داده کاوی

تمرین سری چهارم

سوال اول

در ابتدا کتابخانه های مورد نیاز را import میکنیم

```
import random
import pandas as pd
import math
import matplotlib.pyplot as plt
```

حال داده را میخوانیم و به فرم لیستی ازتاپل ها در می اوریم تا به این صورت که عضو اول هر تاپل x داده و عضو دوم y داده میباشد

```
#function below is for initializing random centres from points and saving data in array
dataArray = []
for i in range(len(data)):
    dataArray.append((data.iloc[i][0],data.iloc[i][1]))
def initialize_centres(data,clusterNumber):
    centres = random.choices(data,k = clusterNumber)
    return centres
```

تابع زیر نیز برای مقدار دهی اولیه به مرکز ها به صورت رندوم میباشد

```
def initialize_centres(data,clusterNumber):
    centres = random.choices(data,k = clusterNumber)
    return centres
```

تابع زیر برای محاسبه فاصله دو نقطه میباشد

```
#function below is for calculating distance between two points
def distance(dot1,dot2):
    return math.sqrt((dot1[0]-dot2[0])**2+(dot1[1]-dot2[1])**2)
```

تابع findClosestCentres با گرفتن داده و مراکز خوشه ها را با اعضاش برمیگرداند که مقدار برگردانده شده به شکل زیر میباشد

```
clusters form
            [[(),(),()],
             [(),()],
             [(),()],
             [(),(),(),()]]
"""
```

هر کلاستر یک لیست میباشد که شامل تعدادی تاپل میباشد که نشان گر نقطه متعلق به آن خوشه میباشد

چون مقدار اولیه این کلاستر ها شامل مراکز میباشد و در محاسبه دوباره مراکز را در خوشه قرار میدهیم عضو اول هر خوشه که مرکز آن خوشه است را حذف میکنیم

```
def findClosestCentres(data,centres):
    clusters = []
    #each cluster at first just have centers in it
    for i in range(len(centres)):
        clusters.append([centres[i],])
    for i in range(len(data)):
        #at first we assume minimum distance is with first centre and we will update it later
        minDist = distance(data[i],centres[0])
        minimum = 0
        for j in range(len(centres)):
            dist = distance(data[i], centres[j])
            if dist < minDist:
                minDist = dist
                minimum = j
        clusters[minimum].append(data[i])
    for i in range(len(centres)):
        clusters[i].pop(0)
    return clusters
```

برای محاسبه مرکز با استفاده از تابع computeMeans با میانگین گیری از نقاط هر خوشه مرکز را محاسبه میکنیم

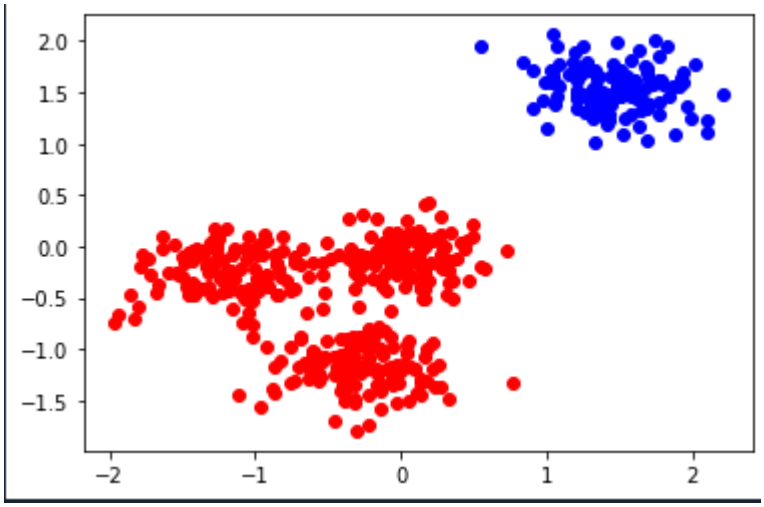
```
#function to compute centres
def computeMeans(data,clusters,clusterNumber):
    centres = []
    for i in range(clusterNumber):
        xSum = 0
        ySum = 0
        for j in range(len(clusters[i])):
            xSum += clusters[i][j][0]
            ySum += clusters[i][j][1]
        centres.append((xSum/len(clusters[i]),(ySum/len(clusters[i]))))
    return centres
```

و در نهایت الگوریتم kmeans که به ازای k های مختلف جواب میدهد و حلقه ای که ۱۵ بار اجرا میشود

```
#function below is for for main algorithm
def k_means(data,clusterNumber):
    centres = initialize_centres(data, clusterNumber)
    for i in range(15):
        clusters = findClosestCentres(data, centres)
        centres = computeMeans(data, clusters, clusterNumber)
    return clusters
```

