CSE 601 : Data Mining and Bioinformatics

Clustering Algorithms Project Report

Weiyang Chen
University at Buffalo, NY
wchen38@buffalo.edu

ZhiWen Huang
University at Buffalo, NY
zhiwenhu@buffalo.edu

ABSTRACT

The project aim to implement five clustering algorithms to find clusters of genes that exhibit similar expression profiles: K-means, Hierarchical Agglomerative clustering with Min approach, density-based, mixture model, and spectral clustering. Compare these five methods and discuss their pros and cons. For each of the above tasks, validating the clustering results using the following methods:

- **Using external index (Rand Index and Jaccard Coefficient) and compare the clustering results from different clustering algorithms. The ground truth clusters are provided in the datasets.**
- **Visualize data sets and clustering results by Principal Component Analysis (PCA).**

1 DATA

1.1 Description

Each row represents a gene:

(1) the first column is gene_id.
(2) the second column is the ground truth clusters. You can compare it with your results. "-1" means outliers.
(3) the rest columns represent gene's expression values (attributes).

1.2 Pre-Processing

Read txt file, and Output data to three variables which are 'id' only contains first column contents of input data, 'result' only contains second column contents of input data and 'gene' contains rest columns contents of input data, All the return variables are the numpy.array data type.

2 K-MEAN

2.1 Introduction

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. It aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest cluster centers. In each iteration, it will re-calculate the cluster centers by taking the mean of all points in the same cluster, and will stop until the results converged or reach to the fixed iteration number.

Supervised by Jing Gao.

2.2 Pseudo-code

Algorithm 1: K-Means Algorithm

```
Choose the number of clusters(K) and obtain the data points;
Randomly select K center points;
while Not convergence or not reach to # of iterations do
    foreach data point $x_i \in X$ do
        Find nearest center point by SSE;
        Assign the point to that cluster;
    end
    foreach cluster $c_j \in C$ do
        $c_j$ = mean of all points assigned to the $c_j$;
    end
end
```

2.3 Implementation

(1) Our K_mean function accept series parameters :
- *k* (integer type, Number of Parameter)
- *gene* (numpy.array type, Matrix of data features)
- *central* (array type, Initial k center points)
- *iteration* (integer type, max number of iteration)

(2) Define the variables :
- *previous_error* = sys.maxsize
- *current_error* = 0
- *iter* = 0
- *geneGroup* = [[] for _ in range(k)]

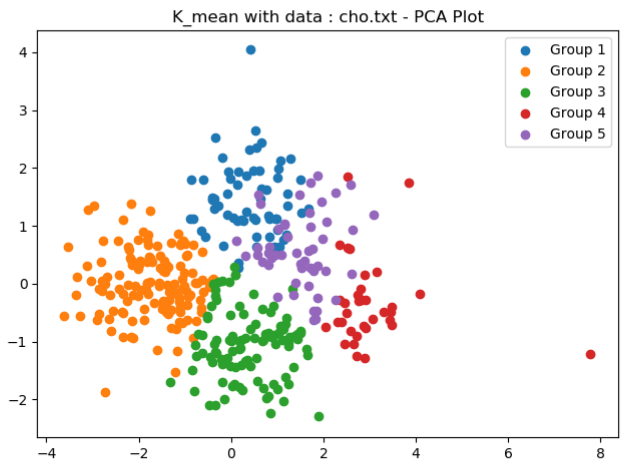
previous_error and *current_error* will be used during the iteration, if two variables are the same, it means the iteration not changed and result is converged, *iter* is count current iteration time, in each iteration it will +1 .

(3) Using while loop with condition if $(iter! = k)$
  (a) *current_error* = 0
  (b) *geneGroup* = [[] for _ in range(k)]
  (c) Iteration each point in the gene (Each row of gene is a point)
    (i) Calculate the distance by SSE between current point with all center points. we will get a array *error* which is distance from current array to k center points.
    (ii) Find minimum value *e* in the *error* array and its index *e_index*.
    (iii) Add *e* into *current_error* and put current point into cluster : $geneGroup[e\_index].append(currentpoint)$
  (d) if $previous\_error! = current\_error$ then $previous\_error = current\_error$ (Means it not converged yet) else break the while loop (Means *previous_error* is equal to *current_error* and clusters result not changed, it is converged)
  (e) *iter* += 1

   (f) Iterator each clusters (For i from 0 to len(geneGroup)) :
      (i) Calculate the cluster's new center by calculating the mean of all points in this cluster ($geneGroup[i]$).
(4) After the end of while loop, we will get 2D array $geneGroup$ with length k. $geneGroup[i]$ will return the ith cluster that contains all points' index belong to this cluster.
(5) Using the geneGroup to compute the Jaccard Coefficient and RandIndex of the clustering result.

