**National Institute Of Technology, Hamirpur**

**Department Of Computer Science and Engineering**



**DATA MINING LAB ASSIGNMENT-6**

|  |  |
| --- | --- |
| Subject Name: **Data Mining Lab** | Subject Code: **CSD-426** |
| Student Name and Roll No:  **Nishant Singh Hada (17MI524)** | Semester: **VIII** |
| Submitted To: **Dr. Vijay Kumar Chahar** | Date of Submission: **31 March 2021** |
| **Faculty e-Signature** | |

**Problem 1 - Apply k-means clustering on IRIS Dataset**

1. **Code -** Reading the csv data

from copy import deepcopy

import numpy as np

import pandas as pd

import random

data = pd.read\_csv('iris.csv')

c1 = data['slength'].values

c2 = data['swidth'].values

c3 = data['plength'].values

c4 = data['pwidth'].values

X = np.array(list(zip(c1,c2,c3,c4)))

1. **Code –** Setting k = 3 & initializing cluster centers

k=3

c1 = [X[0][0],X[1][0],X[2][0]]

c2 = [X[0][1],X[1][1],X[2][1]]

c3 = [X[0][2],X[1][2],X[2][2]]

c4 = [X[0][3],X[1][3],X[2][3]]

c = np.array(list(zip(c1,c2,c3,c4)), dtype=np.float32)

1. **Code –** 
   1. dist = function to calculate distance between points
   2. c\_old = Stores the centroid values, as they are updated, initialiased to zeros
   3. clusters = Stores the centroid nearest to the point
   4. error = Stores the error at this stage, iteration runs till error becomes zero

def dist(a, b, ax=1):

return np.linalg.norm(a - b, axis=ax)

c\_old = np.zeros(c.shape)

clusters = np.zeros(len(X))

error = dist(c,c\_old,None)

1. **Code –**

while error!=0:

# Assigning each point to its nearest cluster

for i in range(len(X)):

distances = dist(X[i],c)

cluster = np.argmin(distances)

clusters[i] = cluster

# Old centroid values stored in c\_old

c\_old = deepcopy(c)

# Finding the new mean of each cluster

for i in range(k):

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

c[i] = np.mean(points, axis=0)

error = dist(c, c\_old, None)

1. **Output –**

print(c)

print(clusters)

print(error)

***Initial Cluster Centers***

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]]

***Final Cluster Centers***

[[6.853846 3.0769231 5.7153845 2.0538461 ]

[5.8836064 2.7409837 4.3885245 1.4344262 ]

[5.006 3.428 1.462 0.246 ]]

***Clusters***

[2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.

2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.

2. 2. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.

0. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0.

0. 0. 1. 0. 0. 1.]

***Error***

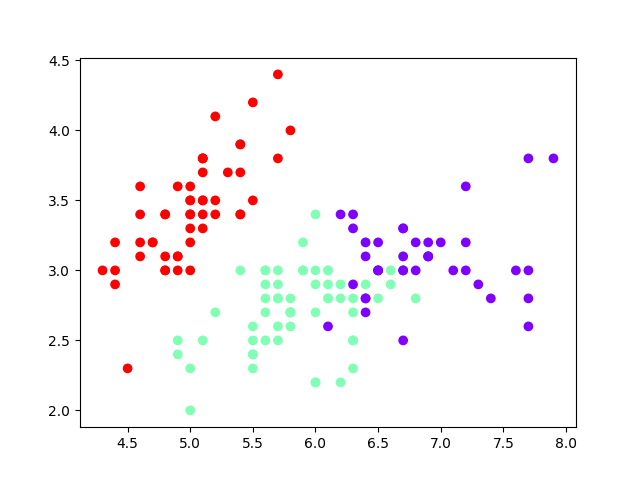
0.0

1. **Output –**

fig = plt.figure()

plt.scatter(X[:, 0], X[:, 1], c=clusters, cmap='rainbow')

plt.show()



