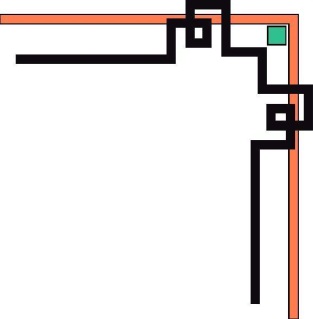
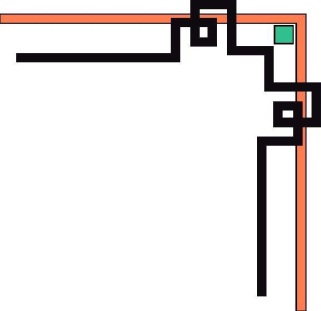
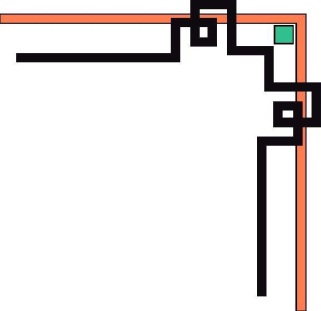
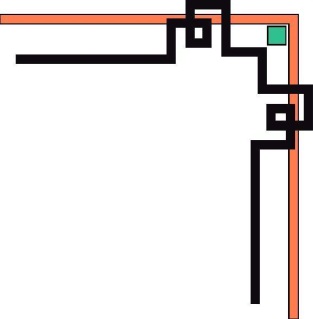
**TRƯỜNG ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH**



**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN – ĐIỆN TỬ**

**HOMEWORK**

**TRÍ TUỆ NHÂN TẠO TRONG ĐIỀU KHIỂN**

**GVHD:** Phạm Việt Cường

**Nhóm 09:**

|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **MSSV** | **Tí lệ đóng góp** |
| Trần Xuân Lộc | 1812981 | 25% |
| Phạm Nguyễn Hoàng Khương | 1812693 | 25% |
| Lê Hữu Khánh | 1812590 | 25% |
| Lê Huy | 1812361 | 25% |

HỌC KỲ 221, NĂM HỌC 2022-2023

TP. HỒ CHÍ MINH, THÁNG 09 NĂM 2022

**MỤC LỤC**

[HW1. Write a program implementing perceptron learning algorithm 2](#_Toc114580574)

[1. Lý thuyết 2](#_Toc114580575)

[2. Mô phỏng huấn luyện Perceptron bằng Python 2](#_Toc114580576)

[HW2. Compare mean squared error and mean absolute error 10](#_Toc114580577)

[HW3. Write a program implementing feature nomalization. Find and demonstrate a real world example 11](#_Toc114580578)

[1. Công thức Feature Normalize 11](#_Toc114580579)

[2. Áp dụng vào một ví dụ trong thức tế 13](#_Toc114580580)

[HW4. Write a program implementing linear regression algorithm solving a freely chosen problem 15](#_Toc114580581)

[1. Chuẩn bị dữ liệu 15](#_Toc114580583)

[2. Chọn một phương pháp phù hợp 15](#_Toc114580584)

[3. Chọn mô hình 15](#_Toc114580585)

[4. Kiểm tra chất lượng và điều chỉnh mô hình được trang bị 15](#_Toc114580586)

[HW5. Write a program inplementing kNN algorithm solving a freely chosen problem 22](#_Toc114580587)

[1. Chương trình 22](#_Toc114580588)

[2. Kết quả 23](#_Toc114580589)

[HW6. Write a program inplementing k-means clustering algorithm solving a freely chosen problem 27](#_Toc114580590)

[1. Chương trình 27](#_Toc114580591)

[2. Kết quả 28](#_Toc114580592)

[HW7. Write a programe implementing K++ algorithm (centroid initialization only) 35](#_Toc114580593)

[1. Lý thuyết 35](#_Toc114580594)

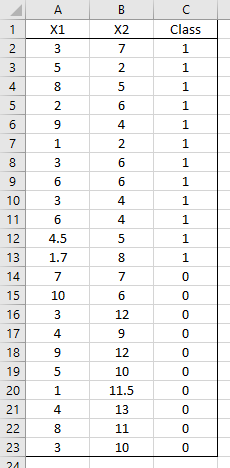
[2. Viết chương trình và ví dụ minh họa bằng Python 36](#_Toc114580595)

# **HW1. Write a program implementing perceptron learning algorithm**

1. **Lý thuyết**

Perceptron là một thuật toán Classification cho trường hợp đơn giản nhất: chỉ có 2 class và cũng chỉ hoạt động được trong một trường hợp rất cụ thể. Bài toán Perceptron được phát biểu như sau: Cho 2 class được gán nhãn, hãy tìm một đường thẳng sao cho toàn bộ các điểm thuộc class 1 nằm về 1 phía, toàn bộ các điểm thuộc class 2 nằm về phía còn lại của đường thẳng đó. Với giả định rằng tồn tại một đường thẳng như thế.

Giả sử ta có 2 tập dữ liệu là X1 và X2 đã được gán nhãn theo thứ tự là “green” và “red”.



1. **Mô phỏng huấn luyện Perceptron bằng Python**

* Đọc dữ liệu từ file CSV và biểu diễn trên đồ thị

def Read\_Data():

df = pd.read\_csv('D:\HCMUT\AI\HW\PLA.csv')

    plt.plot(df['X1'][1:12], df['X2'][1:12], 'go', color='green')

    plt.plot(df['X1'][12:], df['X2'][12:], 'gs', color='red')

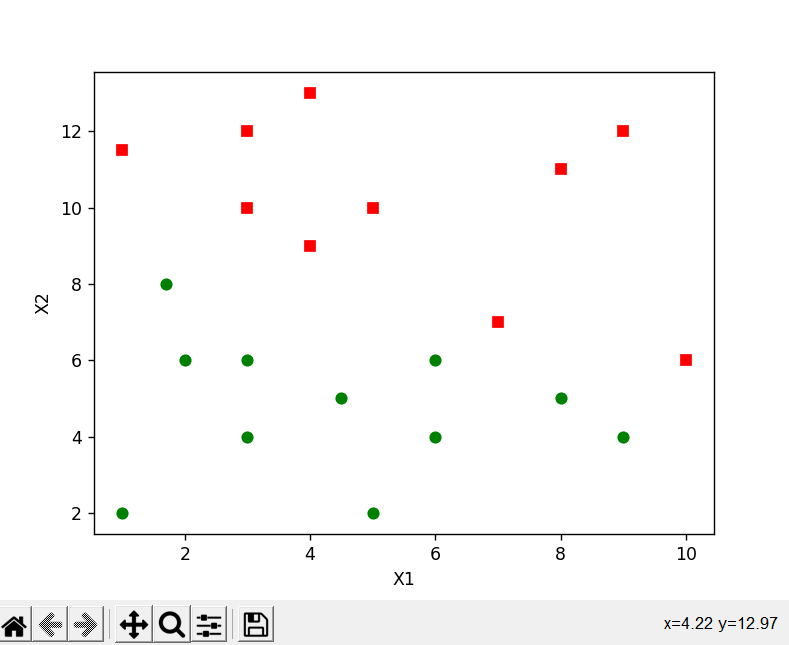
    plt.xlabel('X1') *# label x*

    plt.ylabel('X2') *# label y*

    plt.show()

    return df

* Hàm Read\_Data(): trả về df để lưu dữ liệu.
* Kết quả sau khi plot



* Tiến hành Perceptron Learning Algorithm
* Khởi tạo mảng X =[], thêm vào dữ liệu các cột X1, X2 và X có dạng: [ [1, X1, X2] ]

def PLA(data):

    X = []

    for i in range(0, 22):

        X.append([1, data['X1'][i], data['X2'][i]])

* Tạo W = [0, 0, 0]

W = np.zeros(len(X[0]))

* Chọn miền hội tụ là 400. Và tiến hành giải thuật PLA

for i in range(400):

        for i in range(len(X)):

            if data['Class'][i] == 1 and (W @ np.array(X[i])) < 0:

                W += X[i]

            if data['Class'][i] == 0 and (W @ np.array(X[i])) >= 0:

                W -= X[i]

print("W =", W)

    sampleX = np.linspace(min(data['X1']) - 3, max(data['X1']) + 3, int((max(data['X1']) - min(data['X1']))\*100))

    sampleY = (-W[0] - W[1]\*sampleX)/W[2]

    plt.plot(sampleX, sampleY, color='black')

    plt.plot(data['X1'][1:12], data['X2'][1:12], 'go', color='green')

    plt.plot(data['X1'][12:], data['X2'][12:], 'gs', color='red')

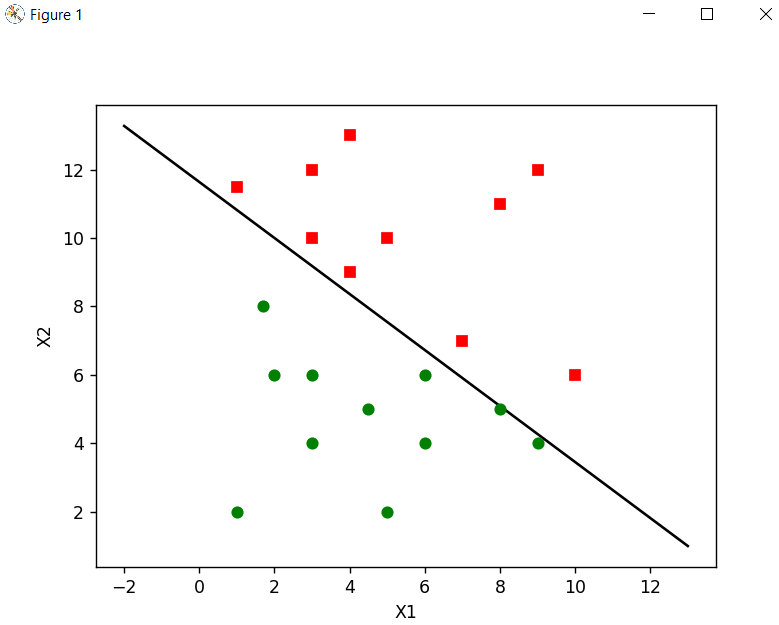
    plt.xlabel('X1') *# label x*

    plt.ylabel('X2') *# label y*

    plt.show()

    return W

* Sau đó in ra màn hình giá trị của W và vẽ đường phân tách 2 class
* Kết quả sau khi chạy



