Lab Session-IV

(Clustering: K-Means, Hierarchical)

Dr. JASMEET SINGH

ASSISTANT PROFESSOR

CSED, TIET

K-Means Clustering: Introduction

- The K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids
- It halts creating and optimizing clusters when either:
  - The centroids have stabilized — there is no change in their values because the clustering has been successful.
  - The defined number of iterations has been achieved.

K-Means Algorithm

Algorithm 2.1 k-means clustering algorithm.

Input: Data $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, the order k, MAX number of allowed iterations
Output: A partition $\mathcal{P} = \{\mathcal{C}_1, ..., \mathcal{C}_K\}$

1: $t = 0, \mathcal{P} = \emptyset$
2: Randomly initialize $\boldsymbol{\mu}_i, i = 1, ..., K$
3: **loop**
4: $t+ = 1$
5: Assignment Step: assign each sample $\mathbf{x}_j$ to the cluster with the nearest representative
6: $\mathcal{C}_i^{(t)} = \{\mathbf{x}_j : d(\mathbf{x}_j, \boldsymbol{\mu}_i) \leq d(\mathbf{x}_j, \boldsymbol{\mu}_h) \text{ for all } h = 1, \ldots, K\}$
7: Update Step: update the representatives
8: $\boldsymbol{\mu}_i^{(t+1)} = \frac{1}{|\mathcal{C}_i^{(t)}|} \sum_{\mathbf{x}_j \in \mathcal{C}_i} \mathbf{x}_j$
9: Update the partition with the modified clusters:
$\mathcal{P}^t = \{\mathcal{C}_1^{(t)}, ..., \mathcal{C}_K^{(t)}\}$
10: **if** $t \geq$ MAX OR $\mathcal{P}^t = \mathcal{P}^{t-1}$ **then**
11: **return** $\mathcal{P}^t$
12: **end if**
13: **end loop**

K-Means: Step-by-Step Implementation

Step 1. Randomly pick k data points as our initial Centroids.

Step 2. Find the distance (Euclidean distance or any other relevant distance measure) between each data points in our training set with the k centroids.

Step 3. Now assign each data point to the closest centroid according to the distance found.

Step 4. Update centroid location by taking the average of the points in each cluster group.

Step 5. Repeat the Steps 2 to 4 till our centroids don't change.

We can choose optimal value of K (Number of Clusters) using methods like the The Elbow method.

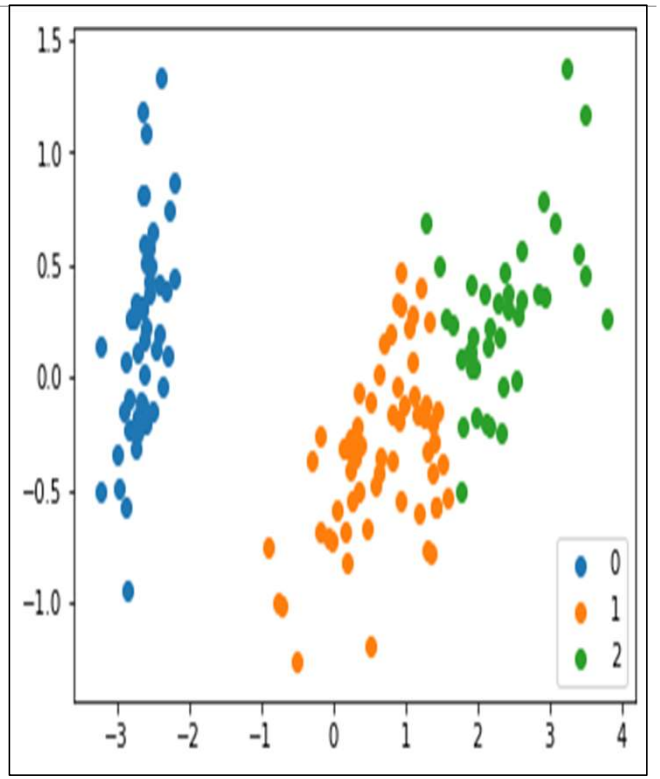
K-Mean Function Implementation

```
import numpy as np
from scipy.spatial.distance import cdist

#Function to implement steps given in previous section
def kmeans(x,k, no_of_iterations):
    idx = np.random.choice(len(x), k, replace=False)
    #Randomly choosing Centroids
    centroids = x[idx, :] #Step 1

#finding the distance between centroids and all the data points
    distances = cdist(x, centroids ,'euclidean') #Step 2

    #Centroid with the minimum Distance
    points = np.array([np.argmin(i) for i in distances]) #Step 3
```

```
#Repeating the above steps for a defined number of iterations
    #Step 4
    for epochs in range(no_of_iterations):
        centroids = []
        for idx in range(k):
 #Updating Centroids by taking mean of Cluster it belongs to
            temp_cent = x[points==idx].mean(axis=0)
            centroids.append(temp_cent)
        centroids = np.vstack(centroids) #Updated Centroids
        distances = cdist(x, centroids ,'euclidean')
        points = np.array([np.argmin(i) for i in distances])

    return points
```