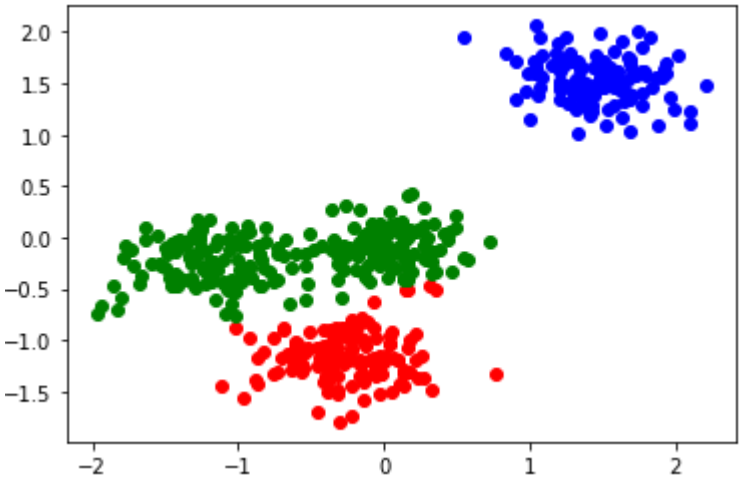
محاسبه و نمایش الگوریتم با دو خوشه

```
#computing kmeans on 2 clusters
k2means = k_means(dataArray,2)
#plotting clusters
x1_k2means = []
y1_k2means = []
x2_k2means = []
y2_k2means = []
for i in range(len(k2means[0])):
    x1_k2means.append(k2means[0][i][0])
    y1_k2means.append(k2means[0][i][1])
for i in range(len(k2means[1])):
    x2_k2means.append(k2means[1][i][0])
    y2_k2means.append(k2means[1][i][1])
plt.scatter(x1_k2means,y1_k2means,color = 'blue')
plt.scatter(x2_k2means,y2_k2means,color = 'red')
plt.show()
```



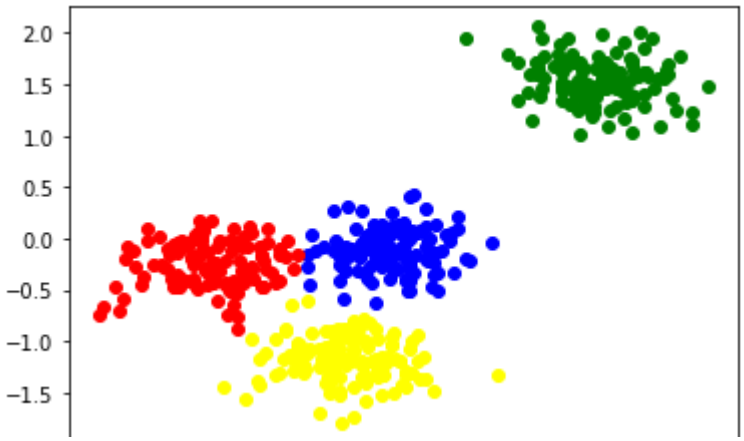
محاسبه و نمایش الگوریتم با سه خوشه

```
#computing kmeans on 3 clusters
k3means = k_means(dataArray,3)
#plotting clusters
x1_k3means = []
y1_k3means = []
x2_k3means = []
y2_k3means = []
x3_k3means = []
y3_k3means = []
for i in range(len(k3means[0])):
    x1_k3means.append(k3means[0][i][0])
    y1_k3means.append(k3means[0][i][1])
for i in range(len(k3means[1])):
    x2_k3means.append(k3means[1][i][0])
    y2_k3means.append(k3means[1][i][1])
for i in range(len(k3means[2])):
    x3_k3means.append(k3means[2][i][0])
    y3_k3means.append(k3means[2][i][1])
plt.scatter(x1_k3means,y1_k3means,color = 'blue')
plt.scatter(x2_k3means,y2_k3means,color = 'red')
plt.scatter(x3_k3means,y3_k3means,color = 'green')
plt.show()
```