* Và W = [128 -9 -11]
* Dùng Perceptron để phân loại các điểm dữ liệu
* Tạo các điểm chưa phân loại bằng hàm random và được biểu diễn bằng hình tam giác

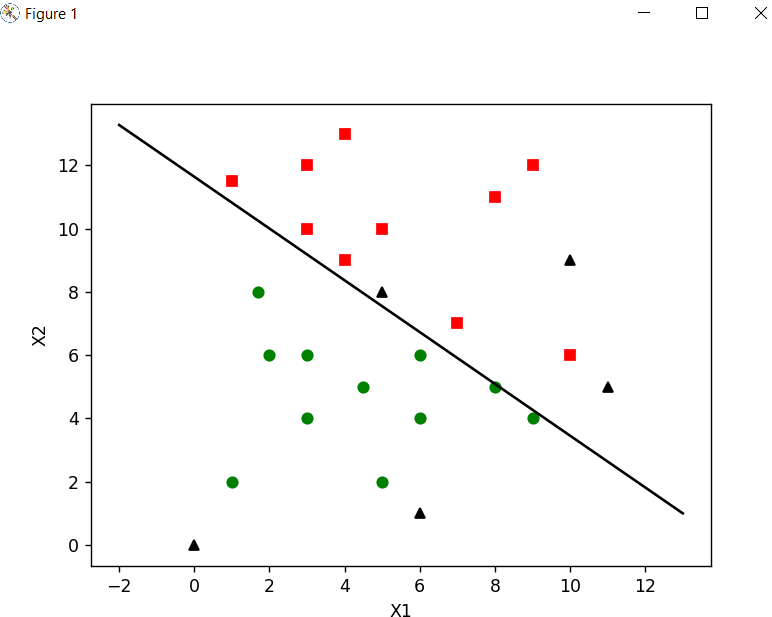
x\_test = []

    for i in range(5):

        x\_test.append([1, random.randint(0, 11), random.randint(0, 11)])

        plt.plot(x\_test[i][1], x\_test[i][2], 'b^', color='black')

* Kết quả



* Hàm phân loại

for i in range(len(x\_test)):

        if W @ np.array(x\_test[i]) < 0:

            plt.plot(x\_test[i][1], x\_test[i][2], 'b^', color='red')

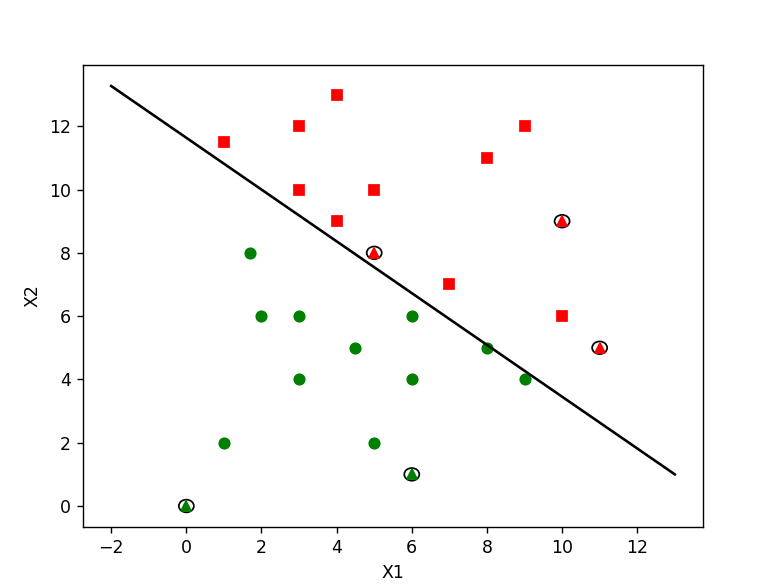
        else:

            plt.plot(x\_test[i][1], x\_test[i][2], 'b^', color='green')

        circle1 = plt.Circle((x\_test[i][1], x\_test[i][2]), .2, color='black', fill = False)

        plt.gcf().gca().add\_artist(circle1)

* Kết quả sau khi phân loại



* Full code

from os import X\_OK

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import random

*#Input*

def Read\_Data():

    df = pd.read\_csv('D:\HCMUT\AI\HW\PLA.csv')

    plt.plot(df['X1'][1:12], df['X2'][1:12], 'go', color='green')

    plt.plot(df['X1'][12:], df['X2'][12:], 'gs', color='red')

    plt.xlabel('X1') *# label x*

    plt.ylabel('X2') *# label y*

    plt.show()

    return df

*#Perceptron Learning Algorithm*

def PLA(data):

    X = []

    for i in range(0, 22):

        X.append([1, data['X1'][i], data['X2'][i]])

    W = np.zeros(len(X[0]))

    for i in range(400):

        for i in range(len(X)):

            if data['Class'][i] == 1 and (W @ np.array(X[i])) < 0:

                W += X[i]

            if data['Class'][i] == 0 and (W @ np.array(X[i])) >= 0:

                W -= X[i]

    print("W =", W)

    sampleX = np.linspace(min(data['X1']) - 3, max(data['X1']) + 3, int((max(data['X1']) - min(data['X1']))\*100))

    sampleY = (-W[0] - W[1]\*sampleX)/W[2]

    plt.plot(sampleX, sampleY, color='black')

    plt.plot(data['X1'][1:12], data['X2'][1:12], 'go', color='green')

    plt.plot(data['X1'][12:], data['X2'][12:], 'gs', color='red')

    plt.xlabel('X1') *# label x*

    plt.ylabel('X2') *# label y*

    plt.show()

    return W

*#Main*

if \_\_name\_\_ == "\_\_main\_\_":

    data = Read\_Data()

    W = PLA(data)

    x\_test = []

    for i in range(5):

        x\_test.append([1, random.randint(0, 11), random.randint(0, 11)])

        plt.plot(x\_test[i][1], x\_test[i][2], 'b^', color='black')

    plt.plot(data['X1'][1:12], data['X2'][1:12], 'go', color='green')

    plt.plot(data['X1'][12:], data['X2'][12:], 'gs', color='red')

    sampleX = np.linspace(min(data['X1']) - 3, max(data['X1']) + 3, int((max(data['X1']) - min(data['X1']))\*100))

    sampleY = (-W[0] - W[1]\*sampleX)/W[2]

    plt.plot(sampleX, sampleY, color='black')

    plt.xlabel('X1') *# label x*

    plt.ylabel('X2') *# label y*

    plt.show()

    for i in range(len(x\_test)):

        if W @ np.array(x\_test[i]) < 0:

            plt.plot(x\_test[i][1], x\_test[i][2], 'b^', color='red')

        else:

            plt.plot(x\_test[i][1], x\_test[i][2], 'b^', color='green')

        circle1 = plt.Circle((x\_test[i][1], x\_test[i][2]), .2, color='black', fill = False)

        plt.gcf().gca().add\_artist(circle1)

    plt.plot(data['X1'][1:12], data['X2'][1:12], 'go', color='green')

    plt.plot(data['X1'][12:], data['X2'][12:], 'gs', color='red')

    sampleX = np.linspace(min(data['X1']) - 3, max(data['X1']) + 3, int((max(data['X1']) - min(data['X1']))\*100))

    sampleY = (-W[0] - W[1]\*sampleX)/W[2]

    plt.plot(sampleX, sampleY, color='black')

    plt.xlabel('X1') *# label x*

    plt.ylabel('X2') *# label y*

    plt.show()

# **HW2. Compare mean squared error and mean absolute error**

* **MSE**
* Là số liệu phổ biến nhất được sử dụng cho các bài toán Regression. Về cơ bản, nó tìm bình phương sai số trung bình giữa các giá trị được dự đoán và thực tế. MSE là thước đo chất lượng của một công cụ ước tính – nó luôn không âm và các giá trị càng gần 0 càng tốt.
* Khi vẽ biểu đồ của toàn bộ dữ liệu, MSE cho biết mức độ gần của đường hồi quy với một tập hợp các điểm. Để giảm thiểu MSE, mô hình có thể chính xác hơn, có nghĩa là mô hình gần với dữ liệu thực tế hơn. MSE càng thấp thì dự báo càng tốt.
* Công thức của MSE là

(Với là biến quan sát; là giá trị dự đoán)

* **MAE:** Sai số trung bình tuyệt đối, đo độ lớn trung bình các lỗi trong 1 tập hợp các dự đoán mà không cần xem xét hướng của chúng.
* Công thức của MAE là

(Với là giá trị dự đoán; là giá trị thực)

* Chính vì vậy, sử dụng MSE nếu các giá trị ngoại lệ ( các giá trị có sai số cao hơn hẳn các mẫu khác) là quan trọng. Lý do chính là trong MSE có bình phương sai số, các mẫu ngoại lệ sẽ được chú ý nhiều hơn. Và nếu các ngoại lệ không quan trọng thì sử dụng MAE.

# **HW3. Write a program implementing feature nomalization. Find and demonstrate a real world example**

## **Công thức Feature Normalize**

**a. Feature Scaling**

**Standard Deviation (độ lệch chuẩn)** có thể hiểu là độ biến động của dữ liệu. Dữ liệu càng ổn định (ít chênh lệch), độ lệch chuẩn càng thấp.

trong numpy, ta có thể tính độ lệch chuẩn bằng hàm **std()**

**np.std(<mảng dữ liệu>,<index>,dtype = <kiểu dữ liệu>)**

**Trong đó**

* **Mảng dữ liệu** là ma trận hoặc vector cần tìm độ lệch chuẩn, trong Feature Normalize chính là ma trận X input của chúng ta.
* **Index**: Nếu bằng 0 sẽ tính độ lệch chuẩn theo từng cột (trả về độ lệch chuẩn của từng cột), bằng 1 sẽ tính theo hàng.
* **Kiểu dữ liệu:** Kiểu dữ liệu càng phức tạp, độ chính xác càng cao, trong bài này sử dụng kiểu **np.float64**.

**Công thức Feature Scaling**

Để thực hiện Feature Scaling, ta chỉ cần lấy X chia cho độ lệch chuẩn

**Xscaled = X/s với s=std(X)**

Tuy nhiên, ta sẽ không thực hiện Feature Normalize vớ**i bias feature (x0 = 1)**, vì thế sau khi Feature Normalize xong, ta sẽ gán giá trị **x0 = 1** lại.

