BÁO CÁO BÀI TẬP

THÀNH VIÊN:

B1812346 - Ôn Đình Khang

B1812393 - Cao Thanh Tuấn

Câu 1:

Bài toán tối ưu trong Support Vector Machine(SVM) chính là bài toán tìm đường phân chia sao cho margin là lớn nhất. Đây cũng là lý do vì sao SVM còn được gọi là Maximum Margin Classifier.

Các phần tử càng cách xa đường biên càng tốt(lề càng lớn càng tốt).

Mục tiêu của bài toán là tìm w và b để sao cho mỗi lớp dương hoặc âm nằm về hai phía của đường thẳng.

Dữ liệu dùng để huấn luyện là một ma trận có m hàng và n cột. Mỗi hàng là một phần tử dùng để huấn luyện gồm có n thuộc tính.

1. Véc tơ pháp tuyến w: là một vectơ có n phần tử tương ứng với n thuộc tính của tập dữ liệu huấn luyện.

2 . Độ lệch b : là số thực.

Từ 1 và 2 => bài toán trở thành tìm w =(w1,w2) và b

2/||w|| -> max (hay (hàm mục tiêu|w^2| **-> min**))Thành viên:

```
import numpy as np
import matplotlib.pyplot as plot
import quadprog as qp
X = np.matrix([[2, 2],
               [3, 1],
               [1, 1]])


Y = np.array([1, 1, -1])


G = np.matrix([[1.0, 0.0, 0.0],
              [0.0, 1.0, 0.0],
              [0.0, 0.0, 0.0000001]
               ])
a = np.array(([0.0, 0.0, 0.0]))

# Ma Tran C
C = np.matrix([
    [2.0, 2.0, -1.0],
    [3.0, 1.0, -1.0],
    [-1.0, -1.0, 1.0]
]).T

b = np.array([1.0, 1.0, 1.0])
# Giai bai toan
sol = qp.solve_qp(G, a, C, b)
#Ket qua
wb = sol[0]
```

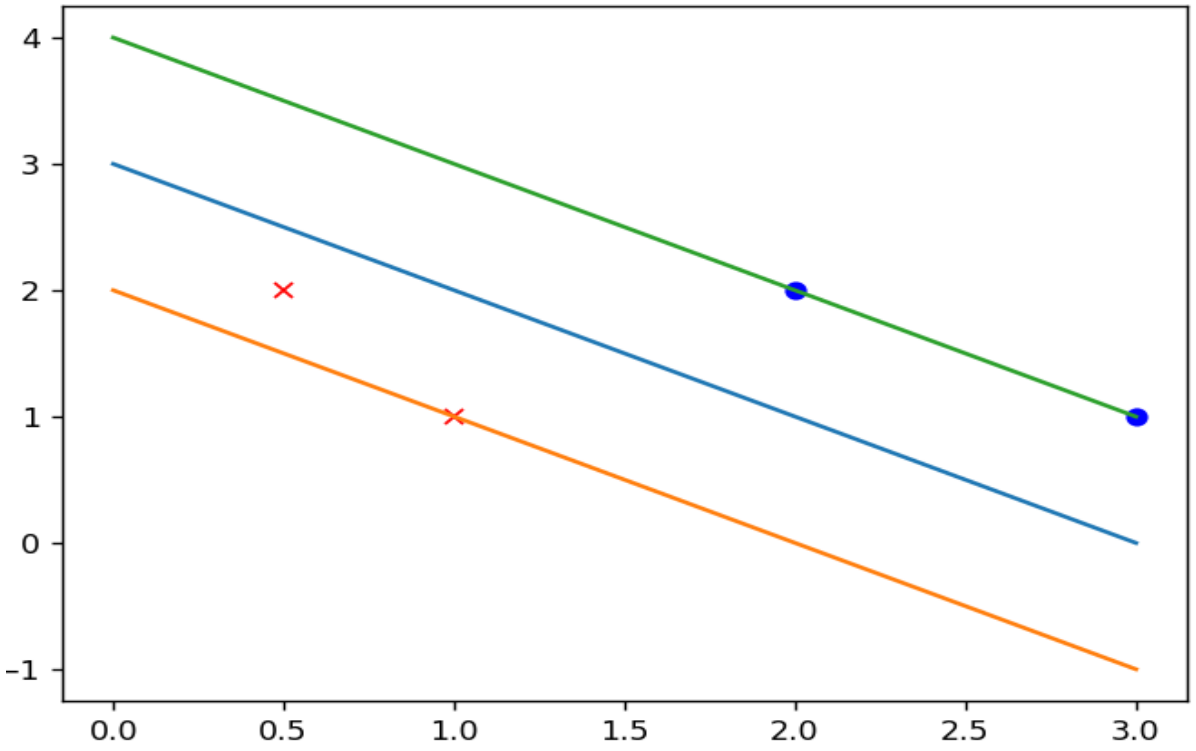
```
# Ve
plot.plot(
    X[0:2, 0], X[0:2, 1], 'bo'
)
plot.plot(
    X[2:3, 0], X[2:3, 1], 'rx'
)

x1 = np.array([0,3])
x2 = (wb[2] - wb[0] * x1) / wb[1]
x3 = (wb[2]-1 - wb[0] * x1) /wb[1]
x4 = (wb[2]+1 - wb[0] * x1) /wb[1]
plot.plot(x1,x2)
plot.plot(x1,x3)
plot.plot(x1,x4)
plot.show()
```

