|  |  |
| --- | --- |
| Name: | Prerna Sunil Jadhav |
| Sap  Id: | 60004220127 |
| Class: | T. Y. B.Tech (Computer Engineering) |
| Course: | Data Mining and Warehouse Laboratory |
| Course Code: | DJ19CEL501 |
| Experiment  No.: | 05 |

**AIM:** Implementation of K Means and Hierarchical Clustering algorithm

**PART A (Using Inbuilt function)**

K-Means

CODE:

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

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

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

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters') plt.ylabel('WCSS')

plt.show()

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42)

y\_kmeans = kmeans.fit\_predict(X) print(y\_kmeans)

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0],

kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()

A white rectangular table with black text

Description automatically generatedOUTPUT:

A chart of clusters of customers

Description automatically generatedA graph with a line

Description automatically generated

Hierarchical Clustering

CODE:

# Importing the libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

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

X = dataset.iloc[:, [3, 4]].values len(X)

# Using the dendrogram to find the optimal number of clusters import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward')) plt.title('Dendrogram')

plt.xlabel('Customers')

plt.ylabel('Euclidean distances') plt.show()

# Training the Hierarchical Clustering model on the dataset from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters = 5, affinity = 'euclidean', linkage = 'ward')

y\_hc = hc.fit\_predict(X)

print(y\_hc)

# Visualising the clusters

plt.scatter(X[y\_hc == 0, 0], X[y\_hc==0,1],s=100,c ='red', label = 'Cluster 1')

plt.scatter(X[y\_hc == 1, 0], X[y\_hc==1,1],s=100,c =blue, label = 'Cluster 2')

plt.scatter(X[y\_hc == 2, 0], X[y\_hc==2,1],s=100,c ='green', label = 'Cluster 3')

plt.scatter(X[y\_hc == 3, 0], X[y\_hc==3,1],s=100,c =cyan, label = 'Cluster 4')

plt.scatter(X[y\_hc == 4, 0], X[y\_hc==4,1],s=100,c ='magenta', label = 'Cluster 5')

plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()

A number grid with black and white numbers

Description automatically generated with medium confidenceA chart with colorful dots

Description automatically generatedA diagram of a company

Description automatically generated with medium confidenceOUTPUT:

**PART B**

K-Means

CODE:

import pandas as pd

data = pd.read\_csv("driver-data.csv", index\_col="id") data.head()

from sklearn.cluster import KMeans kmeans = KMeans(n\_clusters=4)

kmeans.fit(data)

kmeans.cluster\_centers\_ kmeans.labels\_

import numpy as np

unique, counts = np.unique(kmeans.labels\_, return\_counts=True) dict\_data = dict(zip(unique, counts))

dict\_data

import seaborn as sns

data["cluster"] = kmeans.labels\_ sns.pairplot(data)

kmeans.inertia\_ kmeans.score

data

from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt from matplotlib import style

import pandas as pd

style.use('ggplot') class K\_Means:

def init (self, k =3, tolerance = 0.0001, max\_iterations = 500):

self.k = k

self.tolerance = tolerance

self.max\_iterations = max\_iterations def fit(self, data):

self.centroids = {}

#initialize the centroids, the first 'k' elements in the dataset will be our initial centroids

for i in range(self.k):

self.centroids[i] = data[i]

#begin iterations

for i in range(self.max\_iterations): self.classes = {}

for i in range(self.k):

self.classes[i] = []

#find the distance between the point and cluster; choose the nearest centroid

for features in data:

distances = [np.linalg.norm(features - self.centroids[centroid]) for centroid in self.centroids]

classification = distances.index(min(distances))

self.classes[classification].append(features)

previous = dict(self.centroids)

#average the cluster datapoints to re-calculate

for classification in self.classes:

self.centroids[classification] =

np.average(self.classes[classification], axis = 0) isOptimal = True

for centroid in self.centroids:

original\_centroid = previous[centroid] curr = self.centroids[centroid]

if np.sum((curr -

original\_centroid)/original\_centroid \* 100.0) > self.tolerance:

isOptimal = False

#break out of the main loop if the results are

optimal, ie. the centroids don't change their positions much(more than our tolerance)

if isOptimal:

break

def pred(self, data):

distances = [np.linalg.norm(data -

self.centroids[centroid]) for centroid in self.centroids] classification = distances.index(min(distances)) return classification

def main():

df = pd.read\_csv("Mall\_Customers.csv") df = X = df.iloc[:, [3, 4]]

dataset = df.astype(float).values.tolist()

X = df.values #returns a numpy array km = K\_Means(5)

km.fit(X)

# Plotting starts here

colors = 10\*["r", "g", "c", "b", "k"]

for centroid in km.centroids:

plt.scatter(km.centroids[centroid][0], km.centroids[centroid][1], s = 130, marker = "x")

for classification in km.classes:

color = colors[classification]

for features in km.classes[classification]:

plt.scatter(features[0], features[1], color =

color,s = 30)

plt.show()

if name == " main ":

main()

A screenshot of a graph

Description automatically generatedA screenshot of a graph

Description automatically generatedA graph with different colored dots

Description automatically generatedOUTPUT:

HIERARCHICAL CLUSTERING

CODE:

# Importing the libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

import seaborn as sns

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv') X = dataset.iloc[:, [3, 4]].values