Thực hiện std cho x0 sẽ trả về kết quả là 0, nếu tiếp tục lấy x/0 sẽ bị lỗi chia cho 0, nên ta phải cho x0 một độ biến động giả bằng cách gán x0 đầu tiên bằng 100 (để x0 không còn “luôn bằng 1”)

Ta thực hiện Feature Scaling với **numpy**như sau

import numpy as np

def Normalize(X):

#tạo copy của X(tham chiếu X) để không ảnh hưởng trực tiếp đến X (tham trị)

n=np.copy(X)

#x0 đầu tiên giả = 100

n[0,0]=100

#tính std cho từng feature x

s=np.std(n,0,dtype = np.float64)

n=n/s

#gán lại x0=1

n[:;0]=1

**b. Mean Normalize**

Để thực hiện **Mean Normalize**, ta sẽ tính trung bình từng feature rồi lấy feature cũ trừ trung bình.

Hàm tính trung bình – **mean()**với numpy

**np.std(<mảng dữ liệu>,<index>)**

**Trong đó**

* **Mảng dữ liệu**là ma trận hoặc vector cần tìm trung bình, trong Feature Normalize chính là ma trận X input của chúng ta.
* **Index**: Nếu bằng 0 sẽ tính trung bình theo từng cột (trả về trung bình của từng cột), bằng 1 sẽ tính theo hàng.

Công thức Feature Normalize khi có Mean Normalize

**Xnorm = (X-\mu )/s với \mu  = mean(X)**

**Trong đó**

* đọc là mu

Áp dụng vào hàm Normalize

import numpy as np

def Normalize(X):

#tạo copy của X(tham chiếu X) để không ảnh hưởng trực tiếp đến X (tham trị)

n=np.copy(X)

#x0 đầu tiên giả = 100

n[0,0]=100

#tính std cho từng feature x

s=np.std(n,0,dtype = np.float64)

#tinh mean cho từng feature x

mu=np.mean(n,0)

n=(n-mu)/s

#gán lại x0=1

n[:;0]=1

**c. Trả kết quả**

Hàm normalize sẽ trả về 3 kết quả: **norm**, \mu và **s**. \mu và s được trả về để thuận tiện cho việc predict (trước khi predict phải Normalize input).

import numpy as np

def Normalize(X):

#...

yield n

yield mu

yield s

1. **Áp dụng vào một ví dụ trong thức tế**

Áp dụng Feature Normalize vào vấn đề đa biến

**a. Load data**

<https://drive.google.com/file/d/1FW6i5viMJQco5U--kBU9UL8LvMhCpeed/view?usp=sharing>

Ta sẽ define một hàm riêng để load data, tiết kiệm thời gian hơn

import numpy as np

defLoadtxt(path)

try:

raw=np.loadtxt(path,delimiter=’,’)

X=np.zeros((np.size(raw,0),np.size(raw,1)))

X[:,0]=1

X[:,1:]=raw[:,:-1]

Y=raw[ơ:,-1]

Yield X

Yield y

except:

return 0

**b. Normalize data**

import numpy as np

from function import\*

[X,y]=Loadtxt(‘data.txt’)

[X,mu,s]=Normalize(X)

**c. Train data**

Để train data, ta gọi hàm Gradient Descent đã viết sẵn với **alpha = 0.1** và **iter = 400**

import numpy as np

from function import\*

[X,y]=Loadtxt(‘data.txt’)

[X,mu,s]=Normalize(X)

[Theta,J\_hist]=GradientDescent(X,y,0.1,400)

**d. Predict kết quả**

Đầu tiên, ta phải normalize input với mu và s, sau đó mới dùng hàm predict

import numpy as np

from function import\*

[X,y]=Loadtxt(‘data.txt’)

[X,mu,s]=Normalize(X)

[Theta,J\_hist]=GradientDescent(X,y,0.1,400)

input=np.array([1,1650,3])

input=(input-mu)/s

#Lưu ý sửa lại x0=1

input[0]=1

predict=predict(input,Theta)

print(‘%.2f$’%(predict))

# **HW4. Write a program implementing linear regression algorithm solving a freely chosen problem**

## Chương trình linear regression

## **Chuẩn bị dữ liệu**

mdl = fitlm(tbl,'ResponseVar','BloodPressure');

% or

mdl = fitglm(tbl,'ResponseVar','BloodPressure');

ds = dataset('XLSFile','hospital.xls',...

'ReadObsNames',true);

load carsmall

ds = dataset(MPG,Weight);

ds.Year = ordinal(Model\_Year);

tbl = readtable('hospital.xls',...

'ReadRowNames',true);

load carsmall

tbl = table(MPG,Weight);

tbl.Year = ordinal(Model\_Year);

load carsmall

X = [Weight Horsepower Cylinders Model\_Year];

y = MPG;

[X Xnames] = xlsread('hospital.xls');

y = X(:,4); % response y is systolic pressure

X(:,4) = []; % remove y from the X matrix

## **Chọn một phương pháp phù hợp**

#### Least-Squares Fit

#### Robust Fit

#### Stepwise Fit

## **Chọn mô hình**

#### Brief String

#### Terms Matrix

#### Formula

## **Kiểm tra chất lượng và điều chỉnh mô hình được trang bị**

#### **Model Display**

x=randn(100,5);

y=X\*[1;0;3;3;-1]+randn(100,1);

md1=fitlm(X,y)

#### **ANOVA**

x=randn(100,5);

y=X\*[1;0;3;3;-1]+randn(100,1);

md1=fitlm(X,y)

tbl=anova(md1)

#### **Diagnostic Plots**

load carsmall

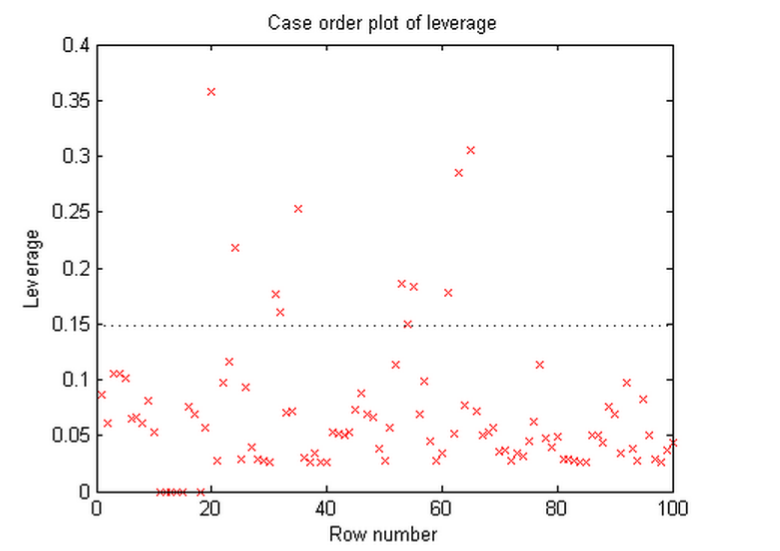
tbl=table(Weight,MPG,Cylinders);

tbl.Cylinders=ordinal(tbl.Cylinders);

mdl=fitlm(tbl,’MPG~Cylinders\*Weight+Weight^2);

plotDiagnostics(mdl)

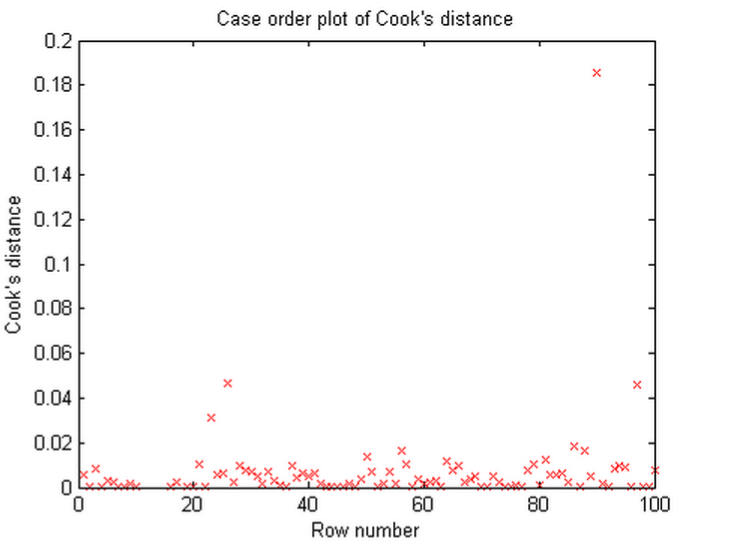
Biểu đồ leverage của dữ liệu và mô hình



Có một số điểm với everage cao. Nhưng biểu đồ này không tiết lộ liệu các điểm everage cao có phải là ngoại lệ hay không

Tìm các điểm có khoảng cách Cook lớn.

plotDiagnostics(mdl,’cookd’)



Có một điểm có khoảng cách Cook lớn. Xác định nó và loại bỏ nó khỏi mô hình.

[~,larg]=max(mdl.Diagnostics.CooksDistance);

mdl2=fitlm(tbl,’MPG~Cylinders\*Weight+Weight^2,…

‘Exclude’,larg);

#### **Residuals — Model Quality for Training Data**

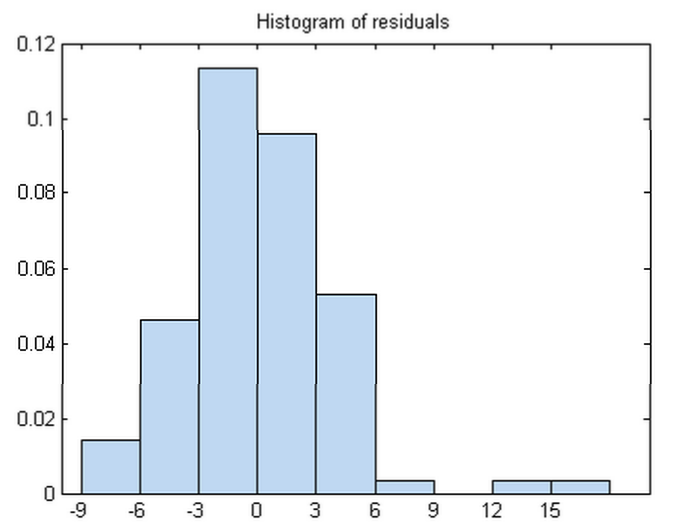
load carsmall

tbl=table(Weight,MPG,Cylinders);

tbl.Cylinders=ordinal(tbl.Cylinders);

