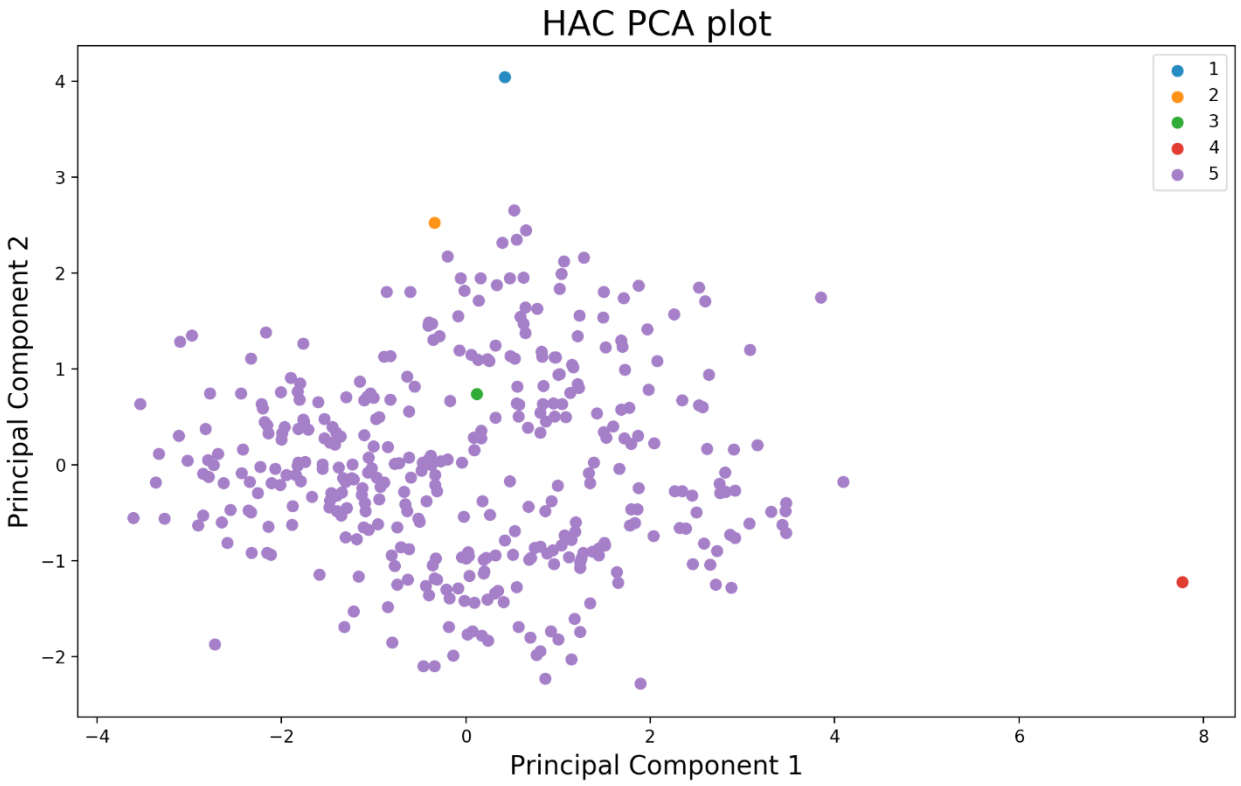
External Libraries used:

- pandas as pd for reading dataset
- numpy as np for arrays and Euclidian distance calculation
- PCA from sklearn.decomposition : to reduce dimensions of data points
- matplotlib.pyplot as pl for plotting clustering results