**Problem 2 - Apply k-medoids clustering on IRIS Dataset**

1. **Code -** Reading the csv data

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from numpy.random import choice

from numpy.random import seed

data = pd.read\_csv('iris.csv')

c1 = data['slength'].values

c2 = data['swidth'].values

c3 = data['plength'].values

c4 = data['pwidth'].values

datapoints = np.array(list(zip(c1,c2,c3,c4)))

1. **Code –** Initializing Medoids

def init\_medoids(X, k):

seed(1)

samples = choice(len(X), size=k, replace=False)

return X[samples, :]

medoids\_initial = init\_medoids(datapoints, 3)

print(medoids\_initial)

***Initial Medoids***

[[5.8 4. 1.2 0.2]

[5.1 2.5 3. 1.1]

[6.6 3. 4.4 1.4]]

1. **Code –** Defining Functions

def distance(X, medoids):

m = len(X)

medoids\_shape = medoids.shape

if len(medoids\_shape) == 1:

medoids = medoids.reshape((1,len(medoids)))

k = len(medoids)

S = np.empty((m, k))

for i in range(m):

d\_i = np.linalg.norm(X[i, :] - medoids, ord=2, axis=1)

S[i, :] = d\_i\*\*2

return S

S = distance(datapoints, medoids\_initial)

def assign\_labels(S):

return np.argmin(S, axis=1)

labels = assign\_labels(S)

def has\_converged(old\_medoids, medoids):

return set([tuple(x) for x in old\_medoids]) == set([tuple(x) for x in medoids])

1. **Code –** Updating Medoids

def update\_medoids(X, medoids):

S = distance(datapoints, medoids)

labels = assign\_labels(S)

out\_medoids = medoids

for i in set(labels):

avg\_dissimilarity = np.sum(distance(datapoints, medoids[i]))

cluster\_points = datapoints[labels == i]

for datap in cluster\_points:

new\_medoid = datap

new\_dissimilarity= np.sum(distance(datapoints, datap))

if new\_dissimilarity < avg\_dissimilarity :

avg\_dissimilarity = new\_dissimilarity

out\_medoids[i] = datap

return out\_medoids

def kmedoids(X, k, starting\_medoids=None, max\_steps=np.inf):

if starting\_medoids is None:

medoids = init\_medoids(X, k)

else:

medoids = starting\_medoids

converged = False

labels = np.zeros(len(X))

i = 1

while (not converged) and (i <= max\_steps):

old\_medoids = medoids.copy()

S = distance(X, medoids)

labels = assign\_labels(S)

medoids = update\_medoids(X, medoids)

converged = has\_converged(old\_medoids, medoids)

i += 1

return (medoids,labels)

results = kmedoids(datapoints, 3)

final\_medoids = results[0]

data['clusters'] = results[1]

1. **Output -**

print(final\_medoids)

print(results[1])

***Final Medoids***

[[5.1 3.8 1.9 0.4]

[5.6 2.9 3.6 1.3]

[5.8 2.7 3.9 1.2]]

***Clusters***

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 1 2 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2

2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

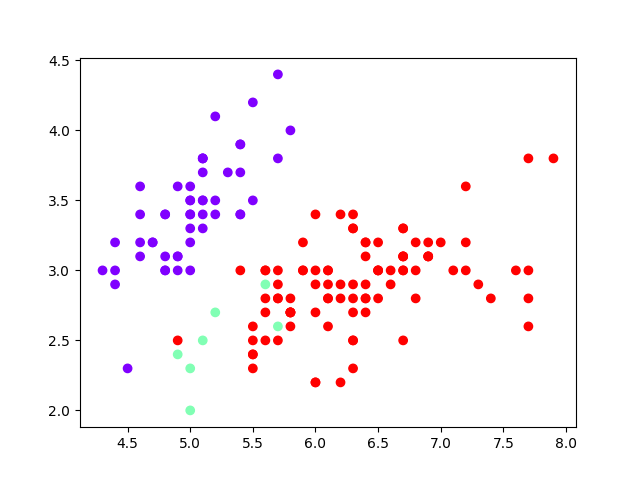
2 2]

1. **Output –**

fig = plt.figure()

plt.scatter(datapoints[:, 0], datapoints[:, 1], c=data['clusters'], cmap='rainbow')

plt.show()

****