2.4 Result

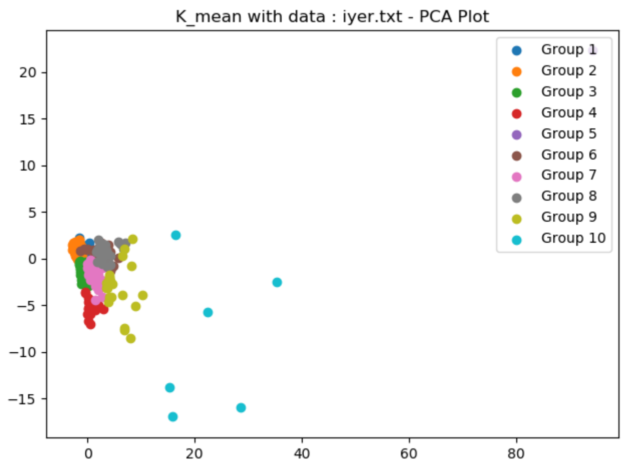
2.4.1 Data file : cho.txt. Input parameter : k = 5, iteration = 30



Jaccard Coefficient is : 0.4095414873030291

Rand Index is : 0.7993100485919085

2.4.2 Data file : iyer.txt. Input parameter : k = 10, iteration = 50



Jaccard Coefficient is : 0.3468332185044284

Rand Index is : 0.7552686418071825

2.5 Discussion

2.5.1 Pros.

- Its very simple to implement it.
- Computing time is fast. (RT : $O(t*k*n)$ where t is iterations, k is number of clusters and n is the size of data)

2.5.2 Cons.

- K-Mean result will be affected by initial center points we pick, which means it is very hard to reach global optimal clustering solution.
- K value pick is also a challenge.
- Sensitive to the outlier points, the outlier will greatly change the center point of a cluster.
- Not suitable for all types of data. For example, irregular shapes and varying density data.

3 HIERARCHICAL AGGLOMERATIVE CLUSTERING (MIN APPROACH)

3.1 Introduction

In data mining and statistics, hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

(1) Agglomerative: This is a "bottom-up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
(2) Divisive: This is a "top-down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

In general, the merges and splits are determined in a greedy manner. The results of hierarchical clustering are usually presented in a dendrogram.[2]

In this problem, we will implement Hierarchical agglomerative clustering with min approach which is a "bottom-up" approach, and in each step, will merge two cluster that the distance between a pair of closest point is global minimum.

3.2 Pseudo-code

Algorithm 2: Hierarchical Agglomerative Clustering (Min Approach)

```
Choose the number of clusters(K) and obtain the data points;
Initialize a distance matrix with size N*N (N is the size of
  the data and calculate the distance between two points by
  using Euclidean Distance Formula.) ;
while len(matrix) != K do
    Find smallest distance's index [i , j];
    Merge clusters i, j into i;
    for index = 0 -> len(matrix);
     do
        $matrix[i, index] = min(matrix[i, index], matrix[j, index])$;
        $matrix[index, i] = min(matrix[index, i], matrix[index, j])$;
    end
    Delete matrix's j row, j col;
end
```

3.3 Implementation

1, creating a N*N distance matrix which N is the size of the data, and we will use Euclidean Distance Formula to calculate the distance between two points.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{1}$$

The equation 1 is known as the Euclidean Distance, n is the number of dimensional, and p_i, q_i represent ith features in point p and q.

2, Initialize the return array geneGroup as a 2D array, and each point belong to unique cluster in the beginning. For example geneGroup[0] = [0].

