# BÀI THỰC HÀNH PHẦN HARD MARGIN SUPPORT VECTOR MACHINE

**Ví dụ 1**: Chúng ta bắt đầu với một ví dụ trên dữ liệu nhân tạo. Trước hết ta tạo một bộ dữ liệu có 200 mẫu (N=200) phân đều vào 02 lớp – mỗi lớp 100 phần tử. Mỗi mẫu có 2 chiều (d=2). Tiếp theo chúng ta xây dựng các phương thức cần thiết cho phương pháp Hard Margin Vector Machine và tiến hành thực hiện phân lớp trên bộ dữ liệu nói trên.

**a) Tự xây dựng các bước**: Ta sẽ dung phương pháp giải bài toán Lagrange đối ngẫu g(λ) để tìm λ, sau đó thay vào tính w và w0 (bám theo công thức được xây dựng trong phần lý thuyết)

Trước hết ta gọi các thư viện cần thiết và tạo dữ liệu – Dữ liệu training sẽ là (X, y):

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(10)
means = [[2, 2], [4, 2]]
cov = [[.3, .2], [.2, .3]]
N = 100
X0 = np.random.multivariate_normal(means[0], cov, N) # class 1
X1 = np.random.multivariate_normal(means[1], cov, N) # class -1
X = np.concatenate((X0.T, X1.T), axis = 1) # all data
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1) # labels
```

Phần hiển thị dữ liệu trên hình vẽ để có quan sát trực quan.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(10, 10))
ani = plt.cla()
#plot points
ani = plt.plot(X0.T[0, :], X0.T[1, :], 'b^', markersize = 5, alpha = .8)
ani = plt.plot(X1.T[0, :], X1.T[1, :], 'ro', markersize = 5, alpha = .8)
ani = plt.axis([0 , 6, 0, 6])
plt.show()
```

Tiếp theo chúng ta giải bài toán tối ưu - g(λ) → min để tính λ trước bằng thư viện CVXOPT. Trong trường hợp chưa cài đặt thư viện này, cần mở lại (bỏ #) ở dòng đầu tiên để cài đặt thư viện cvxopt.

Chúng ta tìm hiểu thêm một số chức năng – phương thức cần thiết trong thư viện này:

- Lớp chứa các phương thức giải các bài toán tối ưu lồi là **solvers** (trong thư viện cvxopt), chúng ta cần import để sử dụng
- Phương thức giải bài toán quy hoạch toàn phương là **qp** (viết tắt của Quadratic Programming), thuộc lớp solvers. Cách gọi phương thức và tên của các tham đối được khai báo theo thứ tự như sau:

<tên đối tượng nhận kết quả> = solvers.qp(P, q, G, h, A, b)

Phương thức qp giải (theo biến x) bài toán quy hoạch toàn phương sau:

$$\begin{aligned} \min_x \quad & \tfrac{1}{2}x^\top P x + q^\top x \\ \text{subject to} \quad & Gx \preceq h \\ & Ax = b \end{aligned}$$

ở đây ký hiệu $\preceq$ dùng để biểu thị so sánh giữa các vector kiểu element-wise (so từng tọa độ). So sánh với ký hiệu trong bài giảng lý thuyết để biết tham đối nào đóng vai trò gì. Để lấy giá trị nghiệm tối ưu λ, ta lấy trường "*x*" từ đối tượng nhận kết quả: <tên đối tượng nhận kết quả>['*x*'].

```
#!pip install cvxopt
from cvxopt import matrix, solvers
```

```
# build P ~ K
V = np.concatenate((X0.T, -X1.T), axis = 1)
P = matrix(V.T.dot(V)) # P ~ K in slide see definition of V, K near eq (8)
q = matrix(-np.ones((2*N, 1))) # all-one vector
# build A, b, G, h
G = matrix(-np.eye(2*N)) # for all lambda_n >= 0! note that we solve -g(lambda) -> min
h = matrix(np.zeros((2*N, 1)))
A = matrix(y) # the equality constrain is actually y^T lambda = 0
b = matrix(np.zeros((1, 1)))
solvers.options['show_progress'] = False
sol = solvers.qp(P, q, G, h, A, b)
l = np.array(sol['x']) # lambda
print('lambda = ')
print(l.T)
```

