BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC CẦN THƠ**

**KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**

**🙞 🕮 🙜**



**BÀI TẬP NHÓM**

**MÁY HỌC NÂNG CAO**

**ĐỀ TÀI**

**GiẢI THUẬT SVM**

**Giảng viên hướng dẫn:**

**Phạm Nguyên Khang**

**Sinh viên thực hiện:**

|  |  |
| --- | --- |
| Nguyễn Thành Hoàng Hải | B1812339 |
| Võ Ngọc Long | B1812282 |

Cần Thơ, 9/2021

# MỤC LỤC

[MỤC LỤC 1](#_Toc81834249)

[MÔ TẢ YÊU CẦU BÀI TOÁN 2](#_Toc81834250)

[I. YÊU CẦU 1 2](#_Toc81834251)

[II. YÊU CẦU 2 2](#_Toc81834252)

[BÁO CÁO KẾT QUẢ 3](#_Toc81834253)

[I. CÀI ĐẶT CÁC THƯ VIỆN CẦN THIẾT 3](#_Toc81834254)

[II. YÊU CẦU 1 3](#_Toc81834255)

[1. Đọc dữ liệu từ tập huấn luyện 3](#_Toc81834256)

[2. Viết hàm huấn luyện mô hình SVM 3](#_Toc81834257)

[2.1. Input và output 3](#_Toc81834258)

[2.2. Ý tưởng giải thuật: 3](#_Toc81834259)

[2.3. Cài đặt 5](#_Toc81834260)

[2.4. Chạy thử 5](#_Toc81834261)

[III. YÊU CẦU 2 7](#_Toc81834262)

[1. Chia tập dữ liệu thành 2 tập train và test 7](#_Toc81834263)

[2. Xây dựng hàm SVM cho K bộ phân lớp 7](#_Toc81834264)

[3. Kiểm tra và đánh giá bộ phân lớp 7](#_Toc81834265)

[KẾT QUẢ ĐẠT ĐƯỢC 10](#_Toc81834266)

[TÀI LIỆU THAM KHẢO 12](#_Toc81834267)

[PHỤ LỤC 13](#_Toc81834268)

# MÔ TẢ YÊU CẦU BÀI TOÁN

## YÊU CẦU 1

Thực hiện viết một chương trình bằng ngôn ngữ lập trình Python có thể đọc dữ liệu đầu vào là một tập tin được lưu dưới dạng, tập tin csv. Huấn luyện một mô hình SVM (Máy học Véc tơ hỗ trợ) cho tập dữ liệu được đọc. Kết quả nhận được là các tham số w1, w2, w3, …, wn và b.

## YÊU CẦU 2

Sử dụng chương trình được tạo ra từ yêu cầu 1, huấn luyện một bộ phân lớp để phân lớp dữ liệu có nhiều lớp.

* Gọi K là số lớp của dữ liệu (với K = 3, 4, ...)
* Chia ngẫu nhiên tập dữ liệu thành 2 tập: Train (80%) và Test (20%)
* Huấn luyện: dùng tập train để huấn luyện
* Lặp lại K lần (k chạy từ 1 đến K), mỗi lần lặp
* Cho lớp k là lớp dương (+1)
* Các lớp còn lại làm lớp âm (-1)
* Sử dụng chương trình ở yêu cầu 1 thực hiện huấn luyện để tìm w và b => lưu lại kết quả
* Sau khi kết thúc, ta có K bộ phân lớp, mỗi bộ phân lớp có các tham số (w, b) riêng của nó
* Kiểm tra và đánh giá bộ phân lớp
* Sử dụng tập test để kiểm tra
* Với mỗi phần tử x\_test của tập test, lần lượt cho nó đi K bộ phân lớp => K kết quả
* Sử dụng kết quả kết quả của K bộ phân lớp suy ra nhãn cho phần từ x\_test. Ví dụ: có 3 bộ phân lớp, kết quả lần lượt là: 1, 1, 2 thì kết quả là lớp 1

Dữ liệu được cho để huấn luyện mô hình: [UCI Machine Learning Repository: Iris Data Set](https://archive.ics.uci.edu/ml/datasets/Iris)