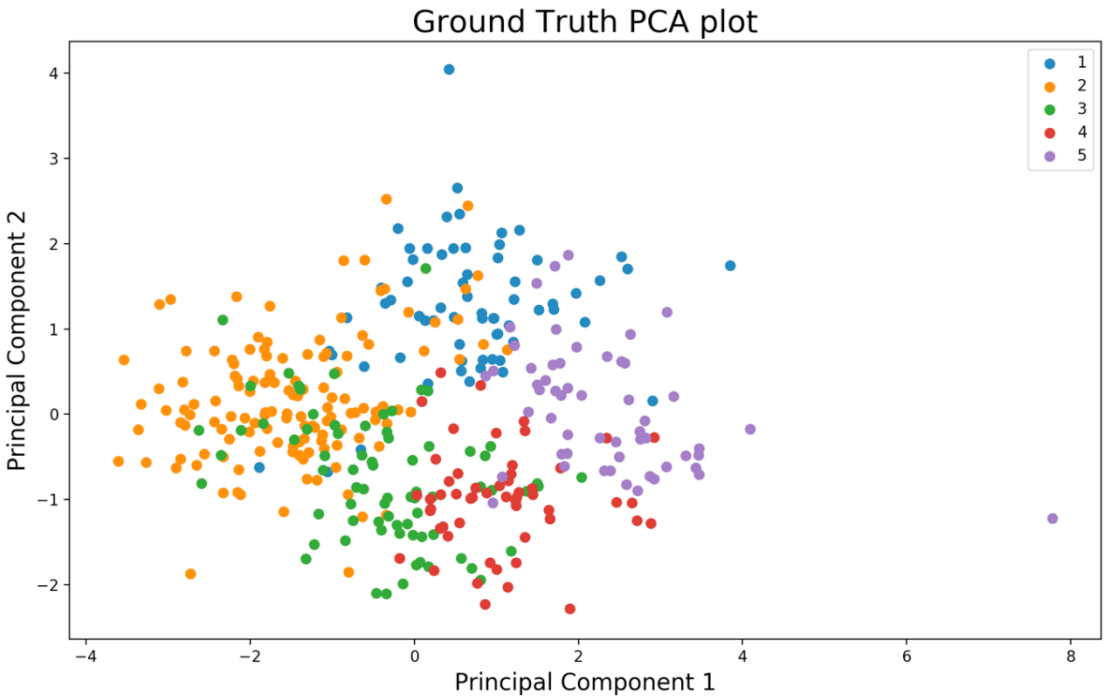
VISUALIZATION

a.) cho.txt

Rand Index: 0.24027490670890495
Jaccard Coefficient: 0.22839497757358454

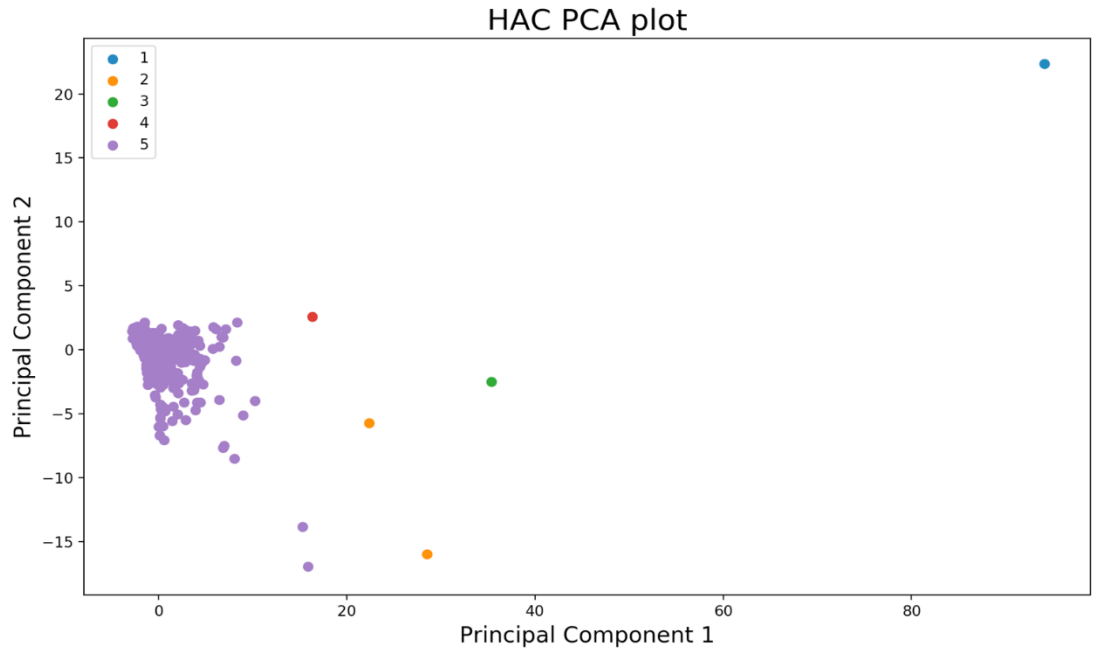


Different combinations of number of clusters provide a range for external index. The Jaccard co efficient varies from 0.22 to 0.24.

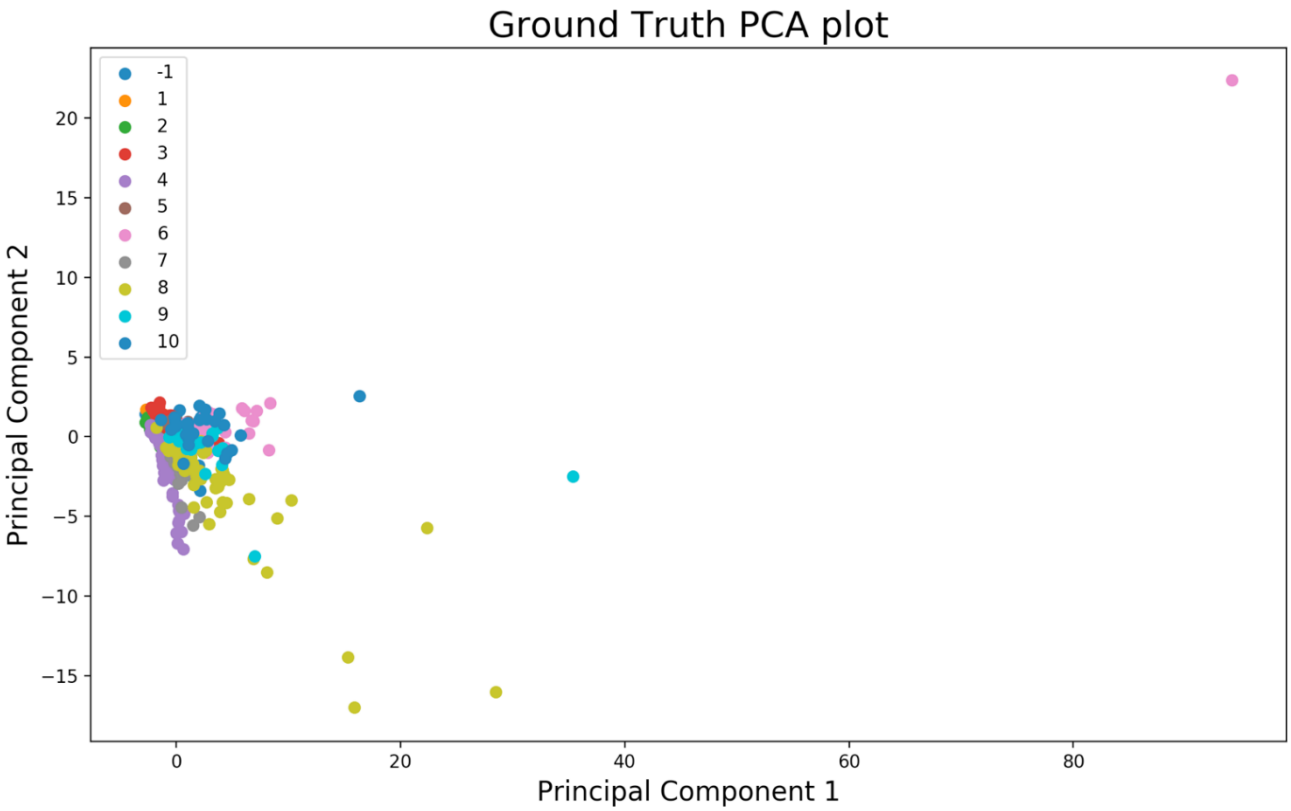


b.) iyer.txt

Rand Index: 0.17144364339722173
Jaccard Coefficient: 0.15647222380925174



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



Pros for Hierarchical Agglomerative Clustering:

- No assumption required for the number of clusters
- Produces ordering of cluster formation which can be useful
- Produces dendrogram which can be useful in visualization

Cons for Hierarchical Agglomerative Clustering:

- Time complexity is high
- Clusters may not have even distribution of points, i.e., some clusters may have many points and some clusters may have only one point
- A decision finalized cannot be taken back

DENSITY-BASED SPATIAL CLUSTERING AND APPLICATION WITH NOISE (DBSCAN)

INTRODUCTION

DBSCAN is a renowned data clustering algorithm that is commonly used in the field of data mining. It focuses on grouping points that are closely packed together which is determined by a distance measurement and marks the points that lie in the low-density regions as outliers. In DBSCAN, a cluster is a dense region that is margined by regions of lower object density. There are two important parameters involved in DBSCAN that determine the formation of clusters. These are:

- **ε-Neighborhood:** Objects within a radius of ε - *distance* from an object.
- **Minimum Points:** Helps establish "High density" regions, i.e, ε-Neighborhood of an object contains at least *MinPts* of objects.

ALGORITHM

The steps involved in implementing clustering using DBSCAN is given below:

1. ε - *distance* and *MinPts* are initially obtained.
2. For each data point in the data space obtain all neighboring points i.e. the points that are within it's ε - distance radius.
3. Check if the ε-Neighborhood count of the particular scanned point is more than MinPts.
4. If less, mark the particular point as NOISE or OUTLIER.
5. Else, mark this point as the centroid of the cluster or a core point and expand the cluster by adding all the points within it's ε-Neighborhood to the same cluster.
6. Repeat this process by scanning each point in the data space.
7. Each core point marks the formation of a new cluster.

PSEUDOCODE

The Pseudocode of the DBSCAN algorithm is shown below:

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

expandCluster(P, NeighborPts, C, eps, MinPts)

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

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

IMPLEMENTATION

1. Initially import all necessary packages for DBSCAN implementation.
   - Numpy
   - Pandas
   - PCA from sklearn.decomposition
   - matplotlib.pyplot
2. Input ε-Neighborhood and Minimum points.
3. Preprocess the dataset to extract the feature dimensions for each data point.
4. Implementation is broken down into 3 major functions.
   - **regionquery(eps, dis):** This function takes as input the ε-distance and the distance matrix formed based on Euclidean distance. Comparing the distance of the currently scanned point with all other data points, this function returns all the points that are within eps distance including itself.
   - **DBSCAN(points_arr, eps, MinPts):** This function takes as input the array of data points, eps distance and the count of Minimum points. It starts by scanning each point in the dataset. When a point is scanned for the first time it is marked as "Visited". Its neighboring points are collected by calling the 'regionquery' function. The count of neighboring points is checked to be greater than MinPts. If it is lesser, the point is marked as 'NOISE'. Else, 'expandcluster' function is called.
   - **expandcluster():** This function takes as input eps distance, MinPts along with the computed results from the 'DBSCAN' function. Mark the particular point as a core point and mark it as a new cluster. For each of the points that lie within its eps distance,

collect the points that are inside its corresponding eps distance. These give all the points that are density-reachable from the core point. If these points do not belong to any other cluster, add them to the current one.

5. The cluster dictionary is created to maintain the mapping of each point to its cluster.
6. Run a loop until the number of clusters is not equal to users' input for the number of clusters.
    a. Merge the points which have the least distance and update the distance matrix by recalculating distances between the points/clusters.
7. Function create_label() is used to assign cluster ID to each point that it belongs to and is stored in an array.
8. Extract ground truth from the dataset and store it in an array.
9. Calculate M00, M01, M10, & M11.
10. Calculate Rand Index and Jaccard Coefficient using M00, M01, M10, & M11.
11. Inbuilt PCA package is used from sklearn.decomposition to reduce the original data to 2 components for plotting.

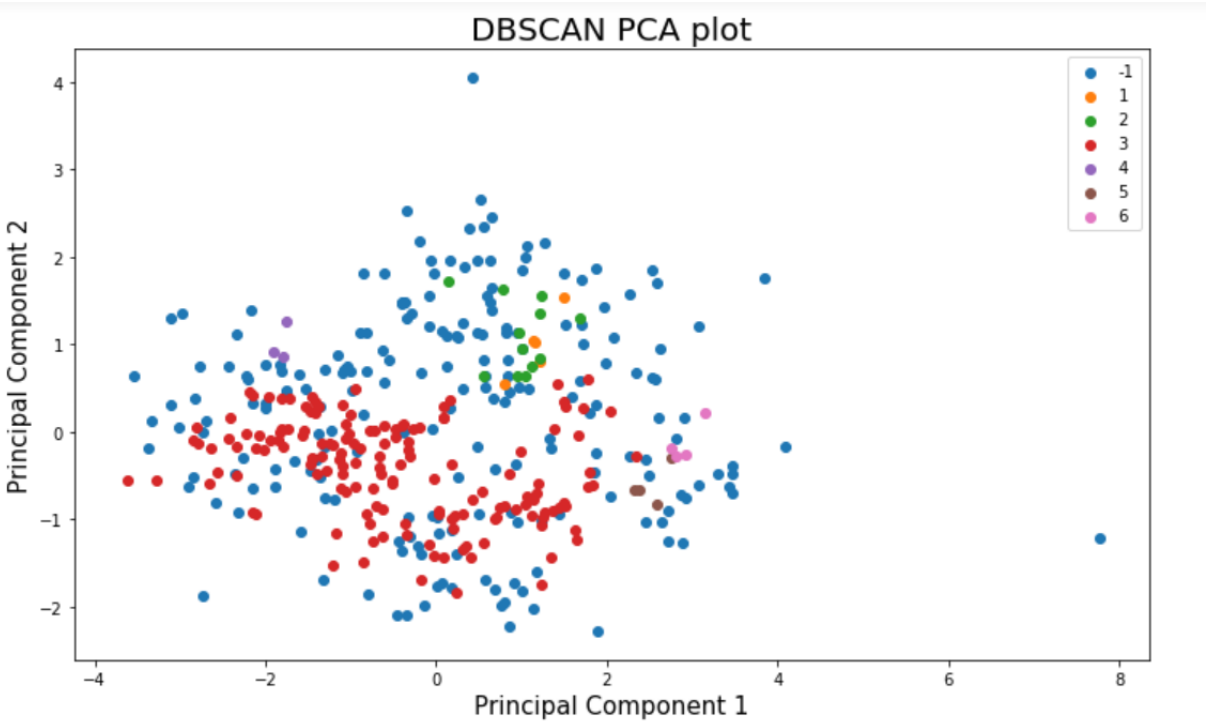
VISUALIZATION

a.) cho.txt:

Setting the values of eps = 1.03 and MinPts = 3,

Rand Index: 0.5544309914360117

Jaccard Coefficient: 0.20164510077444803

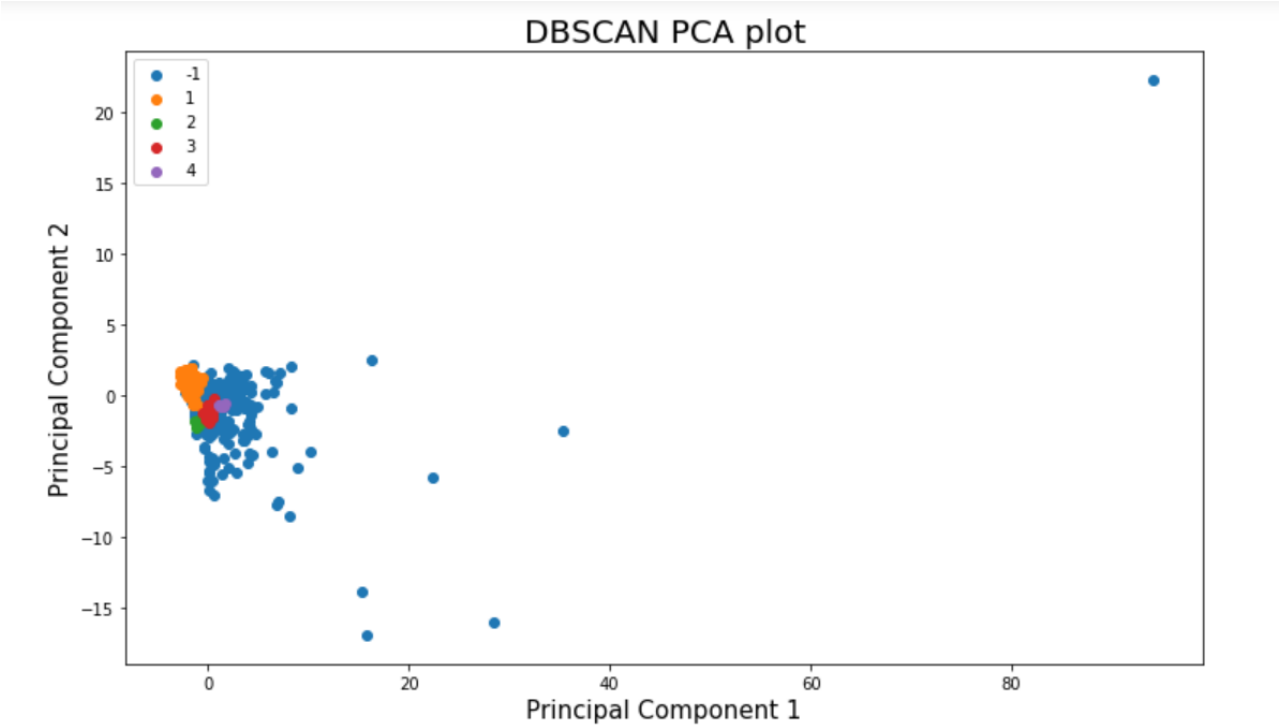


The graph shows that the data points are grouped into six different clusters with cluster indices ranging from 1-6 and the outlier points or the non-density-reachable points are marked as -1.

b.) iyer.txt:

Setting the values of eps = 0.8 and MinPts = 3,

Rand Index: 0.6460684876669074
Jaccard Coefficient: 0.2845701839961885



The graph shows that the data points are grouped into four clusters with cluster indices ranging from 1-4 for the given input parameters and the outlier points are marked as -1.

Pros for DBSCAN

- The algorithm is known to handle noise/outlier data points.
- The number of clusters to be formed does not need to be known in advance.
- DBSCAN is also capable of detecting clusters of arbitrary shapes; does not only focus on circular clusters like K-Means.

Cons FOR DBSCAN

- It faces difficulty when choosing precise parametric values for producing effective results i.e. it is hard to choose the correct set of parametric values if the data is not clearly comprehended.
- It does not work well with clusters of varying densities. It easily separates high-density clusters from low-density ones but struggles while handling clusters of similar density.
- The association of data points to the clusters depends on the order in which they are accessed.

GAUSSIAN MIXTURE MODELLING (GMM)

INTRODUCTION

GMM is a probabilistic clustering algorithm where each cluster is modeled based on a different Gaussian distribution. This algorithm is flexible is the aspect that it delivers soft assignments rather than hard assignments. This means that a point is not restricted to belonging to just one cluster but it can belong to any number of clusters within a particular probability range. Thus, the entire data set is modeled by a mixture of these distributions.

Two important parameters that need to be specified are:

- **μ (Mean):** Shifts the distribution left or right.
- **Σ (Covariance):** Increases or decreases the speed of the distribution.

ALGORITHM

The EM algorithm of GMM consists of two steps, E-step or the Expectation Step and the M-Step or the Maximization Step.

1. E-Step:

The E-step involves the estimation of the posterior probability of the data point on the basis of weight parameters, the mean and covariance which needs to be initialized. The Gaussian mixture distribution can be written as a linear combination of Gaussians.

The formula is given as:

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

Where $\sum_{k=1}^{K} \pi_k = 1, \quad 0 \leqslant \pi_k \leqslant 1$

In the above equation, π represents the weights and K the number of clusters to be modeled. Using the results from the above equation, we can compute the expected values of the latent variables. The formula for calculating the posterior distributions is given as:

$$\gamma(z_{nk}) = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\Sigma_{j=1}^{K} \pi_j N(x_n|\mu_j, \Sigma_j)}$$

2. M-Step:

This step involves computing the log-likelihood. The focus of this step is to maximize the likelihood value. This is given by the followed formula:

$$L(\mu,\Sigma)=\sum_{i=1}^{n}\ln N(x_i\,|\mu,\Sigma)=\sum_{i=1}^{n}(-\frac{1}{2}(x_i-\mu)^T\,\Sigma^{-1}(x_i-\mu))-\pi\ln|\,\Sigma\,|)$$

The next phase involves updating the learning parameters before the next iteration by the following formulae.

The weight parameter is updated by,

$$\pi_k=\frac{\sum_i r_{ik}}{n}$$

The mean is updated by,

$$\mu_k=\frac{\sum_i r_{ik}x_i}{\sum_i r_{ik}}$$

And finally, the covariance is computed by,

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

These two steps are repeated until the values converge, i.e. until the log-likelihood of data does not increase. A local optimal value should be reached.

IMPLEMENTATION

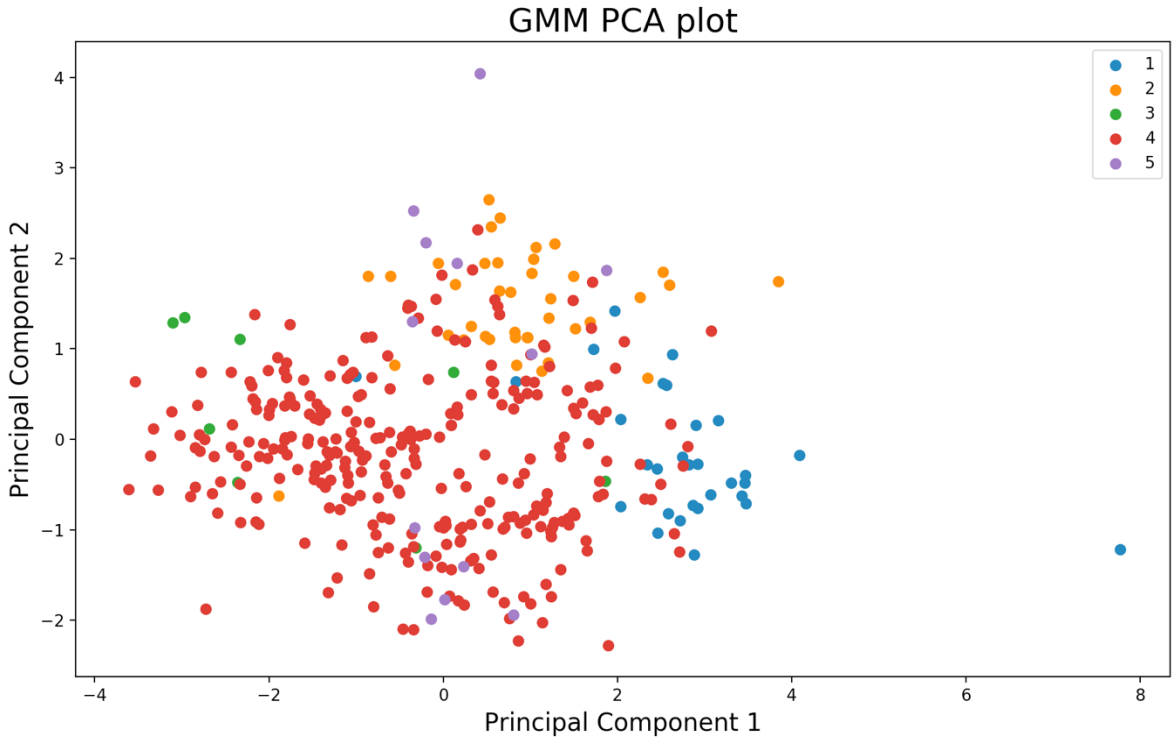
The work-flow of the algorithm is given below:

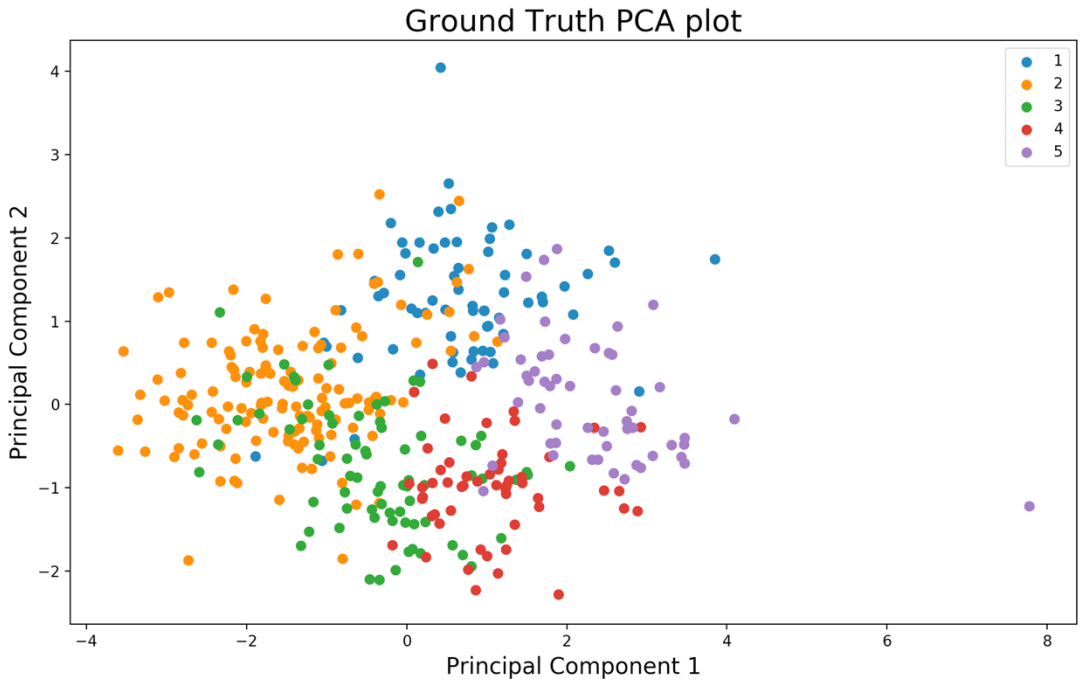
1. Initially import all necessary packages for DBSCAN implementation.
   - Numpy
   - Pandas
   - PCA from sklearn.decomposition
   - matplotlib.pyplot
2. We input two new parameters known as the smoothing value and threshold value. The smoothing value is used to help normalize the data to avoid singular matrix error or divide by zero error. The threshold value is used to determine the breaking of code implementation. When the difference between consecutive log-likelihood values goes below the threshold, the implementation is stopped or if the number of iterations are over the implementation is stopped.
3. Learning parameters such as mean, covariance and weights are either accepted from the user or randomly initialized.
4. Preprocess the dataset to extract the feature dimensions for each data point.
5. The implementation has two important functions:
   - Estep(): This function is used to compute the probability density function given by the formula mentioned above. The probability distribution for each of the points under all the clusters is calculated using scipy.stats **multivariate_normal.pdf**.
   - Mstep(prob): This function takes as input probability computed in the 'Estep'. The learning parameter values (mu,covariance and weights) are updated by their respective formulae to be passed on for the next iteration and are printed after each iteration.
6. Log likelihood is calculated using np.log on probability for each iteration and program is stopped if difference between two consecutive log likelihood values is less than the threshold value.
7. The cluster dictionary is created to maintain the mapping of each point to its cluster.
8. Run a loop until the number of clusters is not equal to users' input for the number of clusters.
   a. Merge the points which have the least distance and update the distance matrix by recalculating distances between the points/clusters.
9. Function create_label() is used to assign cluster ID to each point that it belongs to and is stored in an array.
10. Extract ground truth from the dataset and store it in an array.
11. Calculate M00, M01, M10, & M11.
12. Calculate Rand Index and Jaccard Coefficient using M00, M01, M10, & M11.
13. Inbuilt PCA package is used from sklearn.decomposition to reduce the original data to 2 components for plotting.