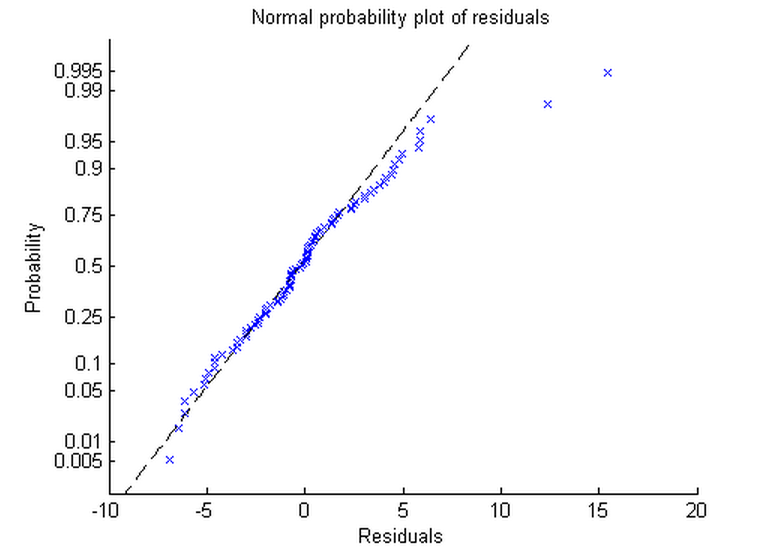
mdl=fitlm(tbl,’MPG~Cylinders\*Weight+Weight^2);

plotResiduals(mdl)



Các quan sát trên 12 là những ngoại lệ tiềm ẩn

plotResiduals(mdl,’probability’)



Hai ngoại lệ tiềm năng cũng xuất hiện. Nếu không, biểu đồ xác suất có vẻ thẳng hợp lý, có nghĩa là phù hợp hợp lý với phần dư được phân phối chuẩn.

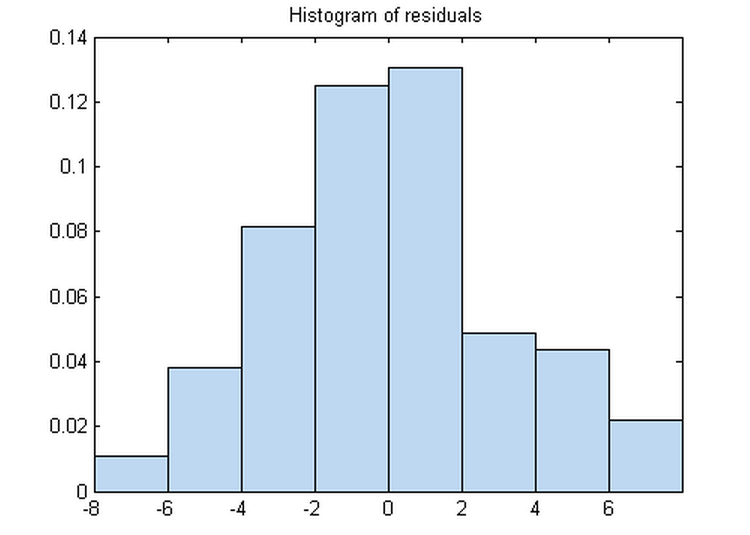
Bạn có thể xác định hai ngoại lệ và loại bỏ chúng khỏi dữ liệu

out1=find(mdl.Residuals.Raw > 12)

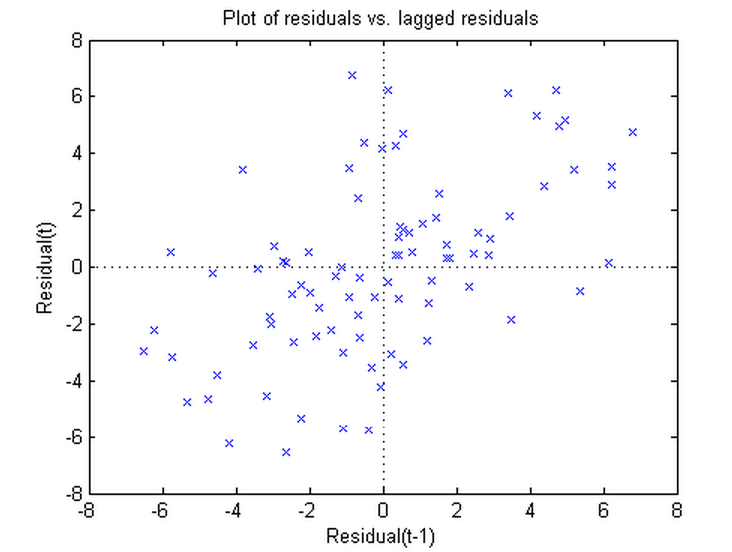
mdl2=fitlm(tbl,’MPGCylinders\*Weight+Weight^2,…

‘Exclude’,out1);

PlotResiduals(mdl2)

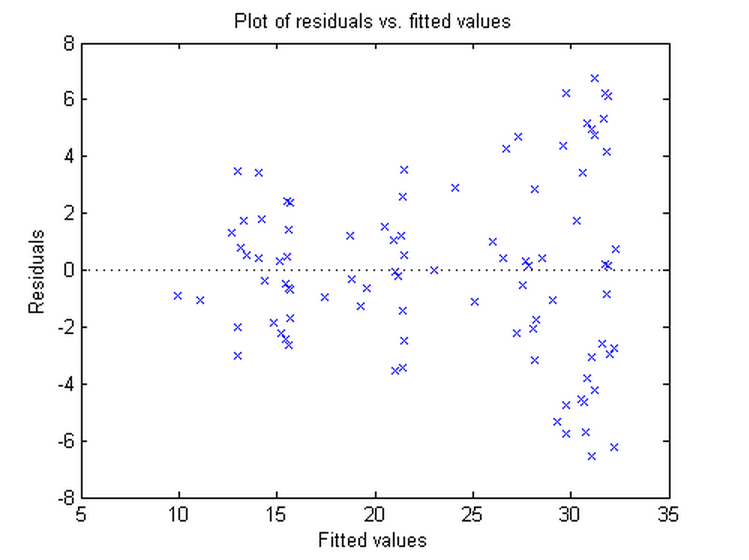


plotResiduals(mdl2,’lagged’)



Biểu đồ phân tán cho thấy nhiều dấu gạch chéo ở góc phần tư phía trên bên phải và phía dưới bên trái hơn ở hai góc phần tư còn lại, cho thấy mối tương quan nối tiếp dương giữa các phần còn lại. Một vấn đề tiềm ẩn khác là khi lượng dư lớn đối với các quan sát lớn. Xem liệu mô hình hiện tại có vấn đề này không.

plotResiduals(mdl2,’fitted’)



#### **Plots to Understand Terms Effects**

Tạo mô hình quãng đường từ một số yếu tố dự đoán trong dữ liệu ô tô nhỏ

load carsmall

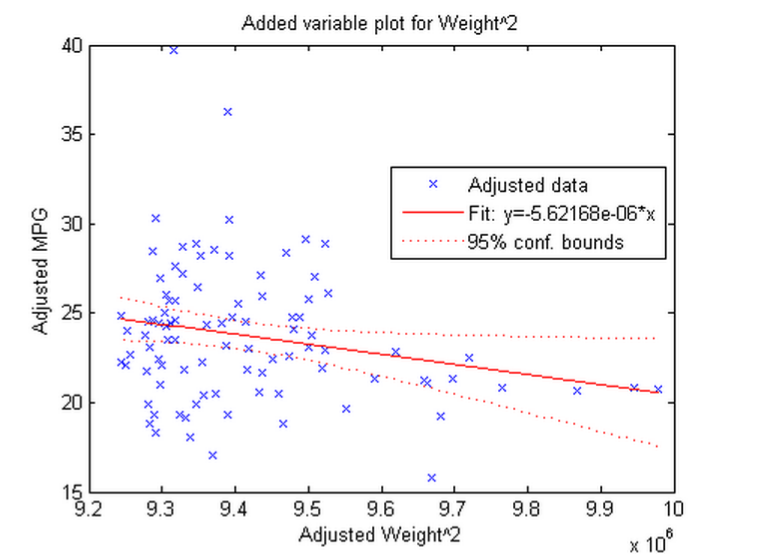
tbl=table(Weight,MPG,Cylinders);

tbl.Cylinders=ordinal(tbl.Cylinders);

mdl=fitlm(tbl,’MPG~Cylinders\*Weight+Weight^2);

Tạo một biểu đồ biến được thêm vào với Weight ^ 2 là biến được thêm vào.

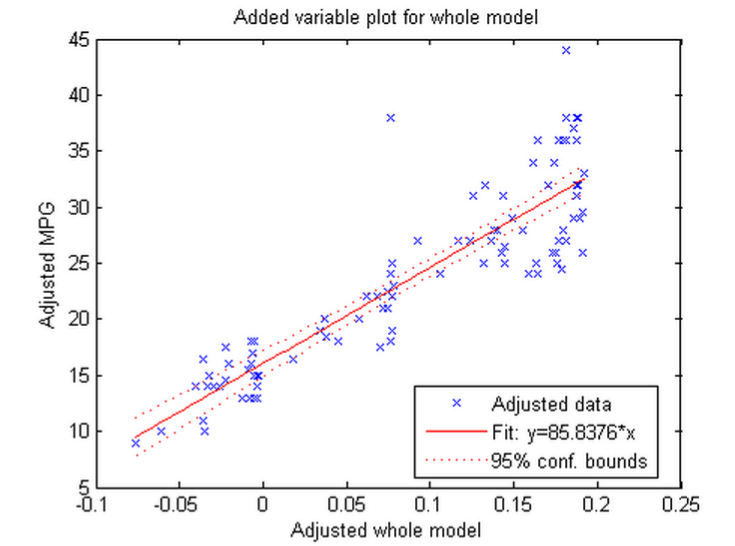
plotAdded(md1,’Weight^2)



Biểu đồ này cho thấy kết quả của việc khớp cả Trọng lượng ^ 2 và MPG với các điều khoản khác với Trọng lượng ^ 2. Lý do sử dụng plotAdded là để hiểu bạn nhận được sự cải tiến bổ sung nào trong mô hình bằng cách thêm Weight ^ 2. Hệ số của một đường phù hợp với những điểm này là hệ số Trọng lượng ^ 2 trong mô hình đầy đủ. Dự đoán Weight ^ 2 chỉ vượt quá mức ý nghĩa (pValue <0,05) có thể thấy trong màn hình bảng hệ số. Các giới hạn tin cậy có vẻ như chúng không thể chứa một đường nằm ngang (không đổi y), do đó, mô hình 0 độ dốc không phù hợp với dữ liệu.