# BÁO CÁO KẾT QUẢ

## CÀI ĐẶT CÁC THƯ VIỆN CẦN THIẾT

Để tiến hành huấn luyện mô hình SVM với tập dữ liệu iris, trước tiên cần cài đặt những thư viện cần thiết của Python

pip install sklearn qpsolvers pandas

## YÊU CẦU 1

### Đọc dữ liệu từ tập huấn luyện

df = pd.read\_csv(

"iris.data",

names=[

"sepal length in cm",

"sepal width in cm",

"petal length in cm",

"petal width in cm",

"class"

]

)

### Viết hàm huấn luyện mô hình SVM

#### Input và output

Hàm huấn luyện mô hình SVM nhận input có kiểu dữ liệu như sau:

* X: numpy.ndarray (chứa dữ liệu các thuộc tính)
* Y: numpy.ndarray (mô tả phần tử thuộc dương hoặc âm)
* c: int (hằng số c trong giải thuật SVM không khả tách tuyến tính, có giá trị mặc định là 100)

Với kết quả trả về là 1 tuple chứa (w, b, z)

#### Ý tưởng giải thuật:

* Đưa bài toàn về dạng quy hoạch toàn phương và giải bằng thư viện qpsolvers.
* Bài toán quy hoạch toàn phương: tìm x sao cho
  + đạt giá trị nhỏ nhất
  + Với điều kiện:
* Từ x sẽ thu được giá trị của w, b, z. Ma trận của x

|  |
| --- |
| w1 |
| w2 |
| … |
| wn |
| b |
| z1 |
| z2 |
| … |
| zm |

* Xây dựng các ma trận của quy hoạch phương từ X và Y:
  + P:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 (w1) |  |  |  |  |  |  |  |  |
|  | 1 (w2) |  |  |  |  |  |  |  |
|  |  | … |  |  |  |  |  |  |
|  |  |  | 1 (wn) |  |  |  |  |  |
|  |  |  |  | 0 (b) |  |  |  |  |
|  |  |  |  |  | 0 (z1) |  |  |  |
|  |  |  |  |  |  | 0 (z2) |  |  |
|  |  |  |  |  |  |  | … |  |
|  |  |  |  |  |  |  |  | 0 (zm) |

* + q

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 (w1) | 0 (w2) | … | 0 (wn) | 0 (b) | c (z1) | c (z2) | … | c (zm) |

* + G

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -Y1 |  |  | 0 | dot | X1[1] | X2[1] | … | Xn[1] | Y1 | -1  (z1) |  |  | 0 |
|  | -Y2 |  |  | X1[2] | X2[2] | … | Xn[2] | Y2 |  | -1 (z2) |  |  |
|  |  | … |  | … | … | … | … | … |  |  | … |  |
| 0 |  |  | -Ym | X­1[m] | X­2[m] | … | Xn[m] | Ym | 0 |  |  | -1 (zm) |
| 0 | | | | | | | | | 0 | -1 (z1) |  |  | 0 |
|  | -1 (z2) |  |  |
|  |  | … |  |
| 0 |  |  | -1 (zm) |

* + h

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -1 (z1) | -1 (z2) | … | -1 (zm) | 0 (z1) | 0 (z2) | … | 0 (zm) |

* + A

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 (w1) | 0 (w2) | … | 0 (wn) | 0 (b) | 0 (z1) | 0 (z2) | … | 0 (zm) |

* + b

|  |
| --- |
| 0 |

#### Cài đặt

def SVM(X: np.ndarray, Y: np.ndarray, c: int = 100) -> tuple:

m, n = X.shape

P = np.diag(np.hstack((np.ones(n), np.full(m+1, 0.000001))))

q = np.hstack((np.zeros(n+1), np.full(m, c)))

G = np.vstack((np.hstack((np.diag(-Y).dot(X),

Y.reshape(-1, 1),

-np.identity(m))),

np.hstack((np.zeros\_like(X),

np.zeros\_like(Y.reshape(-1, 1)),

-np.identity(m)))))

h = np.hstack((np.full(m, -1), np.zeros(m)))