X

new\_data = dataset

new\_data = new\_data.drop('CustomerID', axis=1) new\_data

sns.pairplot(dataset)

from sklearn.preprocessing import LabelEncoder

new\_data = new\_data.apply(LabelEncoder().fit\_transform) X = new\_data.to\_numpy()

class Distance\_computation\_grid(object):

'''class to enable the Computation of distance matrix'''

def init (self): pass

def compute\_distance(self,samples):

'''Creates a matrix of distances between individual samples and clusters attained at a particular step'''

Distance\_mat = np.zeros((len(samples),len(samples))) for i in range(Distance\_mat.shape[0]):

for j in range(Distance\_mat.shape[0]): if i!=j:

Distance\_mat[i,j] =

float(self.distance\_calculate(samples[i],samples[j]))

else:

Distance\_mat[i,j] = 10\*\*4 return Distance\_mat

def distance\_calculate(self,sample1,sample2):

dist = []

for i in range(len(sample1)):

for j in range(len(sample2)): try:

dist.append(np.linalg.norm(np.array(sample1[i])-np.array(sample2[j])))

except:

dist.append(self.intersampledist(sample1[i],sample2[j])) return min(dist)

def intersampledist(self,s1,s2):

if str(type(s2[0]))!='<class \'list\'>': s2=[s2]

if str(type(s1[0]))!='<class \'list\'>':

s1=[s1]

m = len(s1) n = len(s2) dist = []

if n>=m:

for i in range(n):

for j in range(m):

if (len(s2[i])>=len(s1[j])) and

str(type(s2[i][0])!='<class \'list\'>'):

dist.append(self.interclusterdist(s2[i],s1[j]))

else:

dist.append(np.linalg.norm(np.array(s2[i])-np.array(s1[j]))) else:

for i in range(m):

for j in range(n):

if (len(s1[i])>=len(s2[j])) and

str(type(s1[i][0])!='<class \'list\'>'):

dist.append(self.interclusterdist(s1[i],s2[j]))

else:

dist.append(np.linalg.norm(np.array(s1[i])-np.array(s2[j]))) return min(dist)

def interclusterdist(self,cl,sample): if sample[0]!='<class \'list\'>':

sample = [sample] dist = []

for i in range(len(cl)):

for j in range(len(sample)):

dist.append(np.linalg.norm(np.array(cl[i])-np.array(sample[j]))) return min(dist)

progression = [[i] for i in range(X.shape[0])]

samples = [[list(X[i])] for i in range(X.shape[0])][:10] m = len(samples)

distcal = Distance\_computation\_grid() while m>2:

print('Sample size before clustering :- ',m)

Distance\_mat = distcal.compute\_distance(samples) sample\_ind\_needed =

np.where(Distance\_mat==Distance\_mat.min())[0]

value\_to\_add = samples.pop(sample\_ind\_needed[1]) samples[sample\_ind\_needed[0]].append(value\_to\_add)

print('Cluster Node 1

:-',progression[sample\_ind\_needed[0]]) print('Cluster Node 2

:-',progression[sample\_ind\_needed[1]])

progression[sample\_ind\_needed[0]].append(progression[sample\_ind\_ne eded[1]])

progression[sample\_ind\_needed[0]] = [progression[sample\_ind\_needed[0]]]

v = progression.pop(sample\_ind\_needed[1]) m = len(samples)

print('Progression(Current Sample) :-',progression) print('Cluster attained

:-',progression[sample\_ind\_needed[0]])

print('Sample size after clustering :-',m) print('\n')

from scipy.cluster.hierarchy import dendrogram, linkage from matplotlib import pyplot as plt

Z = linkage(X, 'single')

fig = plt.figure(figsize=(8, 8)) plt.title('Dendrogram')

dn = dendrogram(Z)

plt.scatter(X[:,2], X[:,3], cmap="rainbow")

from sklearn.cluster import AgglomerativeClustering aggclus = AgglomerativeClustering().fit(X)

aggclus.labels\_

A number in a line

Description automatically generated with medium confidenceA screenshot of a table

Description automatically generatedOUTPUT:

A screenshot of a graph

Description automatically generated

A screenshot of a computer

Description automatically generatedA screen shot of a graph

Description automatically generatedA graph of different colored columns

Description automatically generated

**PART C**

CODE:

from sklearn import datasets, preprocessing

from sklearn.preprocessing import LabelEncoder from sklearn.cluster import KMeans

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

df = df.apply(LabelEncoder().fit\_transform)

scaler = preprocessing.StandardScaler() scaled\_df = scaler.fit\_transform(df)

pd.DataFrame(scaled\_df).describe() clusters = range(1, 11)

sse=[]

for cluster in clusters:

model = KMeans(n\_clusters=cluster, init='k-means++', max\_iter=300, tol=0.0001, verbose=0,random\_state=0)

model.fit(scaled\_df)

sse.append(model.inertia\_)

sse\_df = pd.DataFrame(np.column\_stack((clusters, sse)), columns=['cluster', 'SSE'])

fig, ax = plt.subplots(figsize=(13, 5))

ax.plot(sse\_df['cluster'], sse\_df['SSE'], marker='o') ax.set\_xlabel('Number of clusters')

OUTPUT:

