**VIVIAN JOSEPH**

(20BCE0777)

Machine Learning

**Lab DA-4**

Clustering Algorithms

**K-means Clustering:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cluster import KMeans

%matplotlib inline

import numpy as np

df = pd.read\_csv('Mall\_Customers.csv')

X = df.iloc[:, [3,4]].values

Make a distance function and a function to calculate WCSS

def distance(point1,point2):

    return np.sqrt(np.sum((point1-point2)\*\*2))

def calculate\_wcss(centroids, clusters, data):

    wcss = 0

    for i in range(centroids.shape[0]):

        cluster\_points = data[clusters == i]

        centroid = centroids[i]

        distances = distance(cluster\_points,centroid)\*\*2

        wcss += np.sum(distances)

    return wcss

Make a function to calculate K Means

def kmeans(k):

    n=X.shape[0]

    # centroids = np.random.randn(k, 2)

    centroids = X[:k, :]

    # print(centroids)

    clusters=np.zeros(n)

    epochs=100

    for e in range(epochs):

        # print(centroids)

        for i in range(n):

            distances=[distance(X[i],centroid) for centroid in centroids]

            clusters[i]=np.argmin(distances)

        new\_centroids=np.zeros((k,2))

        for i in range(k):

            points=[X[j] for j in range(n) if clusters[j]==i]

            # print(len(points))

            if len(points)>0:

                new\_centroids[i]=np.mean(points,axis=0)

        if np.all(centroids==new\_centroids):

            # print(e,"epochs")

            break

        else:

            centroids=new\_centroids

    return clusters, centroids

Check WCSS for a range of values

wcss=[]

for k in range(1,13):

    clusters, centroids=kmeans(k)

    print(k,end=": ")

    for i in range(k):

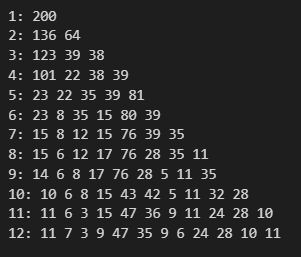
            points=[X[j] for j in range(200) if clusters[j]==i]

            print(len(points),end=" ")

    print()

    wcss.append(calculate\_wcss(centroids,clusters,X))

Checking what is the distribution of points in each cluster for each k



Plot the elbow graph

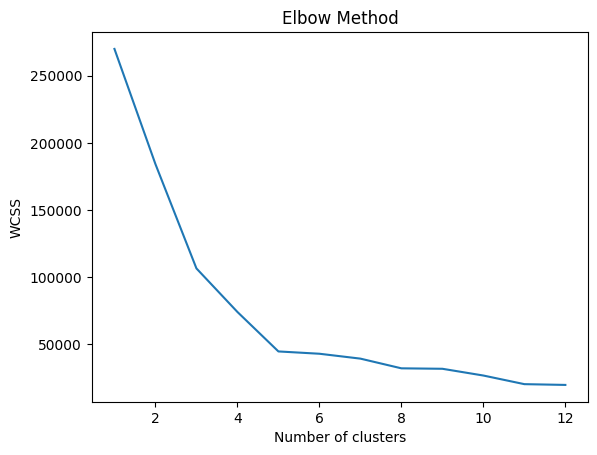
plt.plot(range(1, 13), wcss)

plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

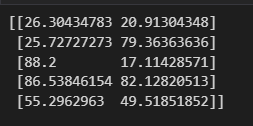




Calculate kmeans clusters for **k=5**

clusters, centroids=kmeans(5)

print(centroids)



print("WCSS:",calculate\_wcss(centroids,clusters,X))





Plot the clusters and respective points

colors=['orange','g','b','c','r']

for i in range(5):

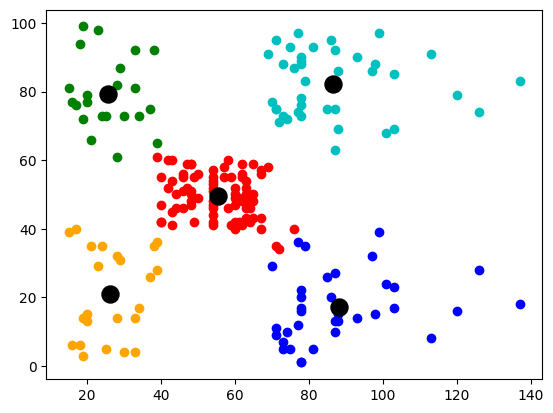
    for j in range(200):

        if clusters[j]==i:

            plt.scatter(X[j,0], X[j,1], c=colors[i], cmap='rainbow')

plt.scatter(centroids[:,0], centroids[:, 1], s=150, c='k')

plt.show()