Tạo một biểu đồ biến được bổ sung cho toàn bộ mô hình

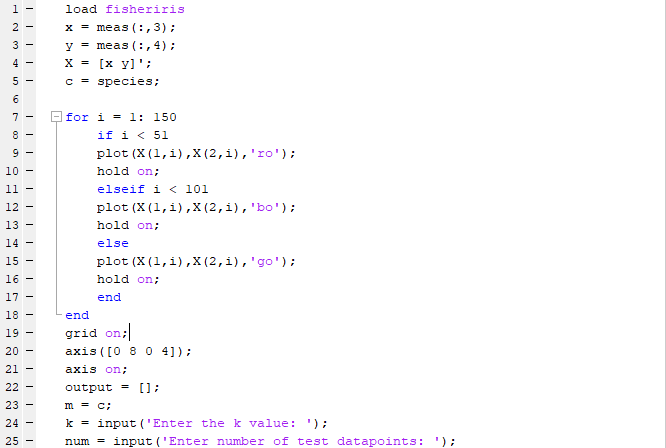
plotAdded(md1)

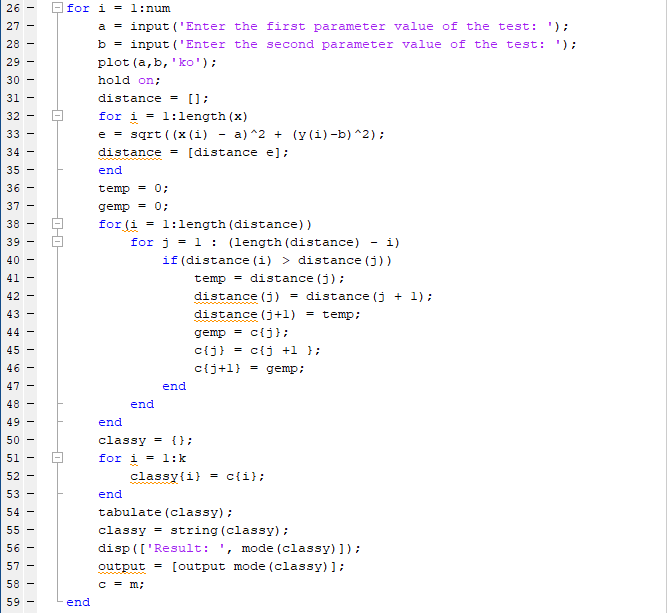


Mô hình nói chung là rất quan trọng, vì vậy các giới hạn không đến gần với việc chứa một đường ngang. Độ dốc của đường là độ dốc của sự phù hợp với các yếu tố dự đoán được chiếu lên hướng phù hợp nhất của chúng, hay nói cách khác, là chuẩn của vectơ hệ số.

**HW5. Write a program inplementing kNN algorithm solving a freely chosen problem**

1. **Chương trình**

****

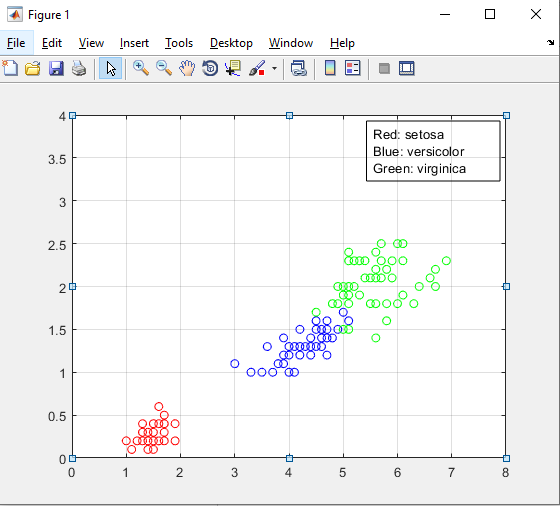
****

* Đầu tiên, ta tải tập dữ liệu fisheriris
* Ta chọn 2 trong 4 thông số bất kì để phân loại các loài hoa và lưu chúng vào các biến x, y
* Sau đó, chúng ta tiến hành vẽ các dữ liệu để dễ quan sát.
* Các bước thực hiện thuật toán:
* Đầu tiên, cho người dùng nhập thông số K và số mẫu cần test num
* Sau đó, người dùng sẽ nhập thông số cho mỗi dữ liệu mà họ muốn test
* Chương trình sẽ tính toán khoảng cách từ điểm dữ liệu do người dùng nhập đến các data point và sắp xếp theo thứ tự từ bé đến lớn
* Sau đó, với số K được chọn trước, chúng ta sẽ tiến hành so sánh các vị trí gần nhất, cái nào chiếm số lượng lớn hơn tức cái đó là kết quả khi áp dụng thuật toán KNN
* Lặp lại cho đến khi test xong các mẫu

1. **Kết quả**

**a. Với số k nhỏ**

* Đây là tập dữ liệu được vẽ

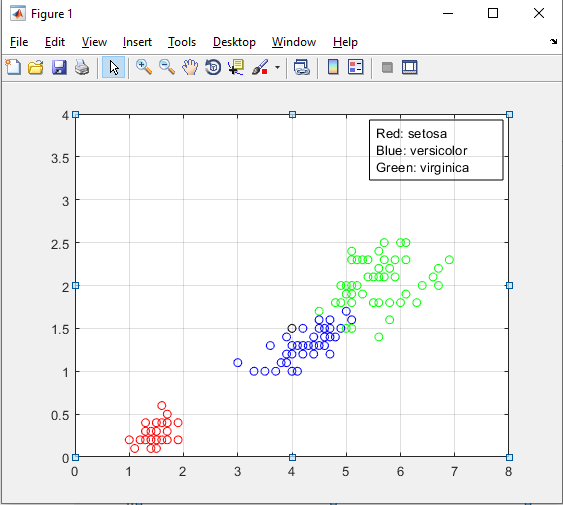


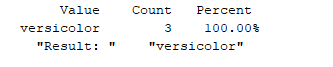
* Tiến hành nhập thông số K và số mẫu cần test: Chúng ta chọn k = 3 và num = 2

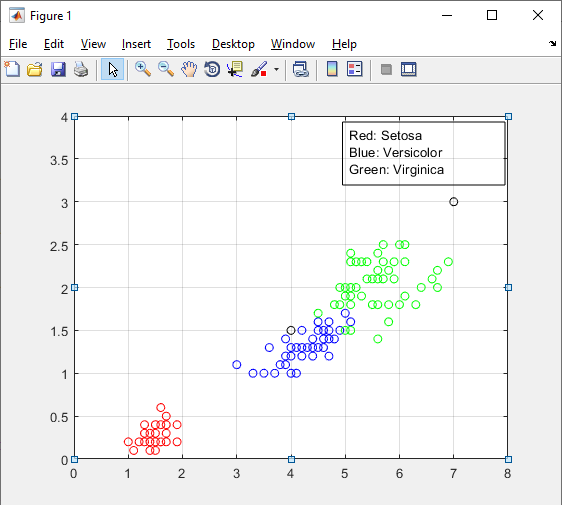


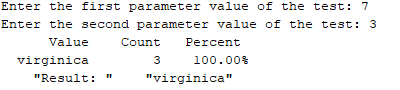
* Vẽ các điểm dữ liệu cần test:





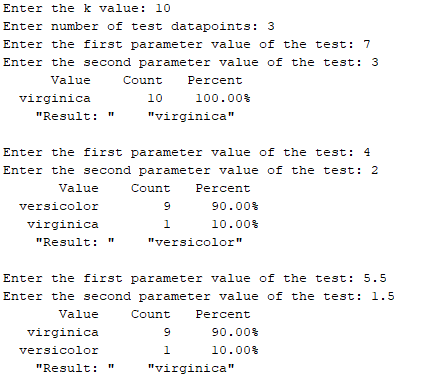
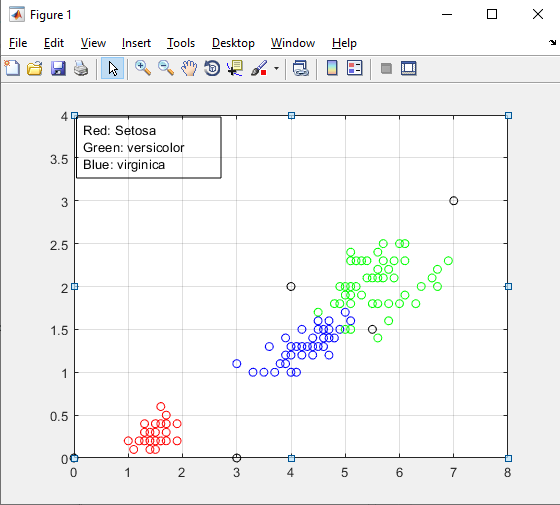


* Với dữ liệu đầu tiên, kết quả sẽ cho ra ‘Versicolor’
* Tương tự với dữ liệu thứ 2, kết quả sẽ cho ra Virginica

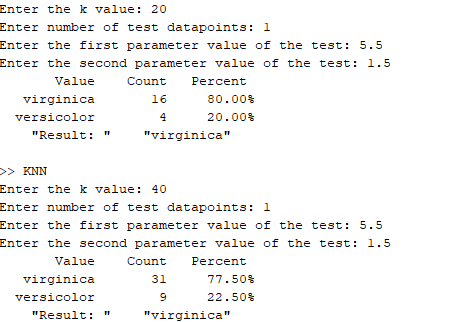


1. **Với số k lớn**

* Với cùng tập dữ liệu trên, ta sẽ tiến hành tương tự nhưng lúc này chọn số K lớn hơn là 10 và ta có kết quả dưới đây