3, Using a while loop, if condition len(matrix) != K (K is the number of final clusters) satisfy, then keep running step 4 - 5 until len(matrix) == K

4, Find the smallest value's index in the distance matrix. And merge the ith and jth array into ith array in the geneGroup, then delete jth array from geneGroup. For example we merge geneGroup[1] = [1], geneGroup[2] = 2 into geneGroup[1] then it will be [1,2] and delete geneGroup[2] from the geneGroup.

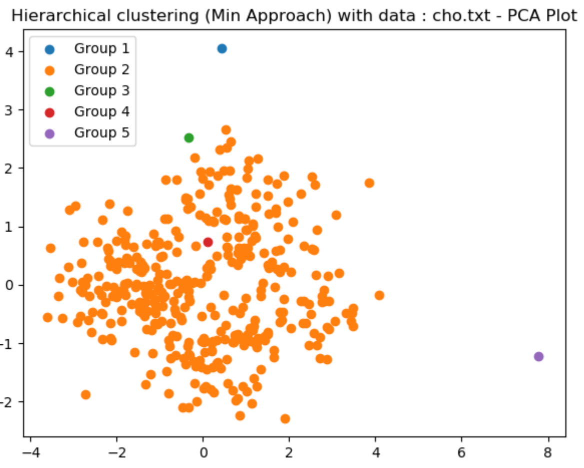
5, After merging the geneGroup, we need to update the matrix. which update the i's row and col in the matrix and delete j's row and col in the matrix. In order to keep min approach, we need to update with min distance from other clusters to new i clusters (After merging). For x from 0 to len(matrix) : matrix[i,x] = min(matrix[i,x],matrix[j,x]) , matrix[x,i] = min(matrix[x,i],matrix[x,j])

6. return the geneGroup, it will contain K arrays, which each array is a clusters that contains points' index.

7. Using return result and true ground result from data set to calculate Jaccard Coefficient and Rand Index.

3.4 Result

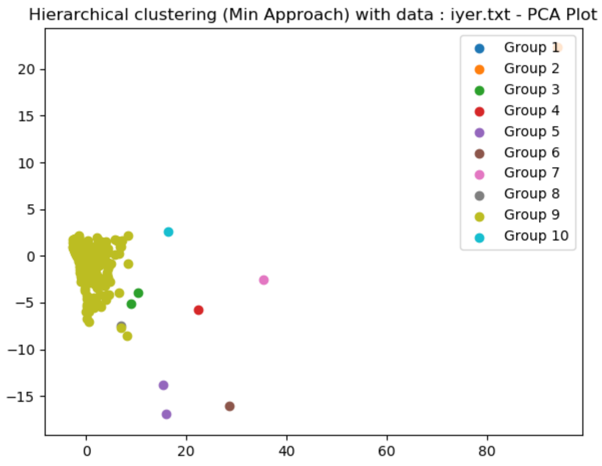
3.4.1 *Data file : cho.txt.* Input parameter : k = 5



Jaccard Coefficient is : 0.22839497757358454

Rand Index is : 0.24027490670890495

3.4.2 *Data file : iyer.txt.* Input parameter : k = 10



Jaccard Coefficient is : 0.1584680239258938

Rand Index is : 0.1941456625600006

3.5 Discussion

3.5.1 *Pros.*

- Easy to decide the number of clusters k by looking at the dendrogram.

3.5.2 *Cons.*

- Min Approach is sensitive to the outliers, For example, in the cho.txt data set, the one cluster will keep merge others clusters, and in the end, the other four outlier point are the left four clusters, the result isn't correct.
- Once decision of merging two clusters made, it cannot be undone.
- No Objective function is directly minimized.
- It might run slower with large dataset and it take a large amount of space : It took N*N space in order to create a distance matrix, and RT will be $O(n^3)$.

4 DENSITY-BASED CLUSTERING

4.1 Introduction

Density-based clustering is a non-parametric which given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions.[1]

In general it has two parameters, epsilon which is the radius distance of point, and *minPts*. There are three types points in the density clustering :

- Core Point : A point has at least *minPts* neighbor within epsilon radius.
- Border Point : A point does not have at least *minPts* neighbor within epsilon radius, but one of its neighbor is core point.
- Outlier Point : A point does not have at least *minPts* neighbor within epsilon radius and none of its neighbor is core point.