**K-modes Clustering:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cluster import KMeans

%matplotlib inline

import numpy as np

To get classes for k-modes, convert numerical data to categorial data

df = pd.read\_csv('Mall\_Customers.csv')

age\_bins = [0, 15, 30, 45,60,80, 100]

age\_labels = [1, 2, 3,4,5,6]

df['Age Class'] = pd.cut(df['Age'], bins=age\_bins, labels=age\_labels)

income\_bins = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,200]

income\_labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11]

df['Income Class'] = pd.cut(df['Annual Income (k$)'], bins=income\_bins, labels=income\_labels)

X = df.iloc[:, [5,6]].values

Make a function to calculate k-mode clusters for any k

def kmode(k):

    epochs=100

    rng = np.random.default\_rng()

    centroids = rng.choice(X, size=k, replace=False)

    n\_samples, n\_features = X.shape

    distances = np.zeros((n\_samples, k))

    for e in range(epochs):

        for i in range(n\_samples):

            data=X[i]

            for j in range(k):

                cen=centroids[j]

                dis=0

                for d in range(n\_features):

                    if data[d]!=cen[d]:

                        dis+=1

                distances[i, j] = dis

        labels = np.argmin(distances, axis=1)

        for j in range(k):

            cluster\_j = X[labels == j]

            centroids[j]  = np.apply\_along\_axis(lambda x: np.bincount(x).argmax(), axis=0, arr=cluster\_j)

    return labels, centroids

We got **K=5,** calculate the clusters for these

clusters, centroids=kmode(5)

colors=['orange','g','b','c','r']

for i in range(5):

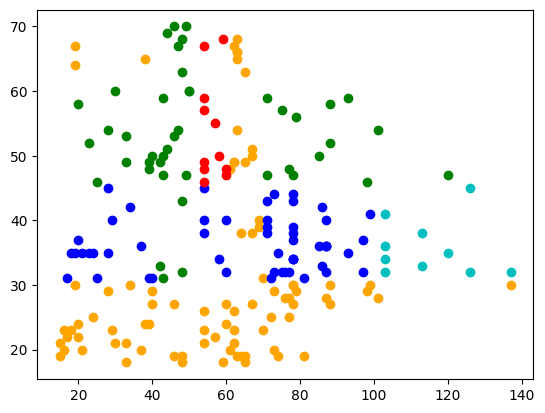
    for j in range(200):

        if clusters[j]==i:

            plt.scatter(df.iloc[j, 3], df.iloc[j, 2], c=colors[i], cmap='rainbow')

# plt.scatter(centroids[:,0], centroids[:, 1], s=150, c='k')

plt.show()



This data aren’t look as impressive on a graph as the it is calculated on mode and not Euclidian distances

**Hierarchial Clustering:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cluster import KMeans

%matplotlib inline

import numpy as np

df = pd.read\_csv('Mall\_Customers.csv')

X = df.iloc[:, [3,4]].values

Function to calculate Euclidian distance

def euclidean\_distance(x1, x2):

    return np.sqrt(np.sum((x1 - x2)\*\*2))

Initialize distance vector

dist\_matrix = np.zeros((len(X), len(X)))

for i in range(len(X)):

    for j in range(i+1, len(X)):

        dist\_matrix[i][j] = euclidean\_distance(X.iloc[i, :], X.iloc[j, :])

        dist\_matrix[j][i] = dist\_matrix[i][j]

Finf=d the closest clusters

def find\_closest\_clusters(dist\_matrix):

    min\_dist = np.inf

    closest\_clusters = ()

    for i in range(len(dist\_matrix)):

        for j in range(i+1, len(dist\_matrix)):

            if dist\_matrix[i][j] < min\_dist:

                min\_dist = dist\_matrix[i][j]

                closest\_clusters = (i, j)

    return closest\_clusters

Merge the found clusters

def merge\_clusters(data, dist\_matrix, cluster1, cluster2):

    new\_cluster = np.mean(np.vstack((data.iloc[cluster1, :], data.iloc[cluster2, :])), axis=0)

    new\_dist\_row = np.zeros((1, len(dist\_matrix)))

    for i in range(len(dist\_matrix)):

        if i != cluster1 and i != cluster2:

            new\_dist\_row[0][i] = np.mean([dist\_matrix[cluster1][i], dist\_matrix[cluster2][i]])

    new\_dist\_row[0][cluster1] = np.inf

    new\_dist\_row[0][cluster2] = np.inf

    new\_dist\_col = np.concatenate((dist\_matrix, np.zeros((1, len(dist\_matrix)))), axis=0)

    new\_dist\_col = np.concatenate((new\_dist\_col, np.zeros((len(dist\_matrix)+1, 1))), axis=1)

    new\_dist\_col[-1][:-1] = new\_dist\_row

    new\_dist\_col[:-1, -1] = new\_dist\_row.T

    new\_dist\_col[-1][-1] = np.inf

    new\_data = data.drop([cluster1, cluster2], axis=0)

    new\_data.loc[len(new\_data)] = new\_cluster

    return new\_data, new\_dist\_col

make a list of all clusters at every given point

clusters = list(range(len(X)))

while len(clusters) > 1:

    closest\_clusters = find\_closest\_clusters(dist\_matrix)

    cluster1, cluster2 = closest\_clusters[0], closest\_clusters[1]