VISUALIZATION

a.) cho.txt

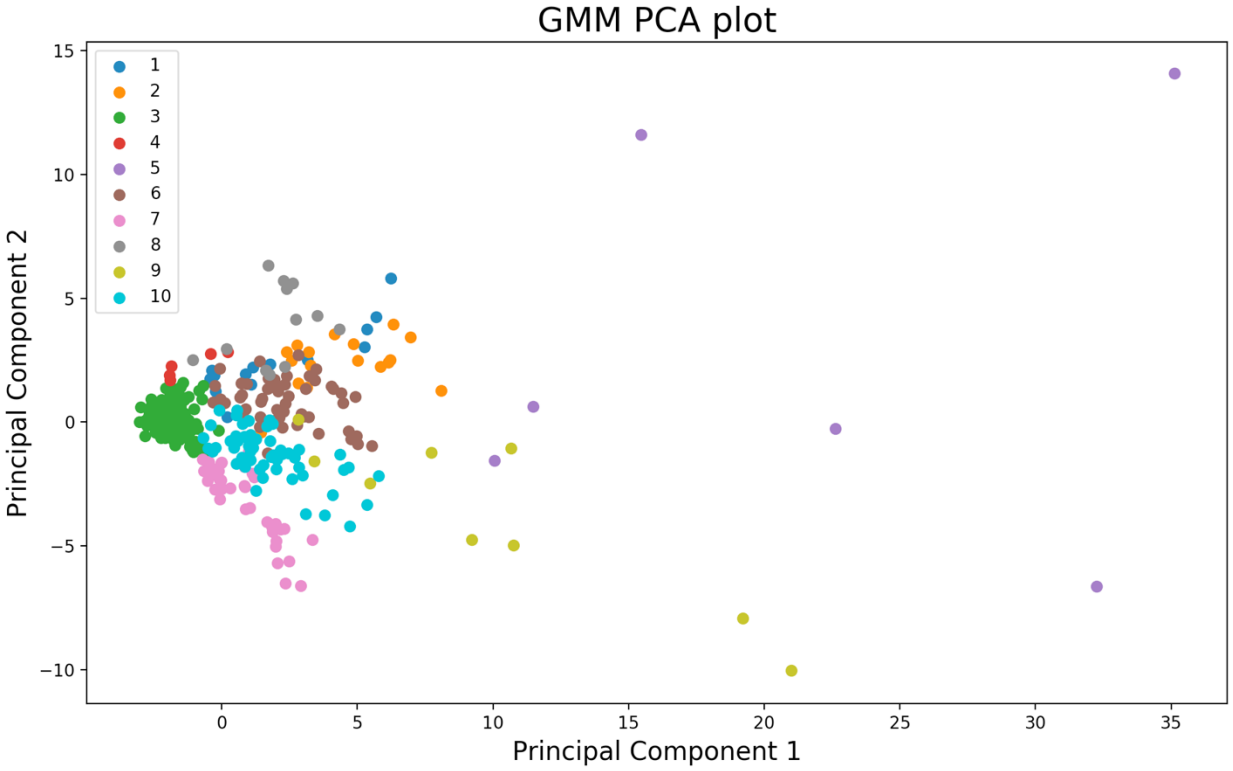
Rand Index: 0.4963220489140648
Jaccard Coefficient: 0.2480210024248983

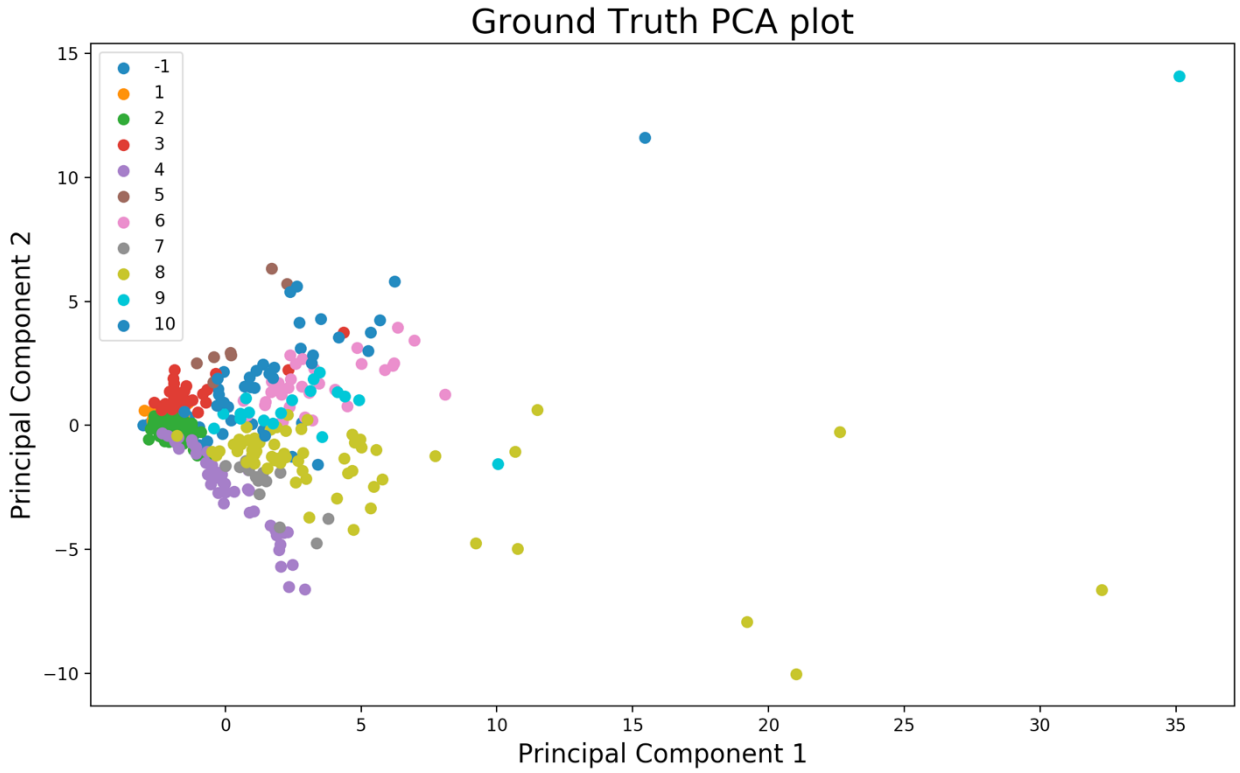




b.) iyer.txt

Rand Index: 0.7341505919115437
Jaccard Coefficient: 0.33857855687828214





Pros for Gaussian Mixture Modeling:

- The clusters are probabilistic which allows each data point to belong to any number of clusters within a probabilistic range.
- It works well with clusters of varying shapes and sizes.

Cons for Gaussian Mixture Modeling:

- It tends to fail to handle problems with high dimensionality.
- The Initialization of learning parameters is crucial and affects performance.

SPECTRAL CLUSTERING

INTRODUCTION

Spectral Clustering treats each data point as a graph-node and thus transforms the clustering problem into a graph-partitioning problem.

ALGORITHM

1. Building the Similarity Graph: This step builds the Similarity Graph in the form of an adjacency matrix which is represented by A. The adjacency matrix can be built in the following manners:-
   a.) Epsilon-neighbourhood Graph:
   b.) K-Nearest Neighbours
   c.) Fully-Connected Graph:
2. Projecting the data onto a lower Dimensional Space: The dimensional space is reduced so that those points are closer in the reduced dimensional space and thus can be clustered together by a traditional clustering algorithm. It is done by computing the Graph Laplacian Matrix . To compute it though first, the degree of a node needs to be defined. The degree of the ith node is given by

$$d_i = \sum_{j=1|(i,j)\epsilon E}^{n} w_{ij}$$

Thus the Graph Laplacian Matrix is defined as:-

$$L = D - A$$

3. Clustering the Data: This process mainly involves clustering the reduced data by using any traditional clustering technique – typically K-Means Clustering. First, each node is assigned a row of the normalized of the Graph Laplacian Matrix. Then this data is clustered using any traditional technique.