4.2 Pseudo-code

Algorithm 3: Algorithm for Excluding the Negation

```
Function DBSCAN($D, eps, MinPts$):
    foreach unvisited point $P \in D$ do
        mark $P$ as visited
        $NeighborPts$ = regionQuery($P, eps$)
        if $sizeof(NeighborPts) < MinPts$ then
            mark $P$ as noise
        else
            $C$ = next cluster
            expandCluster($P, NeighborPts, C, eps, Minpts$)
        end
    end
End Function
Function
 expandCluster($P, NeighborPts, C, eps, MinPts$):
    add $P$ to Cluster $C$
    foreach point $P' \in NeighborPts$ do
        if $P'$ is not visited then
            mark $P'$ as visited
            $NeighborPts'$ = regionQuery($P', eps$)
            if $sizeof(NeighborPts') >= MinPts$ then
                $NeighborPts = NeighborPts$ joined with
                 $NeighborPts'$
            else
                ;
            end
        else
            ;
        end
        if $P'$ is not yet member of any cluster then
            add $P'$ to cluster $C$
        else
            ;
        end
    end
End Function
Function regionQuery($P, eps$):
    return all points within $P's$ eps-neighborhood
     (including $P$)
End Function
```

4.3 Implementation

1. Initialize visited, in_cluster, noise as set data type, to help us to build the function later, and a geneGroup as array type that store all the points that belong to the same cluster.

2. Implement regionQuery function, which is taking parameters P and *eps*, P is a data point, and *eps* is the epsilon, so the function will euclidean distance formula to calculate all points distance to this point, and if the distance is less or equal to *eps*, then add the point into return array. So after return, we will get all the neighbors of input point P that within *eps* radius.

3. Iterator each data point in the data set, check if the point in the visited set or not, if it is in the visited set then it's means the point has already visited and go next point, if it is not in the visited set, then add it into visited set and using *regionQuery* function to

find its all neighbors that within *eps* radius around it. Check the size of neighbors' array, if size less then *MinPts*, it means this point is not core point, then add *P* into noise set. if size equal or greater than *MinPts*, it means this point is a core point, then we will add a empty array [] into geneGroup, means it is a new cluster, and run the expandCluster function.

4. In the expandCluster function, it takes parameter *geneGroup*, *NeighborPts*, *P*, *eps* ,*MinPts*, *visited* and *in_cluster*. So first it will add *P* into last array in the geneGroup, (We add new cluster as empty array in the geneGroup, and right now it should be in the last index of geneGroup) and add *P* into in_cluster set means this point *P* is already clustered. Then iterator all the points P' in the *NeighborPts* (It should be all the neighbors points around *P*), and repeat 4a - 4b until iteration finish.

4a. Check P' is in the visited set or not, if not in the visited set then do nothing, else add P' into visited set, run the regionQuery with parameter P', *eps* to get neighbor array $NeighborPts'$ of P', to check the size of $NeighborPts'$, to determine the P' is a core point or not. if P' is a core point, then Joint the *NeighborPts* and $NeighborPts'$.

4b. After Check P' is in the visited set or not, Then check if P' in *in_cluster* or not. If P' is not in the cluster then add it to the current cluster which is the way add P' in to last index array in the *geneGroup* (Because the *P* is core point, then its neighbors should be either border point or core point and stay the same clusters)

4c. return the geneGroup as return value for expandCluster function.

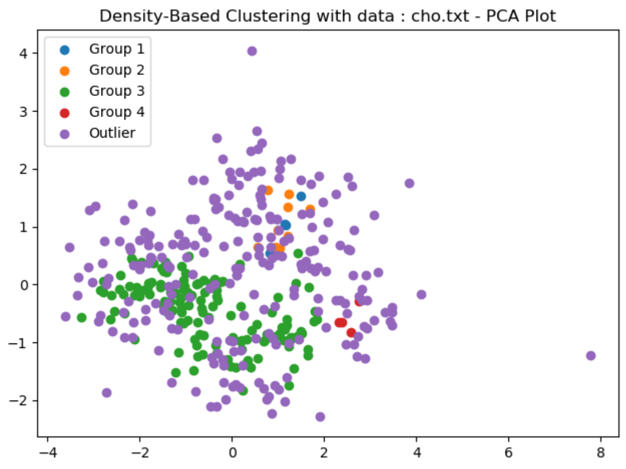
5. After the end, when step 3 finish iteration, we will get all clusters in the geneGroup, each index array in the geneGroup is a cluster that contains all the points belong to this cluster.