A = np.zeros(n+1+m)

b = np.zeros(1)

ans = solve\_qp(P, q, G, h, A, b)

return np.split(ans, [n, n+1])

#### Chạy thử

Chọn Iris-setosa để phân lớp

from sklearn.model\_selection import train\_test\_split

from qpsolvers import solve\_qp

import numpy as np

import pandas as pd

def SVM(X: np.ndarray, Y: np.ndarray, c: int = 100) -> tuple:

m, n = X.shape

P = np.diag(np.hstack((np.ones(n), np.full(m+1, 0.000001))))

#print(P)

q = np.hstack((np.zeros(n+1), np.full(m, c)))

#print(q)

G = np.vstack((np.hstack((np.diag(-Y).dot(X),

Y.reshape(-1, 1),

-np.identity(m))),

np.hstack((np.zeros\_like(X),

np.zeros\_like(Y.reshape(-1, 1)),

-np.identity(m)))))

#print(G)

h = np.hstack((np.full(m, -1), np.zeros(m)))

#print(h)

A = np.zeros(n+1+m)

#print(A)

b = np.zeros(1)

#print(b)

ans = solve\_qp(P, q, G, h, A, b)

#print(ans)

return np.split(ans, [n, n+1])

def main():

df = pd.read\_csv(

"iris.data",

names=[

"sepal length in cm",

"sepal width in cm",

"petal length in cm",

"petal width in cm",

"class"

]

)

X = df.drop(columns="class")

Y = df["class"]

Y[Y != "Iris-setosa"] = -1

Y[Y == "Iris-setosa"] = 1

w, b, z = SVM(X.to\_numpy(), Y.to\_numpy(np.float64))

print(w)

print(b)

print(z)

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

main()

Thu được kết quả w, b, z lần lượt là

[-0.04603094 0.52172083 -1.00316549 -0.46418034]

[-1.45055056]

[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. 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. 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.]

## YÊU CẦU 2

### Chia tập dữ liệu thành 2 tập train và test

Sử dụng train\_test\_split của thư viện sklearn để phân tập dữ liệu ra thành 80% train và 20% test

df = pd.read\_csv(

"iris.data",

names=[

"sepal length in cm",

"sepal width in cm",

"petal length in cm",

"petal width in cm",

"class"

]

)

X = df.drop(columns="class")

Y = df["class"]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X,

Y,

test\_size=0.2,

random\_state=7777777

)

### Xây dựng hàm SVM cho K bộ phân lớp

def SVM\_fit(X\_train: pd.DataFrame, Y\_train: pd.Series) -> tuple:

W = []

B = []

Z = []

labels = Y\_train.unique()

for label in labels:

Y\_train\_tmp = Y\_train.copy()

Y\_train\_tmp[Y\_train\_tmp != label] = -1

Y\_train\_tmp[Y\_train\_tmp == label] = 1

w, b, z = SVM(X\_train.to\_numpy(np.float64), Y\_train\_tmp.to\_numpy(np.float64))

W.append(w)

B.append(b)

Z.append(z)

return W, B, Z

### Kiểm tra và đánh giá bộ phân lớp

* Từ K bộ phân lớp ta viết hàm dự đoán trả về kết quả của các bộ phân lớp

def SVM\_predict(X\_test: pd.DataFrame, W: list, B: list) -> np.ndarray:

res = []

for cnt in range(len(W)):

res.append(X\_test.to\_numpy().dot(np.vstack(W[cnt]))-B[cnt])

return np.hstack(res)

* Kết quả dự đoán từ X\_test thu được trên tập dữ liệu iris

[[-31.49773756 1.60223064 -2.6 ]

[ -3.9638009 -2.38289094 -0.75 ]

[-27.88235294 1.18586937 -0.45 ]

[ -4.68325792 -2.04088521 -1.25 ]

[-28.57013575 1.11152245 -0.85 ]

[ 6.53846154 -3.33456631 -0.2 ]

[-31.92307692 1.55761916 -1.3 ]

[-29.21719457 1.36430972 -1.4 ]

[ -5.49321267 -2.56133166 1.9 ]