****

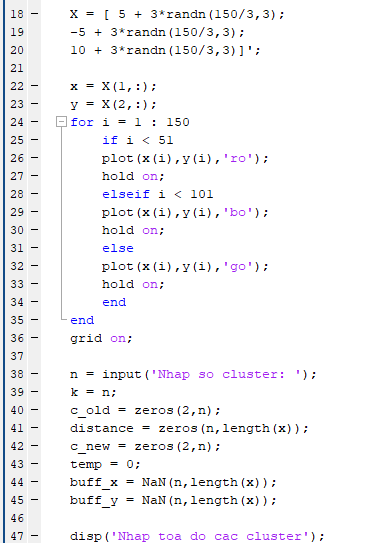
* Với K = 10 và với các dữ liệu được đưa vào test, ta nhận thấy kết quả cho ra vẫn hợp lí
* Tiếp tục, ta chọn k = 20 và 40 cho dữ liệu dưới đây, dễ dàng nhận thấy, khi K càng lớn, tính cục bộ của dữ liệu càng mất đi, rõ ràng có thể nhận ra phần trăm của loài versicolor tăng dần mặt dù đối tượng chúng ta nhận biết là virginica

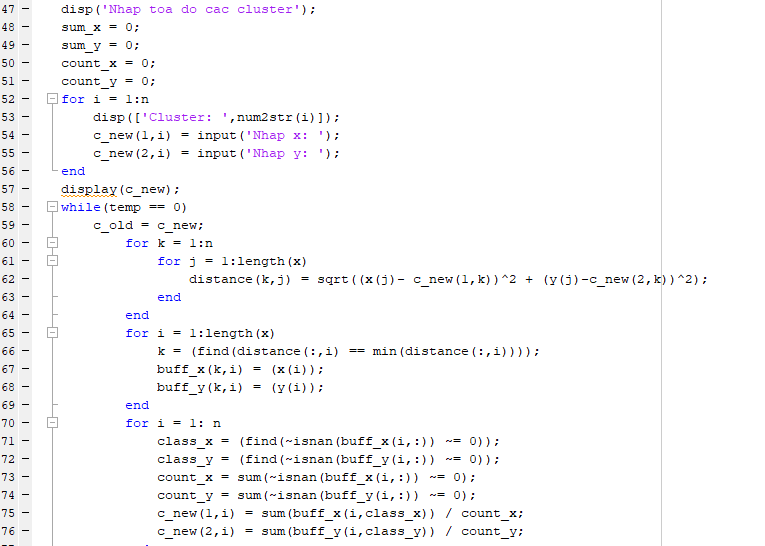
****

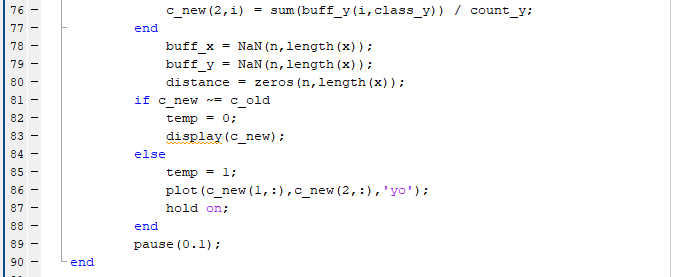
# **HW6. Write a program inplementing k-means clustering algorithm solving a freely chosen problem**

1. **Chương trình**

* Đầu tiên, ta tạo tập dữ liệu, ở đây, ta sẽ tạo ra dữ liệu với 3 nhóm tương ứng 3 màu sắc khác nhau
* Sau đó, người dùng sẽ nhập các cực và nhập lần lượt giá trị tọa độ của các cực
* Chương trình chúng ta sẽ tính toán khoảng cách từ các datapoint tới các cực và tính xem datapoint đó thuộc cực nào
* Sau khi thực hiện xong bước trên, chương trình sẽ tiến hành tính toán ra các cực mới
* Nếu giá trị cực mới trùng với giá trị cực cũ, chương trình dừng, nếu không chương trình sẽ lặp lại các bước trên cho đến khi thuật toán thỏa điều kiện hội tụ

****

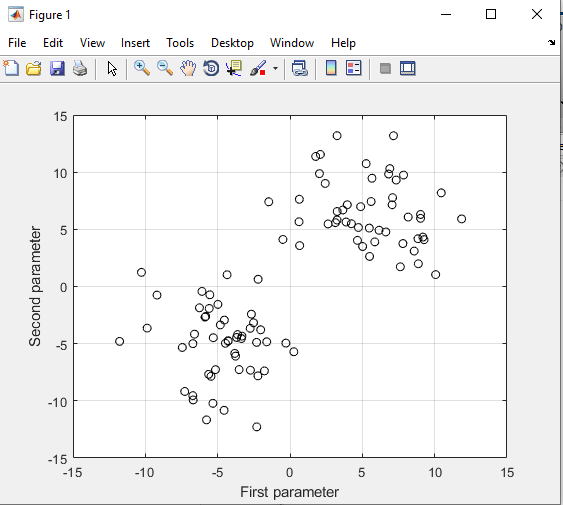
****

****

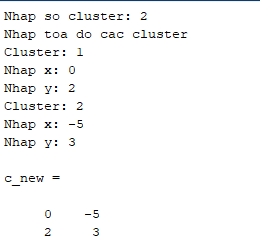
1. **Kết quả**

**a. Dữ liệu được chia thành 2 nhóm**

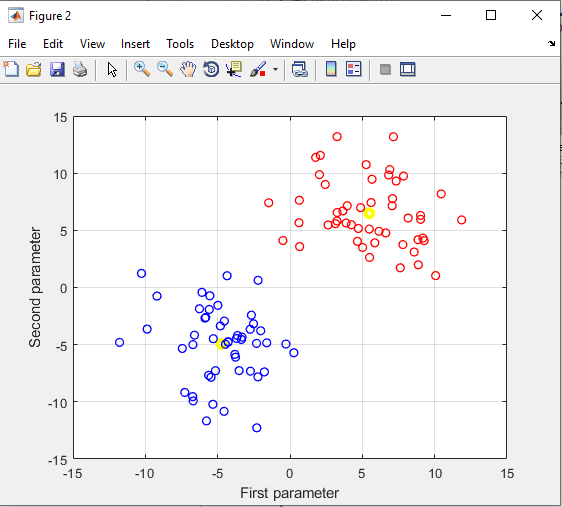
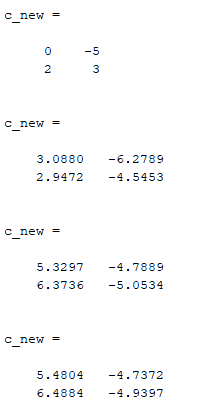
* Đầu tiên, ta tạo tập dữ liệu dưới đây



* Sau đó, người dùng sẽ nhập số cluster và nhập tọa độ các cluster vào như sau:



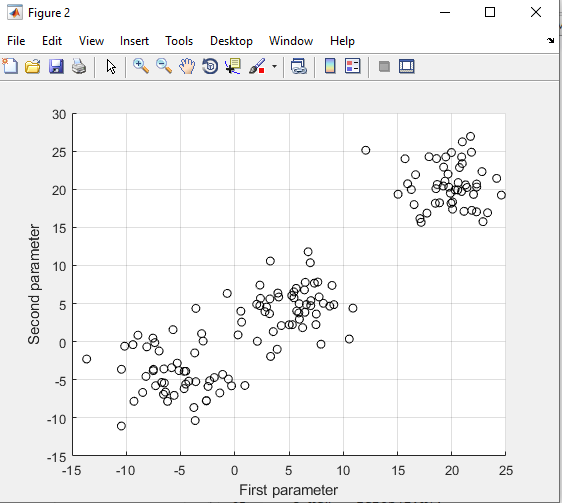
* Chương trình sẽ tính toán các cực mới cho đến khi thuật toán hội tụ và vẽ các cực đó lên mặt phẳng, đồng thời chương trình chúng ta cũng sẽ tô màu riêng biệt các datapoint để phân biệt nó thuộc cực nào



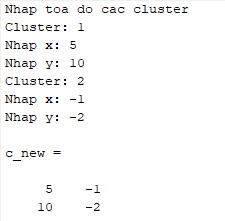
* Như kết quả tính toán được, chương trình chúng ta sẽ vẽ ra 2 cực được tô màu vàng và các phân tử thuộc các cực đó bởi 2 màu khác nhau

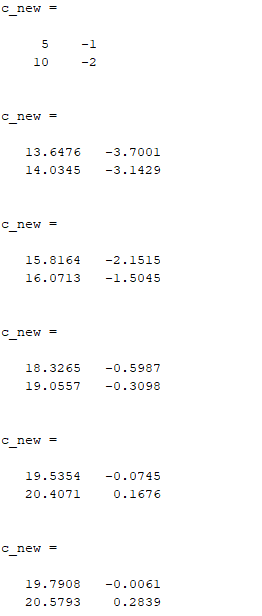
1. **Tiếp tục với dữ liệu khác**

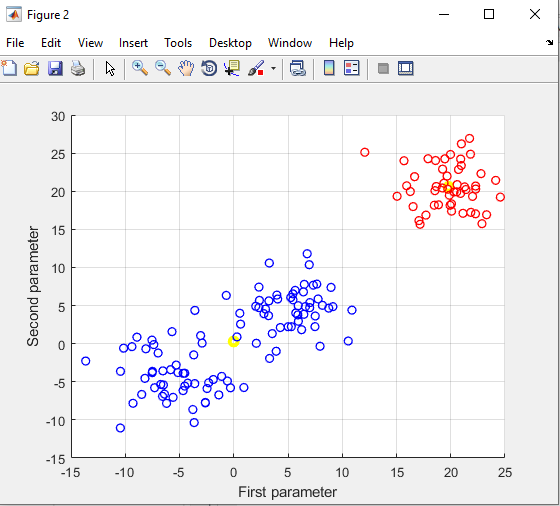
* **Số cực là 2**
* Chúng ta tạo dữ liệu như sau:



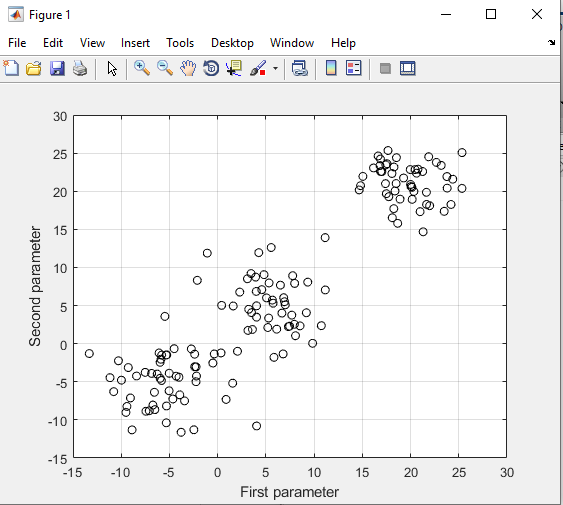
* Sau đó, người dùng vẫn nhập số cluster và tọa độ của chúng, ta tiên, ta sẽ thử nhập số cluser là 2