Using K-Means Function

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
#Load Data
data = load_iris().data
pca = PCA(2)
#Transform the data
df = pca.fit_transform(data)
#Applying our function
label = kmeans(df,3,300)
#Visualize the results
u_labels = np.unique(label)
for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.legend()
plt.show()
```



Choosing Value of K- Method 1

- One way to choose the value of K is to find and plot the accuracy of the clustering output for each value of K
- If the labels of each clusters are known, we can compute accuracy, mean square error, Rand Index, Adjusted Rand Index, etc.
- If the labels are not known, we can compute Silhouette Coefficient.
- Code

```
from sklearn import metrics
MSE=[]
for K in range(1,4):
    label = kmeans(df,K,300)
    MSE.append(metrics.mean_squared_error(label,load_iris().target))
```

Choosing Value of K- Method 2

Elbow Method:

We iterate the values of k in a range and calculate the values of distortions for each value of k and calculate the distortion and inertia for each value of k in the given range.

1. **Distortion:** It is calculated as the average of the squared distances from the cluster centers of the respective clusters. Typically, the Euclidean distance metric is used.

2. **Inertia:** It is the sum of squared distances of samples to their closest cluster center.

Choosing Value of K- Method 2 (Contd….)

```
distortions = []
#Defining our function
def kmeans(x,k, no_of_iterations):
    idx = np.random.choice(len(x), k, replace=False)
    #Randomly choosing Centroids
    centroids = x[idx, :] #Step 1
   #finding the distance between centroids and all the data points
    distances = cdist(x, centroids ,'euclidean') #Step 2
    #Centroid with the minimum Distance
    points = np.array([np.argmin(i) for i in distances]) #Step 3
```

```
for epochs in range(no_of_iterations):
        centroids = []
        for idx in range(k):
          #Updating Centroids by taking mean of Cluster it belongs to
           temp_cent = x[points==idx].mean(axis=0)
           centroids.append(temp_cent)
        centroids = np.vstack(centroids) #Updated Centroids
        distances = cdist(x, centroids ,'euclidean')
        points = np.array([np.argmin(i) for i in distances])

distortions.append(sum(np.min(cdist(x,centroids,'euclidean'), axis=1))/x.shape[0])
    return points

for K in range(1,5):
    label = kmeans(df,K,300)
print (distortions)
```

K- means Inbuilt Function

We can also use inbuilt K- means algorithm

Syntax:

sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances ='deprecated', verbose=0, random_state=None, copy_x=True, n_jobs='deprecated', algorithm='auto')

Code: (On load_iris)

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=0).fit(df)

label=kmeans.labels_

print(metrics.mean_squared_error(label,load_iris().target))
```

Agglomerative Clustering-Inbuilt Function

Syntax:

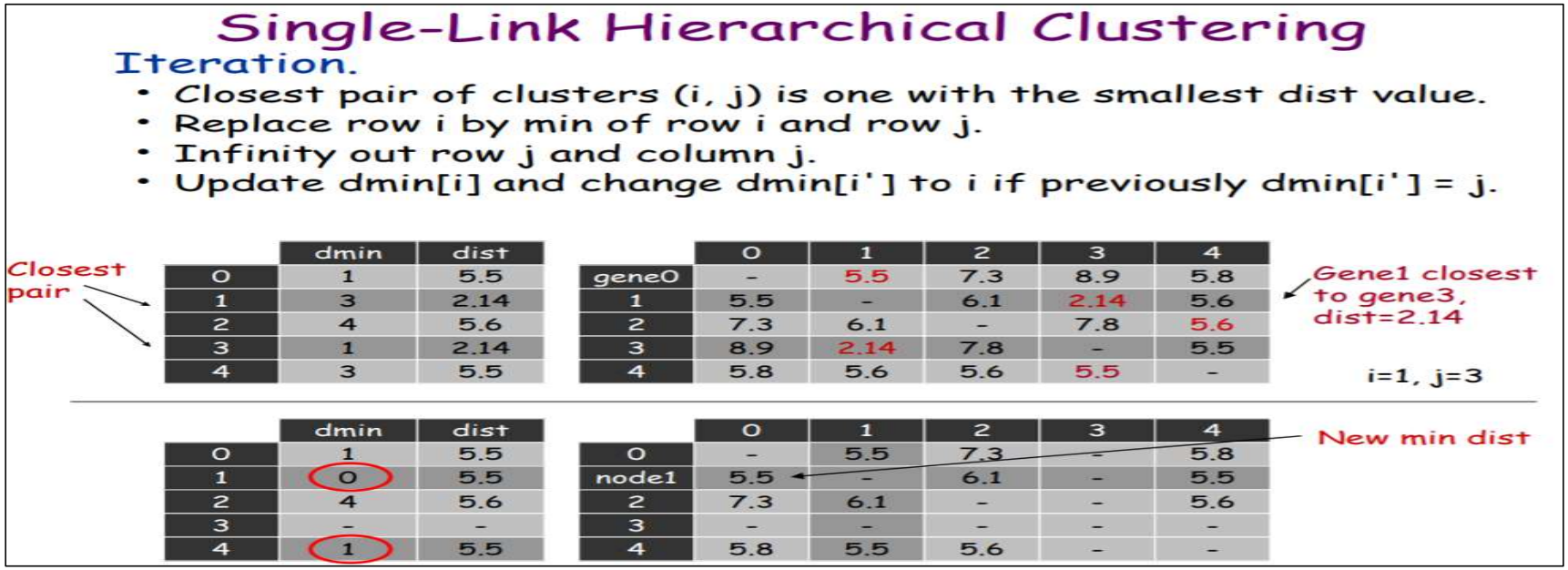
klearn.cluster.AgglomerativeClustering(n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False)

Code:

```
from sklearn.cluster import AgglomerativeClustering

clustering = AgglomerativeClustering(n_clusters=3).fit(df)

label=clustering.labels_

print(metrics.mean_squared_error(label,load_iris().target))
```

Coding- Agglomerative Clustering

Idea (Try Implementation):



	dmin	dist
0	1	5.5
1	3	2.14
2	4	5.6
3	1	2.14
4	3	5.5

	0	1	2	3	4
gene0	-	5.5	7.3	8.9	5.8
1	5.5	-	6.1	2.14	5.6
2	7.3	6.1	-	7.8	5.6
3	8.9	2.14	7.8	-	5.5
4	5.8	5.6	5.6	5.5	-

	dmin	dist
0	1	5.5
1	0	5.5
2	4	5.6
3	-	-
4	1	5.5

	0	1	2	3	4
0	-	5.5	7.3	-	5.8
node1	5.5	-	6.1	-	5.5
2	7.3	6.1	-	-	5.6
3	-	-	-	-	-
4	5.8	5.5	5.6	-	-