[ -3.05882353 -2.39032687 -0.7 ]

[ 8.12669683 -3.31969096 -3.5 ]

[ 9.89140271 -3.55760843 -3.55 ]

[ 8.60180995 -4.29365902 -3.3 ]

[ 1.34841629 -2.63567886 -2.25 ]

[-30.96832579 1.59479505 -2.5 ]

[-26.75565611 0.88103469 1.45 ]

[ -0.3800905 -3.02972995 0.1 ]

[ 3.76470588 -3.23047221 -1.45 ]

[ 1.31221719 -3.28995627 2. ]

[ -6.01809955 -1.97397216 -0.3 ]

[ -6.03167421 -1.85501273 -1.05 ]

[-26.35294118 1.17100373 -2.95 ]

[ -3.09502262 -2.13753716 -2.65 ]

[ -2.4479638 -2.8066854 1.85 ]

[-31.05882353 1.52788269 -3.15 ]

[ -4.84615385 -2.13010643 0.05 ]

[ 4.2081448 -3.40890876 -1.7 ]

[ 1.87782805 -3.1338206 0.1 ]

[-30.13574661 1.29739872 -2.05 ]

[ -5.28506787 -2.05575847 0.6 ]]

* Lấy nhãn bằng cách chọn phần tử có vị trí mang giá trị lớn nhất trong K bộ phân lớp

labels = Y\_train.unique()

res = SVM\_predict(X\_test, W, B)

print(res)

Y\_pred = labels[res.argmax(1)]

* Thu được Y\_pred có giá trị như sau

['Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor'

'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'

'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'

'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'

'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor'

'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor'

'Iris-setosa' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'

'Iris-setosa' 'Iris-versicolor']

* Thống kê và so sánh với Y\_test

stats = pd.DataFrame({

"Y\_test": Y\_test,

"Y\_pred": Y\_pred,

"Verdict": Y\_test == Y\_pred,

})

* Kết quả của stats

Y\_test Y\_pred Verdict

4 Iris-setosa Iris-setosa True

91 Iris-versicolor Iris-versicolor True

8 Iris-setosa Iris-setosa True

61 Iris-versicolor Iris-versicolor True

30 Iris-setosa Iris-setosa True

113 Iris-virginica Iris-virginica True

35 Iris-setosa Iris-setosa True

47 Iris-setosa Iris-setosa True

87 Iris-versicolor Iris-versicolor True

78 Iris-versicolor Iris-versicolor True

148 Iris-virginica Iris-virginica True

136 Iris-virginica Iris-virginica True

117 Iris-virginica Iris-virginica True

70 Iris-versicolor Iris-virginica False

40 Iris-setosa Iris-setosa True

41 Iris-setosa Iris-versicolor False

133 Iris-virginica Iris-versicolor False

147 Iris-virginica Iris-virginica True

119 Iris-virginica Iris-versicolor False

99 Iris-versicolor Iris-versicolor True

88 Iris-versicolor Iris-versicolor True

43 Iris-setosa Iris-setosa True

85 Iris-versicolor Iris-setosa False

68 Iris-versicolor Iris-versicolor True

46 Iris-setosa Iris-setosa True

94 Iris-versicolor Iris-versicolor True

139 Iris-virginica Iris-virginica True

83 Iris-versicolor Iris-virginica False

31 Iris-setosa Iris-setosa True

89 Iris-versicolor Iris-versicolor True

* Hiển thị độ chính xác của mô hình

print(stats["Verdict"].value\_counts(True))

* Thu được độ chính xác của mô hình SVM vừa xây dựng là

True 0.8

False 0.2

Name: Verdict, dtype: float64

# KẾT QUẢ ĐẠT ĐƯỢC

Xây dựng được K mô hình SVM trên K lớp để dự đoán cho tập dữ liệu iris với độ chính xác 80% được thể hiển ở biểu đồ sau:

# TÀI LIỆU THAM KHẢO

<https://scikit-learn.org/stable/modules/classes.html>

<https://drive.google.com/drive/folders/11xfIPVHbB5RHyXvGQYqNsQVYVG9JprfZ?usp=sharing>