Hàm dự đoán dùng để dự đoán phần tử (0.5,2)

```
def dudoan(x, y, wb):
    kq = x * wb[0] + y * wb[1] - wb[2]
    print(kq)
    if kq > 0:
        print("lop duong")
        plot.plot(x, y, 'bo')
    elif kq < 0:
        print("lop am")
        plot.plot(x, y, 'rx')


dudoan(0.5, 2, wb)
plot.show()
```



Phần tử 0.5 , 2 sau khi được dự đoán

Bài tập 2

- Đầu tiên tạo file input.csv:

lab1_tuan3.py | input.csv

*.csv files are supported in other JetBrains IDEs — Try RubyMine

```
,x1,x2,y
0,3,2,1
1,3,3,1
2,3,4,1
3,3.5,2.5,1
4,1,3,-1
5,1,2,-1
6,1,1,-1
7,2,1,-1
8,3,1.5,1
9,1.5,2,-1
10,1,1,-1
11,2.5,3,1
12,2,3,1
11,2.5,2.5,1
12,1.5,1.5,-1
1 1.3,2,-1
14,1.7,2.3,-1
15,2,2.3,-1
16,2.5,1.3,-1
```

- Sau đó tiến hành tạo file python và import các thư viện cần thiết, tiếp đó đọc file csv đã tạo và gán các ma trận cho các cột tương ứng:

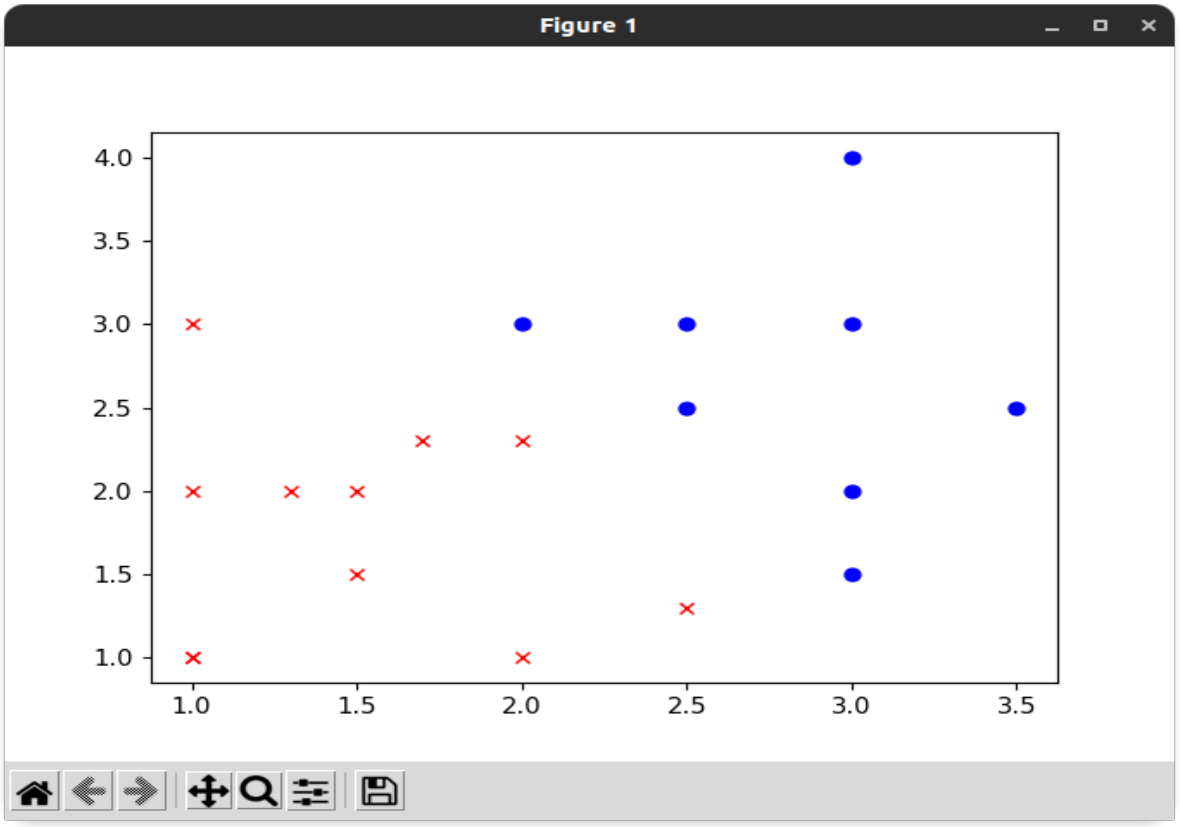
```
import numpy as np
import pandas as pd
import quadprog as qp
import matplotlib.pyplot as plt

data = pd.read_csv('input.csv', index_col=0)
# ma tran X, Y
X = data.iloc[:, 0:2]
Y = data.iloc[:, 2]
matrix_x = np.matrix(X)
matrix_y = np.matrix(Y)
```