    X, dist\_matrix = merge\_clusters(X, dist\_matrix, cluster1, cluster2)

    clusters.remove(cluster1)

    clusters.remove(cluster2)

    clusters.append(len(X)-1)

Let’s look at clusters formed at any given point

clusters = agnes(X, 6)

plt.figure(figsize=(10, 6))

colors = ['r', 'g', 'b', 'y', 'm', 'c']

for i, cluster in enumerate(clusters):

    plt.scatter(X[cluster, 0], X[cluster, 1], color=colors[i%6], label=f'Cluster {i+1}')

plt.legend()

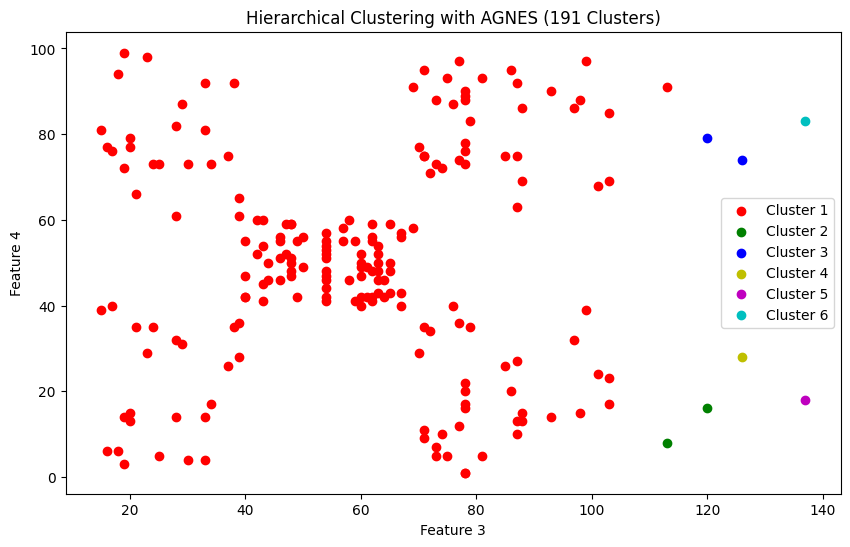
plt.title(f'Hierarchical Clustering with AGNES ({k} Clusters)')

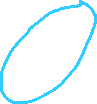
plt.xlabel('Feature 3')

plt.ylabel('Feature 4')

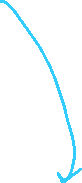
plt.show()

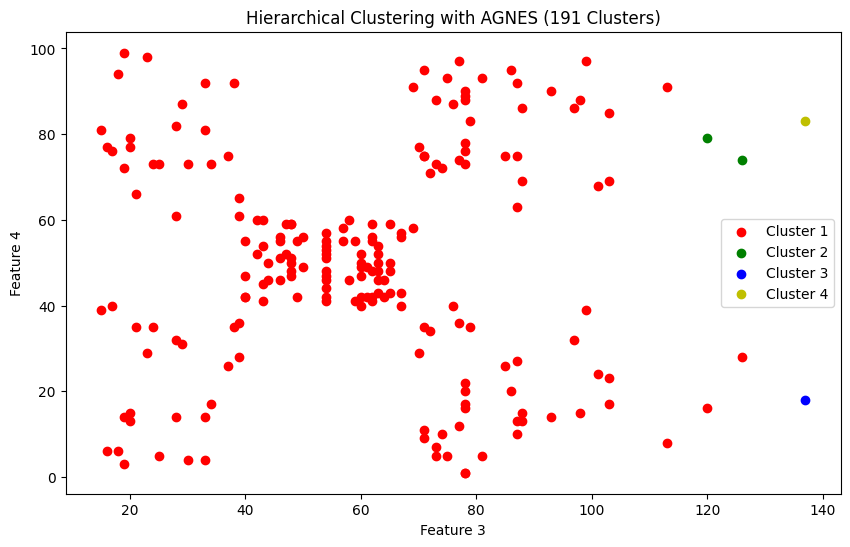
**For 6 clusters formed out of total data points**





Cluster 2 and cluster 4 joined with cluster 1







**For 4 clusters formed out of total data points**