# PHỤ LỤC

## CODE YÊU CẦU 1

from sklearn.model\_selection import train\_test\_split

from qpsolvers import solve\_qp

import numpy as np

import pandas as pd

def SVM(X: np.ndarray, Y: np.ndarray, c: int = 100) -> tuple:

m, n = X.shape

P = np.diag(np.hstack((np.ones(n), np.full(m+1, 0.000001))))

#print(P)

q = np.hstack((np.zeros(n+1), np.full(m, c)))

#print(q)

G = np.vstack((np.hstack((np.diag(-Y).dot(X),

Y.reshape(-1, 1),

-np.identity(m))),

np.hstack((np.zeros\_like(X),

np.zeros\_like(Y.reshape(-1, 1)),

-np.identity(m)))))

#print(G)

h = np.hstack((np.full(m, -1), np.zeros(m)))

#print(h)

A = np.zeros(n+1+m)

#print(A)

b = np.zeros(1)

#print(b)

ans = solve\_qp(P, q, G, h, A, b)

#print(ans)

return np.split(ans, [n, n+1])

def main():

df = pd.read\_csv(

"iris.data",

names=[

"sepal length in cm",

"sepal width in cm",

"petal length in cm",

"petal width in cm",

"class"

]

)

X = df.drop(columns="class")

Y = df["class"]

Y[Y != "Iris-setosa"] = -1

Y[Y == "Iris-setosa"] = 1

w, b, z = SVM(X.to\_numpy(), Y.to\_numpy(np.float64))

print(w)

print(b)

print(z)

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

main()

## CODE YÊU CẦU 2

from sklearn.model\_selection import train\_test\_split

from qpsolvers import solve\_qp

import numpy as np

import pandas as pd

def SVM(X: np.ndarray, Y: np.ndarray, c: int = 100) -> tuple:

m, n = X.shape

P = np.diag(np.hstack((np.ones(n), np.full(m+1, 0.000001))))

q = np.hstack((np.zeros(n+1), np.full(m, c)))

G = np.vstack((np.hstack((np.diag(-Y).dot(X),

Y.reshape(-1, 1),

-np.identity(m))),

np.hstack((np.zeros\_like(X),

np.zeros\_like(Y.reshape(-1, 1)),

-np.identity(m)))))

h = np.hstack((np.full(m, -1), np.zeros(m)))

A = np.zeros(n+1+m)

b = np.zeros(1)

ans = solve\_qp(P, q, G, h, A, b)

return np.split(ans, [n, n+1])

def SVM\_fit(X\_train: pd.DataFrame, Y\_train: pd.Series) -> tuple:

W = []

B = []

Z = []

labels = Y\_train.unique()

for label in labels:

Y\_train\_tmp = Y\_train.copy()

Y\_train\_tmp[Y\_train\_tmp != label] = -1

Y\_train\_tmp[Y\_train\_tmp == label] = 1

w, b, z = SVM(X\_train.to\_numpy(np.float64), Y\_train\_tmp.to\_numpy(np.float64))

W.append(w)

B.append(b)

Z.append(z)

return W, B, Z

def SVM\_predict(X\_test: pd.DataFrame, W: list, B: list) -> np.ndarray:

res = []

for cnt in range(len(W)):

res.append(X\_test.to\_numpy().dot(np.vstack(W[cnt]))-B[cnt])

return np.hstack(res)

def main():

df = pd.read\_csv(

"iris.data",

names=[

"sepal length in cm",

"sepal width in cm",

"petal length in cm",

"petal width in cm",

"class"

]

)

X = df.drop(columns="class")

Y = df["class"]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X,

Y,

test\_size=0.2,

random\_state=7777777

)

W, B, Z = SVM\_fit(X\_train, Y\_train)

labels = Y\_train.unique()

res = SVM\_predict(X\_test, W, B)

print(res)

Y\_pred = labels[res.argmax(1)]

stats = pd.DataFrame({

"Y\_test": Y\_test,

"Y\_pred": Y\_pred,

"Verdict": Y\_test == Y\_pred,

})

print(stats)

print(stats["Verdict"].value\_counts(True))

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

main()