- Tiến hành vẽ hình bằng hàm plot của thư viện matplotlib:
Chạy vòng for để lấy các giá trị nhãn Y, nếu nhãn là 1 thì vẽ biểu tượng đại diện là vòng tròn màu xanh, nếu bằng -1 thì vẽ dấu ‘x’ màu đỏ

```
# Ve hinh
for idx, value in enumerate(Y):
    if value == 1:
        plt.plot(
            matrix_x[idx, 0:1], matrix_x[idx, 1:2], 'bo'
        )
    else:
        plt.plot(
            matrix_x[idx, 0:1], matrix_x[idx, 1:2], 'rx'
        )
```

Kết quả sau khi vẽ:



Tìm ma trận C bằng cách:

- thêm vào 1 cột tập các giá trị -1
- sau đó nhân từng phần tử mỗi hàng với y

```
# them 1 vao cuoi ma tran x
matrix_x = np.insert(matrix_x, 2, -1, axis=1)
# nhan ma tran x voi y (quadprog)
for idx, item in enumerate(matrix_x):
    matrix_x[idx] = item.dot(matrix_y[0, idx])
# Ma tran C = -G (.T ma tran chuyen vi)
C = matrix_x.astype(np.float_)
C = C.T
```

Tạo ma trận b bằng cách:

- tạo một ma trận có số phần tử bằng với số lượng phần tử của ma trận x
- các giá trị phần tử = 1 (=-h với h=-1)

```
# b = -h (=1)
# tao array b voi so phan tu bang voi length cua matrix x
b = np.full(
    shape=len(matrix_x),
    fill_value=1,
    dtype=np.double
)
```

Tạo ma trận a và G:

```
# a = -q
a = np.array(([0.0, 0.0, 0.0]))
# Ma Tran G
G = np.matrix([[1.0, 0.0, 0.0],
               [0.0, 1.0, 0.0],
               [0.0, 0.0, 0.0000001]
               ])
```

Sau khi có đủ các ma trận gọi hàm solve_qp của thư viện quadprog để huấn luyện mô hình:

```
sol = qp.solve_qp(G, a, C, b)
# Ket qua
x1 = np.array([0, 4])
wb = sol[0]
print(wb)
# [ 4.28571429  2.85714286 16.14285714]
```

Kết quả đạt được:

w1 = 4.28
w2 = 2.85
b = 16.14

Phương trình đường thẳng có dạng:

4.28x1 + 2.85x2 - 16.14 = 0

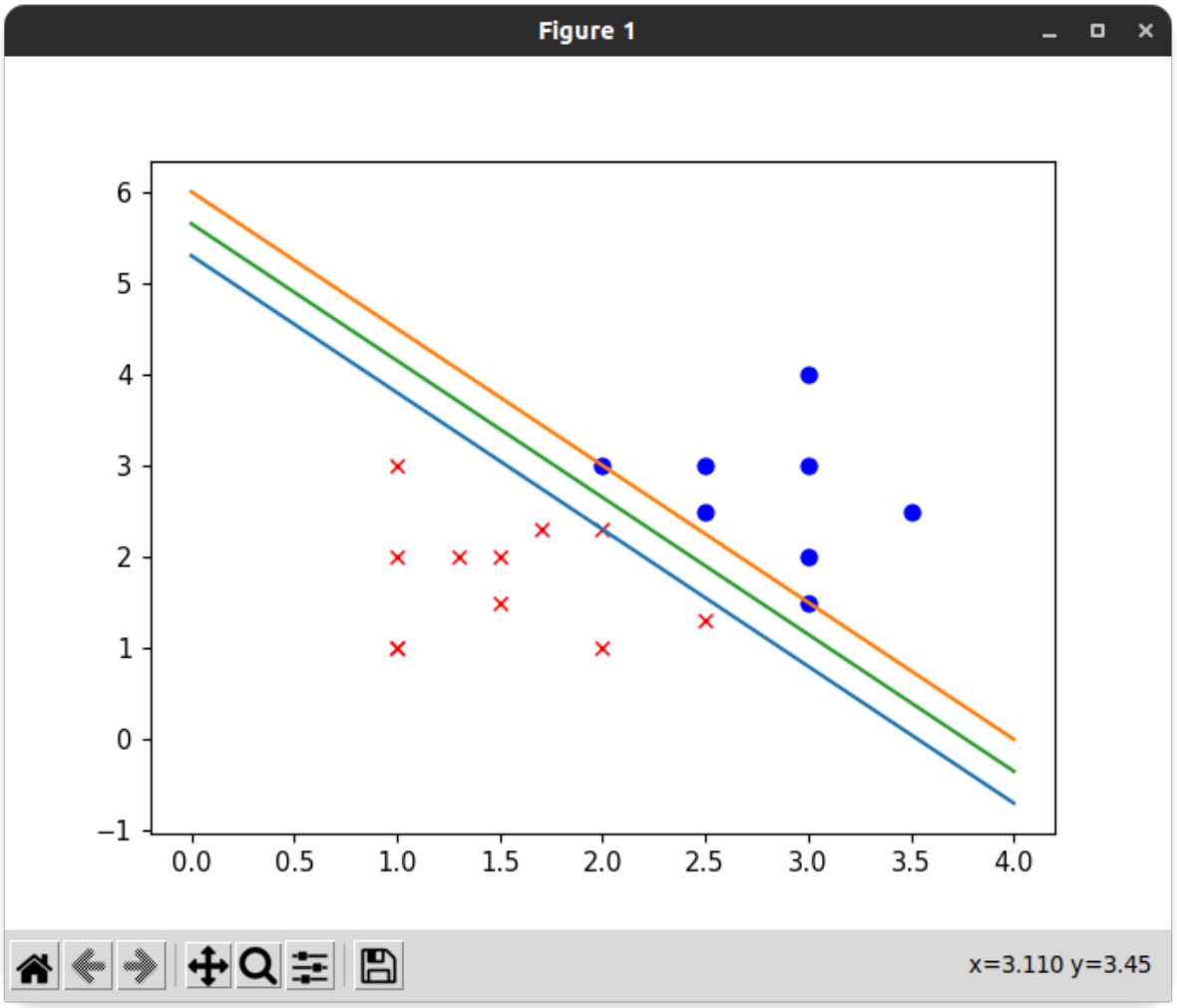
Phương trình đường thẳng của hai đường hỗ trợ:

4.28x1 + 2.85x2 - 16.14 = +- 1

Sau khi đã tính toán xong các giá trị tiến hành vẽ chúng lên đồ thị:

```
# x1 + x2 - b = 0
a1 = (wb[2] - wb[0] * x1) / wb[1]
# x1 + x2 - b = 1
a2 = ((wb[2] - 1) - wb[0] * x1) / wb[1]
# x1 + x2 - b = -1
a3 = ((wb[2] + 1) - wb[0] * x1) / wb[1]
plt.plot(x1, a2)
plt.plot(x1, a3)
plt.plot(x1, a1)
plt.show()
```

Kết quả đạt được:



Lấy ra từng hàng của dữ liệu huấn luyện nhân với w. Nếu có m phần tử thì có m ràng buộc.

Dạng chuẩn của bài toán quy hoạch toàn phương có dạng:

$(1/2)x^T Px + q^T x$

Gx<=h

Ax =b

P là ma trận vuông

Q là véc tơ

G,A là ma trận ràng buộc

H,b là véc tơ