IMPLEMENTATION

- The algorithm is implemented using Python3.
- Dataset path, number of clusters and number of iterations are taken as input from the user.
- The similarity matrix is calculated using np.exp(np.linalg.norm()) where sigma is 1.5
- The degree matrix is calculated by summing up the columns in similarity matrix and store it diagonally.

- Laplacian matrix is calculated by taking the difference of degree matrix and similarity matrix.
- Calculated eigenvector using np.linalg.eig()
- The k-means is used again to cluster the data.

Pros for Spectral Clustering

- Clusters not assumed to be any certain shape/distribution, in contrast to e.g. k-means. This means the spectral clustering algorithm can perform well with a wide variety of shapes of data.
- Do not necessarily need the actual data set, just similarity/distance matrix, or even just Laplacian
    - We can cluster one dimensional data as a result of this

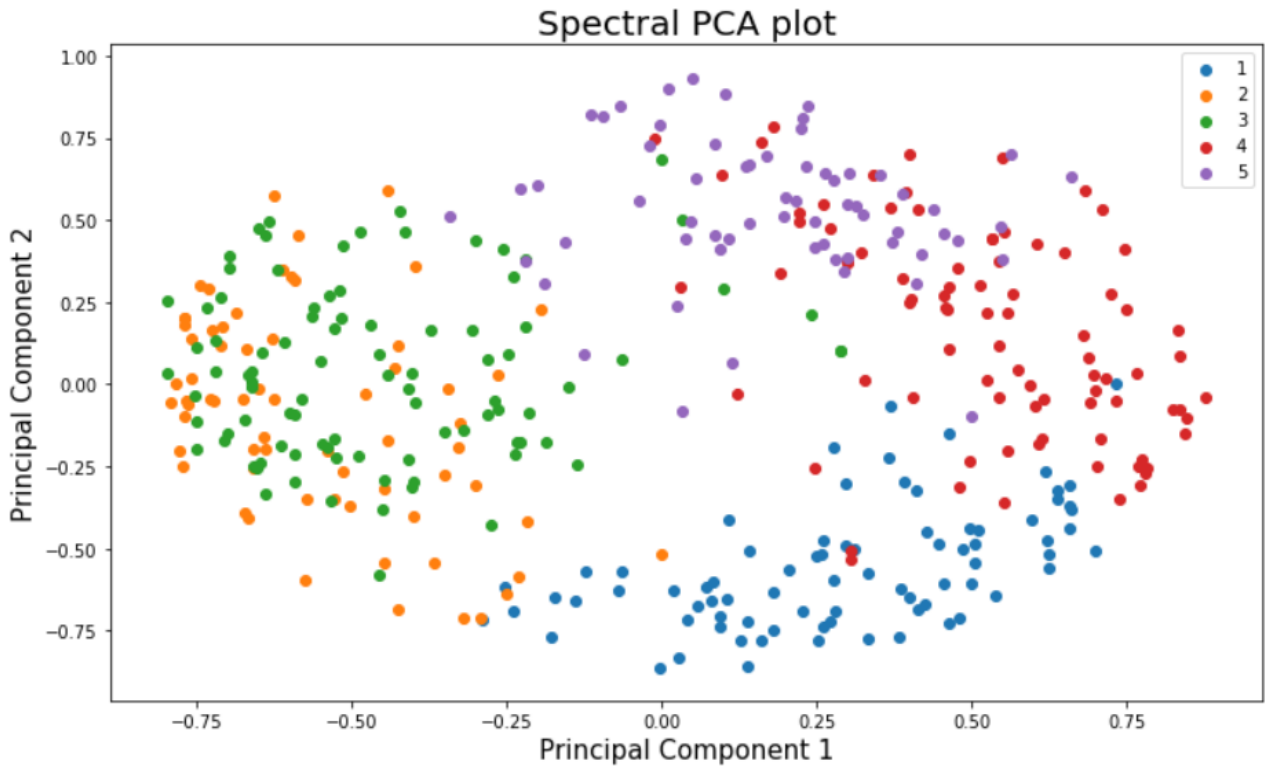
Cons for Spectral Clustering

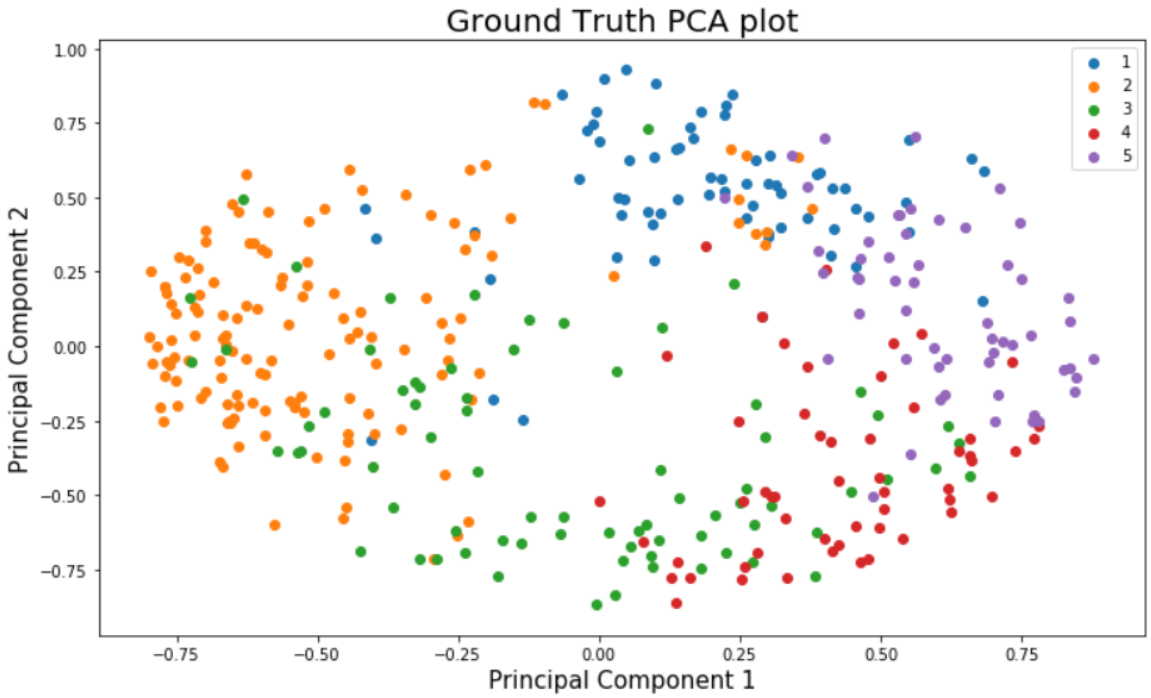
- Need to choose the number of clusters k
- Can be costly to compute, although there are algorithms and frameworks to help