6. Then we need to add all outliers as a new cluster. Way to do, is iterator all points again, and check if point is not in the geneGroup, then add it into outlier cluster, we will get a outlier cluster that contain all outliers.

7, Return outlier array and geneGroup to end the function.

4.4 Result

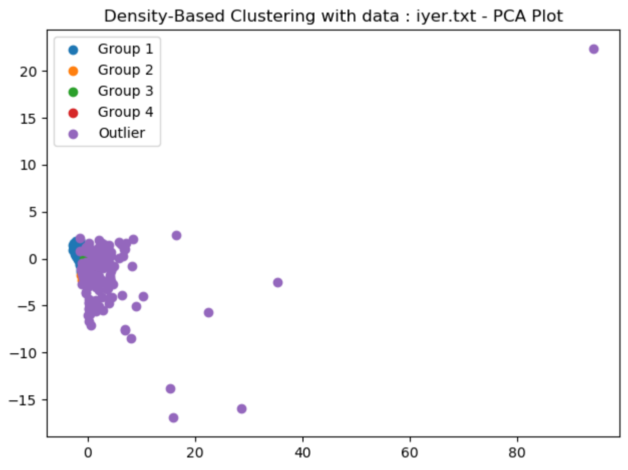
4.4.1 *Data file : cho.txt.* Input parameter : eps = 4, MinPts = 1



Jaccard Coefficient is : 0.203663430710938734

Rand Index is : 0.5232220999221455

4.4.2 *Data file : iyer.txt.* Input parameter : eps = 4, MinPts = 1



Jaccard Coefficient is : 0.283738059615605948

Rand Index is : 0.6507375911466615

4.5 Discussion

4.5.1 *Pros.*

- It can resistant to noise
- Can handle different shapes and size of clusters.

4.5.2 *Cons.*

- Cannot handle varying densities
- Sensitive to parameters, epsilon and minPts. (hard to determine the correct set of parameters)

5 GUASSIAN MIXURE MODEL

5.1 Introduction

Gaussian mixure model is a type of probabilistic clustering. Each cluster is mathematically represented by a parametric distribution. Here, we use the Gaussian distribution to determine if the data are more likely to be part of certain clusters. The parameter we take are $\pi, \mu, and\sigma$. π represent each clusters distribution base on the given data. μ represent the height or amplitude of the distribution. σ represent the increase or decrease spread of the cluster. Using the parameters, we find the maximum likelihood for each data point to be on specific cluster. By doing that, we will have to use Expectation-Maximization Algorithm.

5.2 Pseudo-code

5.3 Implementation

To break it down, we follow the formula for E-Step function:

Algorithm 4: GMM Algorithm

```
while Not convergence or not reach to # of iterations do
    P = E-Step(data,$\pi$, $\sigma$, $\mu$);
    ($\pi$, $\sigma$, $\mu$) = M-Step(P);
    likelihood = Max-Likelihood();
end
```

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

rik will return our probability matrix and each column representing the probability of the data point being into the clusters. After we found our first step, we will need to update our parameters $\pi, \sigma\mu$. Which will lead us to M-Step of the following equations:

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

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

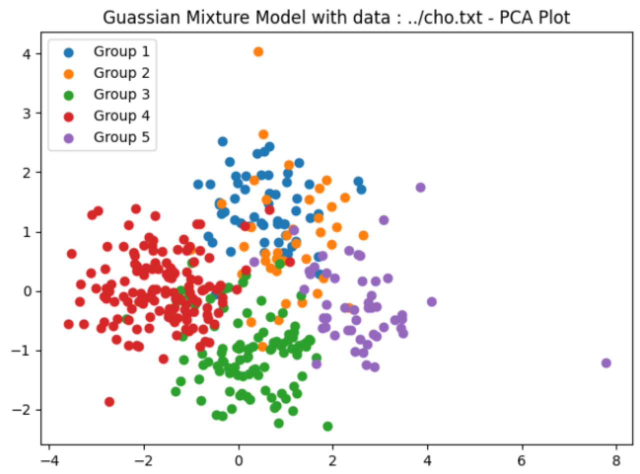
We then use this updated parameters to run i iterations. The threshold are use to stop the iteration because the maximum likelihood has not change as much as we wanted to. In order to find the maximum likelihood we use this equation:

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

Finally, we can use this rik matrix to determined which data point belongs to the different clusters. After grouping the clusters, we can easily plot the graph base with different cluster color.