محاسبه و نمایش الگوریتم با چهار خوشه

```
#computing kmeans on 4 clusters
k4means = k_means(dataArray,4)
#plotting clusters
x1_k4means = []
y1_k4means = []
x2_k4means = []
y2_k4means = []
x3_k4means = []
y3_k4means = []
x4_k4means = []
y4_k4means = []
for i in range(len(k4means[0])):
    x1_k4means.append(k4means[0][i][0])
    y1_k4means.append(k4means[0][i][1])
for i in range(len(k4means[1])):
    x2_k4means.append(k4means[1][i][0])
    y2_k4means.append(k4means[1][i][1])
for i in range(len(k4means[2])):
    x3_k4means.append(k4means[2][i][0])
    y3_k4means.append(k4means[2][i][1])
for i in range(len(k4means[3])):
    x4_k4means.append(k4means[3][i][0])
    y4_k4means.append(k4means[3][i][1])
plt.scatter(x1_k4means,y1_k4means,color = 'blue')
plt.scatter(x2_k4means,y2_k4means,color = 'red')
plt.scatter(x3_k4means,y3_k4means,color = 'green')
plt.scatter(x4_k4means,y4_k4means,color = 'yellow')
plt.show()
```



سوال دوم

در ابتدا کتاب خانه های مورد نیاز را import میکنیم و داده را میخوانیم

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC,SVC
#reading data drom exel
myData = pd.read_excel('dataset2.xlsx')
```

تابع زیر پیاده سازی الگوریتم pca میباشد که مطابق اسلاید ع=های درس گام به گام آن اجرا شده است

```
#function below is for implemetation pca
def pca(data,r):
    m = np.mean(data)#compute mean
    z = data - np.transpose(m)#centre data
    sigma = (1/len(data))*(np.transpose(z) @ z)#compute covariance matrix
    y,U = np.linalg.eig(sigma)#compute eigen values and eigen vectors
    Ur = U[:,:r]#reduced basis
    A = np.zeros((r,len(data)))#reduced dimentionality
    for i in range(len(data)):
        A[:,i] = np.transpose(Ur) @ np.transpose(data.iloc[i,:])
    return np.transpose(A)
```

PCA (D, r)

$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ // compute mean
$Z = D - 1 \cdot \mu^T$ // center the data
$\Sigma = \frac{1}{n}(Z^T Z)$ // compute covariance matrix
$(\lambda_1, \lambda_2, \ldots, \lambda_d) = \text{eigenvalues}(\Sigma)$ // compute eigenvalues
$U = (u_1 \quad u_2 \quad \cdots \quad u_d) = \text{eigenvectors}(\Sigma)$ // compute eigenvectors
$U_r = (u_1 \quad u_2 \quad \cdots \quad u_r)$ // reduced basis
$A = \{a_i \mid a_i = U_r^T x_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

در خط اول تابع ماتریس میانگین بدست میاوریم و در خط دوم z را بدست میاوریم ماتریس کواریانس نیز بر اساس z بدست می آید و بردار و مقادیر ویژه از ماترس کواریانس با استفاده از تابع np.linalg.eig بدست می‌آید و در نهایت ماتریس A که داده های کاهش بعد یافته است را برمیگردانیم

حال داده های کاهش بعد یافته را با لیست کلاس واقعی به صورت جداگانه داریم که با ادغام آن دو ماتریس داده کاهش یافته و از روی ماتریس دیتا فریم داده های کاهش یافته را میسازیم و بعد از آن به دو بخش آموزش و تست تقسیم میکنیم

```
#applying algorithm with data reduced to 2 dimentions
dataReduced = pca(myData[['A','B','C','D']], 2)
realClass = myData['class']
#creating matrix with two features and class lable
newDataMatrix = np.zeros((150,3))
newDataMatrix[:,0:2] = dataReduced
newDataMatrix[:,2] = realClass
#craeting data frame of that matrix and splitting test and train
newDataFrame = pd.DataFrame(newDataMatrix)
trainData,testData = train_test_split(newDataFrame,test_size = 0.2,train_size = 0.8)
```

تابع زیر نیز محاسبه دقت بر اساس داده ها و مدل ماست

```
#function below is for calculating accuracy
def accuracy_calculate(data,lables,model):
    accuracy = 0
    for i in range(len(data)):
        predictedLable = model.predict([data.iloc[i]])
        if predictedLable == lables.iloc[i]:
            accuracy += 1
    return accuracy / len(data)
```

در نهایت یک svm خطی و یک svm غیرخطی میسازیم و دقت را برای هر دو مدل محاسبه میکنیم

```
#creating linear and non linear svm
linearSvm = LinearSVC()
nonLinearSvm = SVC()
linearSvm.fit(trainData[[0,1]],trainData[2])
nonLinearSvm.fit(trainData[[0,1]],trainData[2])
#checking accuracy
print('linear svm accuracy:',accuracy_calculate(testData[[0,1]], testData[2], linearSvm))
print('non linear svm accuracy:',accuracy_calculate(testData[[0,1]], testData[2], nonLinearSvm))
```

```
linear svm accuracy: 0.9666666666666667
non linear svm accuracy: 0.9333333333333333
```