VISUALIZATION

a.) cho.txt

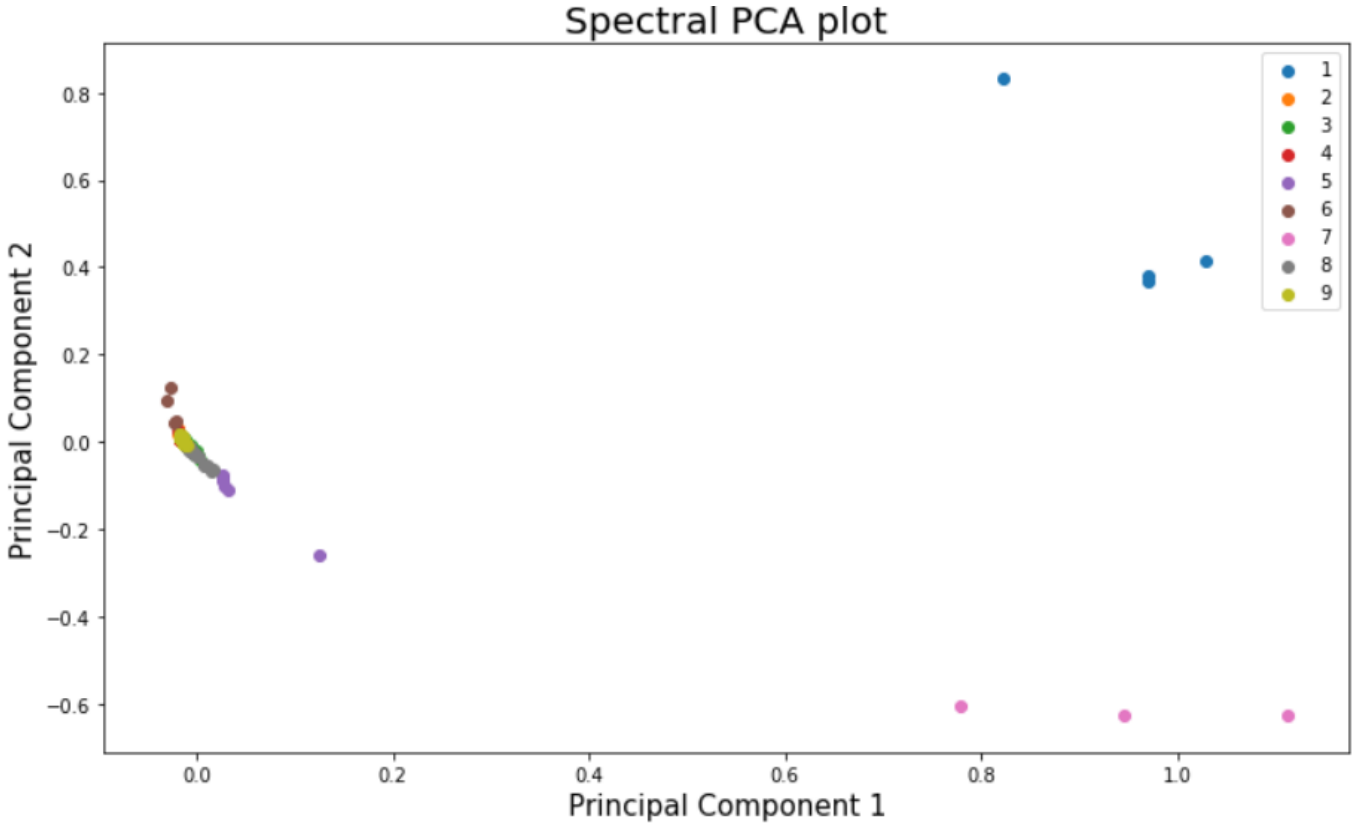
Rand Index: 0.7740878949770463
Jaccard Coefficient: 0.31571457613336046

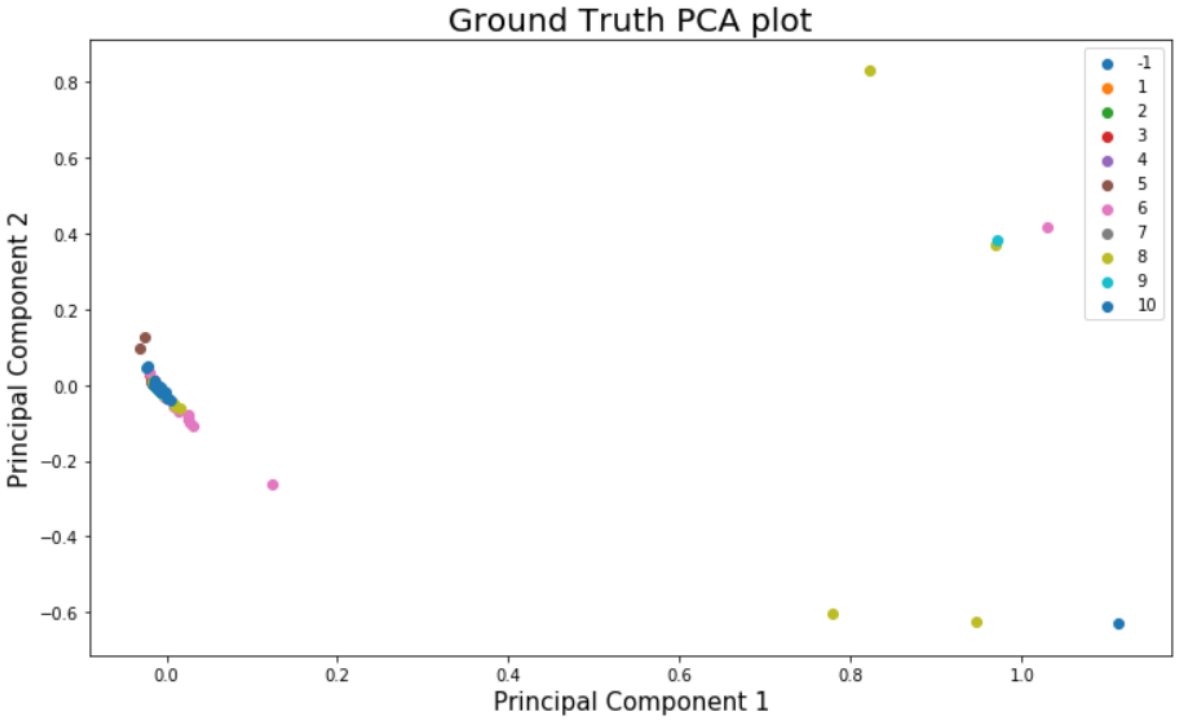




b) iyer.txt

Rand Index: 0.802266460647464
Jaccard Coefficient: 0.33530366103656006




Ground Truth PCA plot
Principal Component 2
Principal Component 1
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
0.2
0.4
0.6
0.8
1.0
-1
1
2
3
4
5
6
7
8
9
10