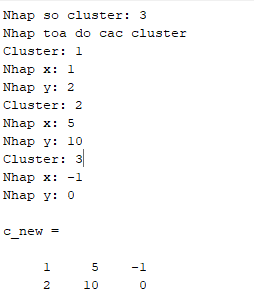
* Giống như trên, chương trình sẽ tính toán ra giá trị mới của các cực cho đến khi thuật toán hội tụ và tiến hành vẽ chúng



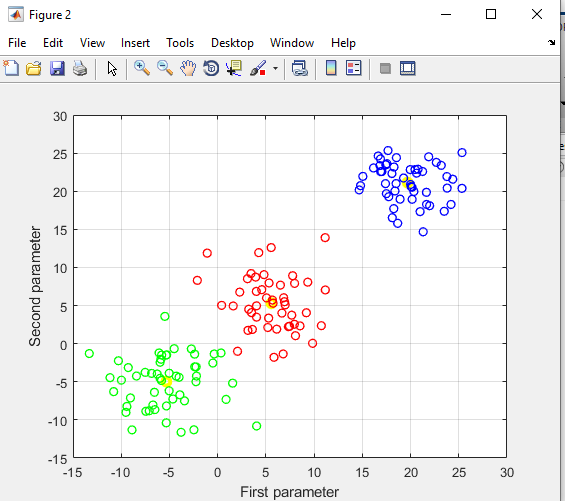
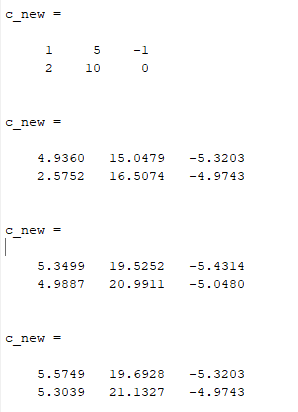
* Nhận xét: Chúng ta có thể thấy dữ liệu chúng ta được phân thành 2 phần rõ rệt do chúng ta chọn số cực là 2. Tiếp theo, chúng ta sẽ chọn số cực là 3 để xem chương trình sẽ tính toán như thế nào
* **Số cực là 3**
* Chúng ta tạo 1 tập dữ liệu gần giống như dữ liệu phía trên



* Tiếp sau đó, ta nhập số cực là 3 và tiến hành nhập tọa độ các cực



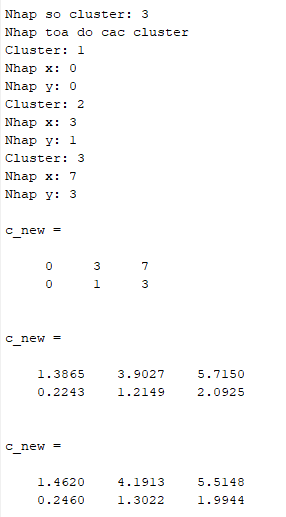
* Chương trình sẽ thực thi tính toán tọa độ các cực cho đến khi thuật toán hội tụ. Kết quả ta được như sau:

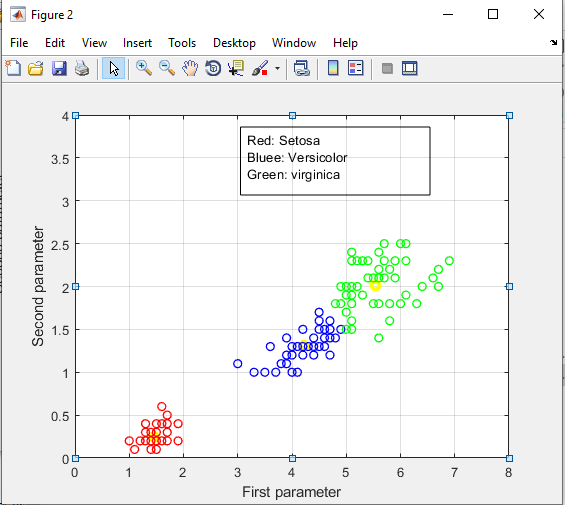
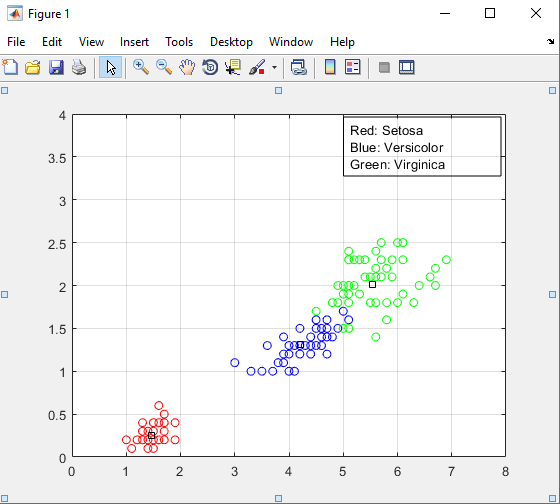


* Vậy với cùng một dạng dữ liệu, khi chọn số lượng cực khác nhau và nhập tọa độ khác nhau thì chương trình sẽ tính toán ra các cực khác nhau và phân nhóm các datapoint theo cách khác nhau.

1. **Giải quyết 1 bài toán thực tế**

* Phân loại hoa Iris, ta cũng thực hiện các bước trên và tìm ra các cực



* Hình bên trái là hình dữ liệu được xếp theo loại hoa chính xác, còn hình bên phải là hình chúng ta dùng thuật toán KMC để tìm ra các cụm hoa có đặc điểm giống nhau. Với thuật toán KMC với số cực là 3 trong trường hợp này, việc phân cụm các loài hoa gần giống với dữ liệu thực. Tuy nhiên, do muốn phân cụm các loài hoa Iris ta cần 4 thông số, nhưng chúng ta chỉ sử dụng 4 thông số trong đó nên việc phân cụm còn thiếu tính chính xác.

**HW7. Write a programe implementing K++ algorithm (centroid initialization only)**

1. **Lý thuyết**

**K – Means Clustering:** là thuật toán phân cụm dữ liệu được đặc trưng bởi một phần tử trung tâm (centroid). Điểm trung tâm này sẽ đại diện cho cụm và sẽ có giá trị là trung bình của tất cả quan sát nằm trong cụm. Dựa vào khoảng cách đến các tâm mà dữ liệu sẽ được gán nhãn trùng với tâm gần nhất.

**Thuật toán của K – Means được tóm tắt như sau**

* Khởi tạo ngẫu nhiên giá trị tâm cụm (centroid) , , …
* Lặp lại quá trình xác định các giá trị tâm cụm:
* Xác định nhãn cho từng điểm dữ liệu dựa trên khoảng cách tới từng tâm cụm
* Tính toán lại giá trị tâm cụm dựa trên các điểm dữ liệu nằm trong cụm

Tuy nhiên, thuật toán này lại tồn tại một nhược điểm cố hữu đó là phụ thuộc rất nhiều vào việc khởi tạo các giá trị tâm cụm (centroid). Nếu một tâm cụm được khởi tạo ngẫu nhiên với giá trị quá xa với tập dữ liệu thí rất có thể cụm đó sẽ không bao gồm bất cứ điểm dữ liệu nào. Ngược lại, nếu tồn tại nhiều hơn 1 tâm cụm được khởi tạo với giá trị quá gần nhau sẽ khiến cho việc phân cụm không đạt được độ chính xác mong muốn (phân cụm kém). Do đó, để cái thiện nhược điểm này, một thuật toán khởi tạo tâm cụm được sử dụng là K++.

**Thuật toán K++ được tóm tắt như sau**

* Khởi tạo giá trị tâm cụm đầu tiên bằng cách chọn ngẫu nhiên một giá trị của tập dữ liệu.
* Lặp lại các bước khởi tạo giá trị tâm cụm tiếp theo đến khi thu được tất cả các giá trị mong muốn
* Xác định khoảng cách từ mỗi điểm dữ liệu đến tâm cụm gần nhất và giá trị tâm cụm đã khởi tạo trước đó.
* Chọn điểm tâm cụm mới từ các điểm dữ liệu sao cho khoảng cách đến các tâm cụm trước đó là xa nhất.

Như vậy, thuật toán K++ đảm bảo việc các giá trị tâm cụm khởi tạo sẽ không nằm ngoài tập dữ liệu và hoàn toàn tách biệt nhau.

1. **Viết chương trình và ví dụ minh họa bằng Python**

Giả sử, khởi tạo ngẫu nhiên một tập dữ liệu 2D cho trước để minh họa cho cách thuật toán K++ hoạt động.

# creating data

mean\_01 = np.array([0.0, 0.0])

cov\_01 = np.array([[1, 0.3], [0.3, 1]])

dist\_01 = np.random.multivariate\_normal(mean\_01, cov\_01, 100)

mean\_02 = np.array([6.0, 7.0])

cov\_02 = np.array([[1.5, 0.3], [0.3, 1]])

dist\_02 = np.random.multivariate\_normal(mean\_02, cov\_02, 100)

mean\_03 = np.array([7.0, -5.0])

cov\_03 = np.array([[1.2, 0.5], [0.5, 1,3]])

dist\_03 = np.random.multivariate\_normal(mean\_03, cov\_01, 100)

mean\_04 = np.array([2.0, -7.0])

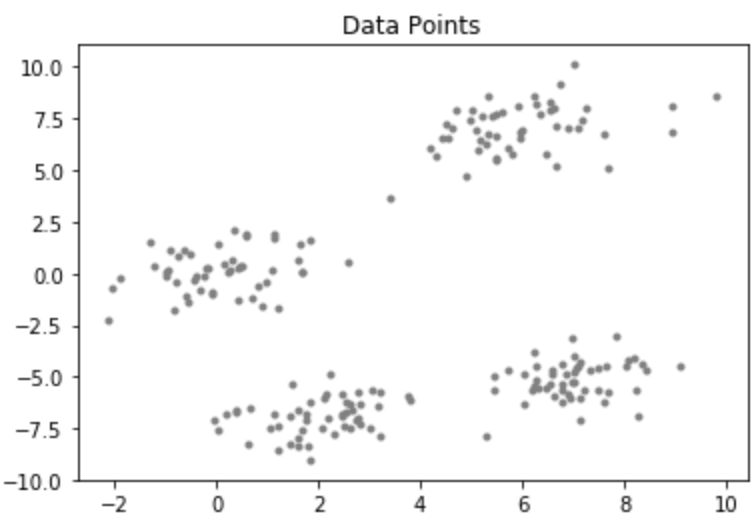
cov\_04 = np.array([[1.2, 0.5], [0.5, 1,3]])

dist\_04 = np.random.multivariate\_normal(mean\_04, cov\_01, 100)

data = np.vstack((dist\_01, dist\_02, dist\_03, dist\_04))

np.random.shuffle(data)

Để dễ dàng minh họa cách thuật toán K++ hoạt động, chúng ta sẽ khởi tạo ngẫu nhiên 4 cụm dữ liệu (gần như tách biệt) với 100 điểm dữ liệu mỗi cụm. Hình vẽ minh họa các điểm dữ liệu trong mặt phẳng 2D. Các điểm dữ liệu sẽ được tô màu xám (grey)



Mục tiêu của việc áp dụng giải thuật K++ cho bài toán minh họa này sẽ là tìm ra 4 tâm cụm (centroid) phù hợp cho giải thuật K – Means để phân cụm ở bước sau đó.