5.4 Result

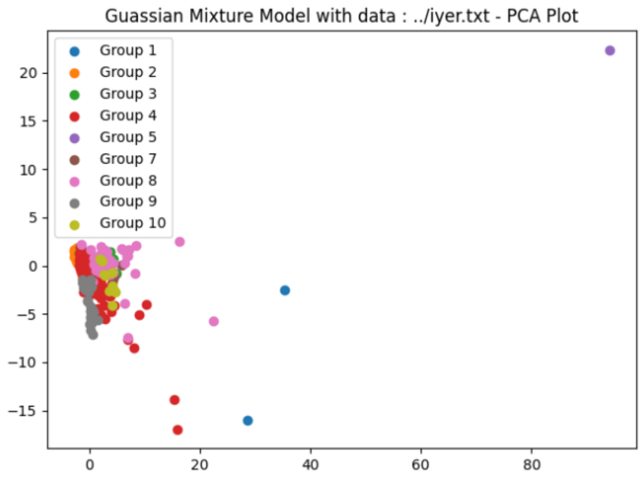
5.4.1 Data file : cho.txt. Input parameter : k = 5, iteration = 100, u = [1,50,100,200,300] pi = [0.2,0.2,0.2,0.2,0.2], sigma = (k * d * d)



Jaccard Coefficient is : 0.4110781404549951

Rand Index is : 0.8001959784155279

5.4.2 Data file : iyer.txt. Input parameter : k = 10, iteration = 100, u = [1,50,100,200,300,400,500,101,102,103], pi = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1] , sigma = (k * d * d)



Jaccard Coefficient is : 0.3565020837748111

Rand Index is : 0.7614192877372432

5.5 Discussion

5.5.1 Pros.

- Give probabilistic cluster assignments
- Have probabilistic interpretation
- Can handle clusters with varying sizes, variance etc.

5.5.2 *Cons.*

- Initialization matters. Parameters are very hard to setting.
- Choose appropriate distributions. This is important because the shape changes differently depends on the distribution initially.
- Overfitting issues.

6 K-WAY SPECTRAL CLUSTERING

6.1 Introduction

Spectral clustering is used when clusters are in a circular shape, like an oval. K-means does not help us clusters in this way, because k-mean look for errors and distance between the points and the center. To solve this, we can use Spectral clustering and the method is to cluster by using eigenvalues and eigenvectors. The direction eigenvectors allow us to determine the point should be cluster base on the different changes of the data points. There are three step to work on, building the Laplacian matrix L, decomposition to find the eigenvectors X and eigenvalue λ of the matrix L to build embedded space, and finally apply k-means to reduced nxd space to produce k clusters.

6.2 Code

Algorithm 5: GMM Algorithm

```
while Not convergence or not reach to # of iterations do
    L = Pre-processing();
    $(X, \lambda)$ = Decomposition(L);
    Cluster($\lambda$);
end
```

6.3 Implementation

Starting with the Laplacian matrix, we have to find the similarity matrix and the degree matrix. By subtracting the similarity matrix and degree matrix, we will able to get the Laplacian matrix. To create the similarity matrix, the follow equation:

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

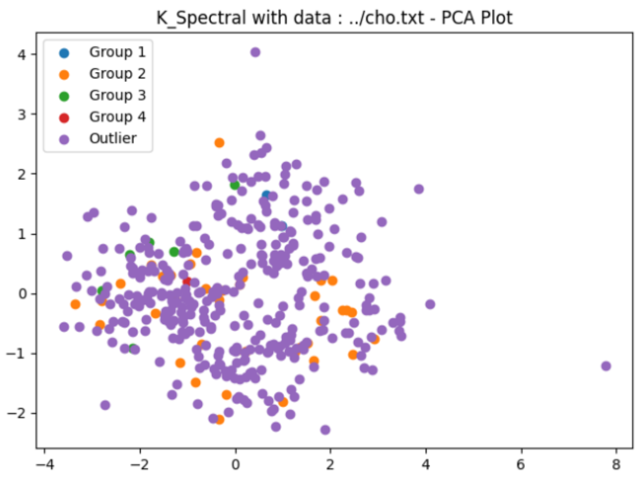
Then to find the degree matrix, we just add all the row and put the sum into the diagonal data point. Where the D matrix only have diagonals with the sum and rest are 0. Next we apply the equation:

$$L = D - W$$

We then take the Laplacian matrix and find the eigenvalues and eigenvectors. Then use the second value of eigenvalues and the corresponding eigenvectors to represent our new data. We then implement the k-mean on the new embedded space to get our clusters.

6.4 Result

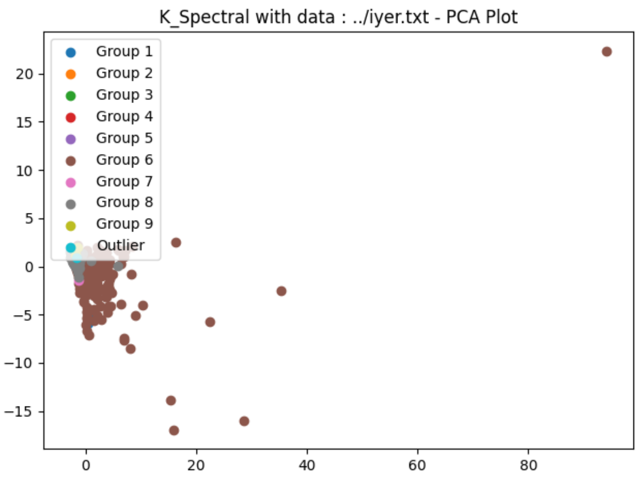
6.4.1 *Data file : cho.txt.* Input parameter : k = 5, iteration = 100, center = [1,2,3,4,5], sigma= 0.1



Jaccard Coefficient is : 0.21293997371111437

Rand Index is : 0.34896238825203363

6.4.2 *Data file : iyer.txt.* Input parameter : k = 10, iteration = 100, center = [1,2,3,4,5,6,7,8,9,10], sigma = 0.1



Jaccard Coefficient is : 0.28165996714585273

Rand Index is : 0.6449835197108748

6.5 Discussion

6.5.1 *Pros.*

- Can find spherical clusters and other variet shapes.
- Do not need actual dataset.

6.5.2 *Cons.*

- Choices of k cluster is hard to determine.

7 CLUSTER COMPARE

Data : cho.txt		
Clusters Name	Jaccard Coefficient	Rand Index
K-Mean	0.4095414873030291	0.7993100485919085
Hierarchical	0.22839497757358454	0.24027490670890495
DBSCAN	0.203663430710938734	0.5232220999221455
GMM	0.4110781404549951	0.8001959784155279
Spectral	0.21293997371111437	0.34896238825203363

Data : iyer.txt		
Clusters Name	Jaccard Coefficient	Rand Index
K-Mean	0.3468332185044284	0.7552686418071825
Hierarchical	0.1584680239258938	0.1941456625600006
DBSCAN	0.283738059615605948	0.6507375911466615
GMM	0.3565020837748111	0.7614192877372432
Spectral	0.28165996714585273	0.6449835197108748

- As the table and the result we show in upper report, we can see that Hierarchical has really bad performance for both data, the reason is, we can see the data set as PCA result, and it has a lot of noise point in the different corners of images, it affect Hierarchical cluster result very bad.
- K-Way spectral clustering and DBSCAN can detect the noise point, other three clusters cannot detect the noise point.
- Because parameter selection is difficult for DBSCAN and Spectral, so result if not very clearly and right.

REFERENCES

[1] [n. d.]. DBSCAN. https://en.wikipedia.org/wiki/DBSCAN

[2] [n. d.]. Hierarchical clustering. https://en.wikipedia.org/wiki/Hierarchical_clustering