Bắt đầu giải thuật, khởi tạo giá trị tâm cụm đầu tiên bằng cách chọn ngẫu nhiên một điểm dữ liệu trên tập dữ liệu. Điểm tâm cụm vừa được khởi tạo sẽ được đánh dấu màu đỏ (red)

## initialize the centroids list and add a randomly selected data point to the list

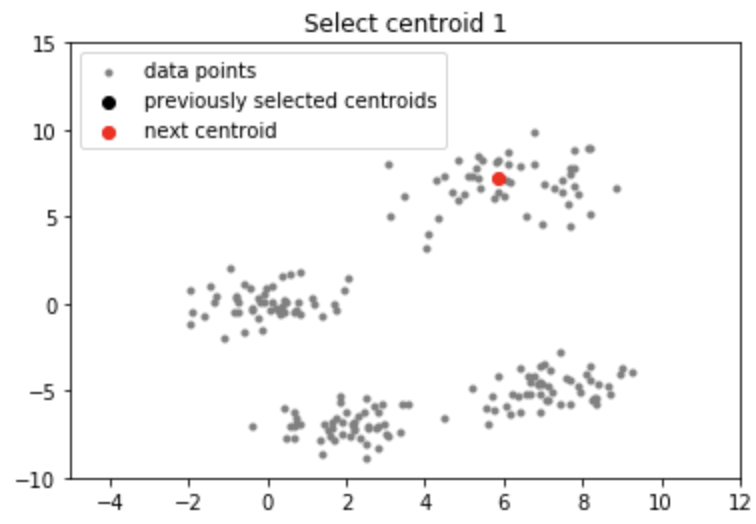
centroids = []

centroids.append(data[np.random.randint(

data.shape[0]), :])

plot(data, np.array(centroids))

**Kết quả của bước khởi tạo này**



Thực hiện vòng lặp để khởi tạo tiếp các điểm tâm cụm còn lại. Các điểm tâm cụm tiếp theo sẽ được đánh dấu bằng màu đen (black)

## compute remaining k - 1 centroids

for c\_id in range(k - 1):

## initialize a list to store distances of data

## points from nearest centroid

dist = []

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

point = data[i, :]

d = sys.maxsize

## compute distance of 'point' from each of the previously

## selected centroid and store the minimum distance

for j in range(len(centroids)):

temp\_dist = distance(point, centroids[j])

d = min(d, temp\_dist)

dist.append(d)

## select data point with maximum distance as our next centroid

dist = np.array(dist)

next\_centroid = data[np.argmax(dist), :]

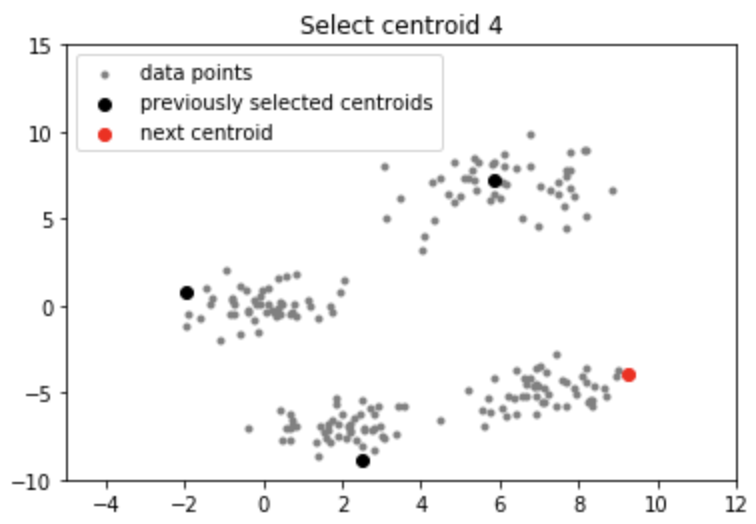
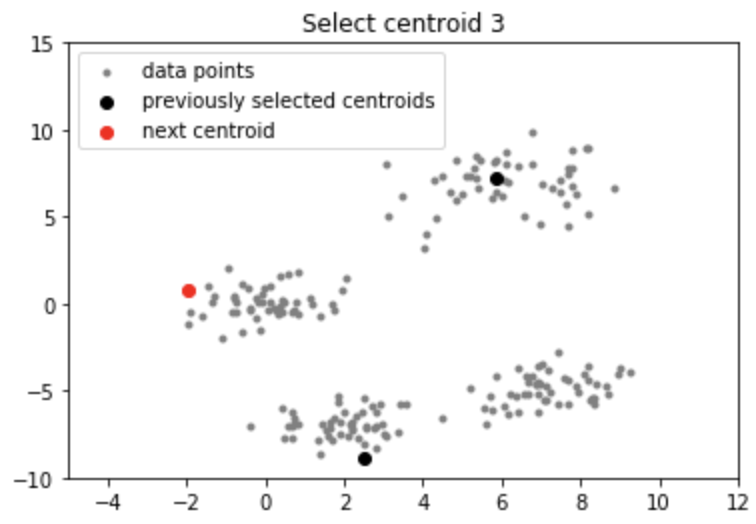
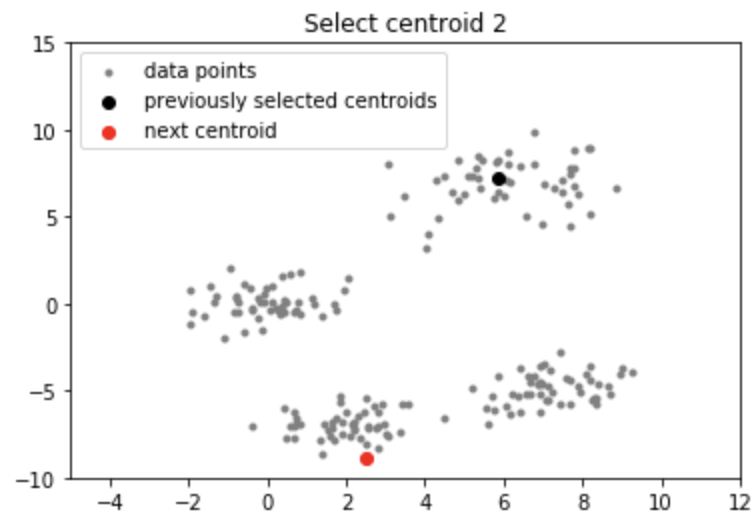
centroids.append(next\_centroid)

dist = []

plot(data, np.array(centroids))

return centroids

**Kết quả của quá trình này**



**Đoạn code chương trình Python hoàn chỉnh**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sys

# creating data

mean\_01 = np.array([0.0, 0.0])

cov\_01 = np.array([[1, 0.3], [0.3, 1]])

dist\_01 = np.random.multivariate\_normal(mean\_01, cov\_01, 100)

mean\_02 = np.array([6.0, 7.0])

cov\_02 = np.array([[1.5, 0.3], [0.3, 1]])

dist\_02 = np.random.multivariate\_normal(mean\_02, cov\_02, 100)

mean\_03 = np.array([7.0, -5.0])

cov\_03 = np.array([[1.2, 0.5], [0.5, 1,3]])

dist\_03 = np.random.multivariate\_normal(mean\_03, cov\_01, 100)

mean\_04 = np.array([2.0, -7.0])

cov\_04 = np.array([[1.2, 0.5], [0.5, 1,3]])

dist\_04 = np.random.multivariate\_normal(mean\_04, cov\_01, 100)

data = np.vstack((dist\_01, dist\_02, dist\_03, dist\_04))

np.random.shuffle(data)

# function to plot the selected centroids

def plot(data, centroids):

plt.scatter(data[:, 0], data[:, 1], marker = '.',

color = 'gray', label = 'data points')

plt.scatter(centroids[:-1, 0], centroids[:-1, 1],

color = 'black', label = 'previously selected centroids')

plt.scatter(centroids[-1, 0], centroids[-1, 1],

color = 'red', label = 'next centroid')

plt.title('Select % d th centroid'%(centroids.shape[0]))

plt.legend()

plt.xlim(-5, 12)

plt.ylim(-10, 15)

plt.show()

# function to compute euclidean distance

def distance(p1, p2):

return np.sum((p1 - p2)\*\*2)

# initialization algorithm

def initialize(data, k):

'''

initialized the centroids for K-means++

inputs:

data - numpy array of data points having shape (200, 2)

k - number of clusters

'''

## initialize the centroids list and add a randomly selected data point to the list

centroids = []

centroids.append(data[np.random.randint(

data.shape[0]), :])

plot(data, np.array(centroids))

## compute remaining k - 1 centroids

for c\_id in range(k - 1):

## initialize a list to store distances of data

## points from nearest centroid

dist = []

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

point = data[i, :]

d = sys.maxsize

## compute distance of 'point' from each of the previously

## selected centroid and store the minimum distance

for j in range(len(centroids)):

temp\_dist = distance(point, centroids[j])

d = min(d, temp\_dist)

dist.append(d)

## select data point with maximum distance as our next centroid

dist = np.array(dist)

next\_centroid = data[np.argmax(dist), :]

centroids.append(next\_centroid)

dist = []

plot(data, np.array(centroids))

return centroids

# call the initialize function to get the centroids

centroids = initialize(data, k = 4)