Do các giá trị λ giải bằng thư viện CVXOPT được tính theo phương pháp lặp, nên hầu hết chúng không có giá trị 0 tuyệt đối mà chỉ là rất nhỏ (cỡ $10^{-9}$). Vì vậy chúng ta sẽ cần loại bỏ các giá trị λ quá nhỏ trong kết quả (thực chất đó là 0 nhưng phương pháp giải lặp có sai số tính toán).

Trong đoạn chương trình dưới đây ta lọc các giá trị λ nhỏ hơn $10^{-6}$ sau đó tính w, w0 theo công thức lý thuyết.

```
epsilon = 1e-6 # just a small number, greater than 1e-9, to filter values of lambda
S = np.where(l > epsilon)[0]
VS = V[:, S]
XS = X[:, S]
yS = y[:, S]
lS = l[S]
# calculate w and b
w = VS.dot(lS)
w0 = np.mean(yS.T - w.T.dot(XS))
print('W = ', w.T)
print('W0 = ', w0)
```

Cuối cùng, chúng ta vẽ kết quả để quan sát trực quan:

```
from matplotlib.backends.backend_pdf import PdfPages
with PdfPages('D:\\svm4hardmargin.pdf') as pdf:
    # draw
    # plot points
    fig, ax = plt.subplots()
    x1 = np.arange(-10, 10, 0.1)
    y1 = -w[0, 0]/w[1, 0]*x1 - w0/w[1, 0]
    y2 = -w[0, 0]/w[1, 0]*x1 - (w0-1)/w[1, 0]
    y3 = -w[0, 0]/w[1, 0]*x1 - (w0+1)/w[1, 0]
    plt.plot(x1, y1, 'k', linewidth = 3)
    plt.plot(x1, y2, 'k')
    plt.plot(x1, y3, 'k')
    y4 = 10*x1
    plt.plot(x1, y1, 'k')
    plt.fill_between(x1, y1, color='red', alpha=0.1)
    plt.fill_between(x1, y1, y4, color = 'blue', alpha = 0.1)
    plt.plot(X0[:, 0], X0[:, 1], 'bs', markersize = 8, alpha = .8)
    plt.plot(X1[:, 0], X1[:, 1], 'ro', markersize = 8, alpha = .8)
    plt.axis('equal')
    plt.ylim(0, 3)
    plt.xlim(2, 4)
    # hide tikcs
    cur_axes = plt.gca()
    cur_axes.axes.get_xaxis().set_ticks([])
    cur_axes.axes.get_yaxis().set_ticks([])
    # add circles around support vectors
    for m in S:
        circle = plt.Circle((X[0, m], X[1, m] ), 0.1, color='k', fill = False)
        ax.add_artist(circle)
```

```
    plt.xlabel('$x_1$', fontsize = 20)
    plt.ylabel('$x_2$', fontsize = 20)
        # plt.savefig('svm4.png', bbox_inches='tight', dpi = 300)
        # pdf.savefig()
    plt.show()
```

**Sử dụng thư viện**: Cũng với dữ liệu như trên, nhưng trong phần này chúng ta sử dụng thư viện sklearn. Trước hết chúng ta cần lấy đoạn chương trình tạo dữ liệu ở phần a) để có dữ liệu. Sau đó đoạn chương trình gọi thư viện và thực hiện thuật toán SVM như sau

```
from sklearn.svm import SVC
# Copy and put code for generate data here
y1 = y.reshape((2*N,))
X1 = X.T # each sample is one row
clf = SVC(kernel = 'linear', C = 1e5) # just a big number
# if C is small, method will be "SoftMagin SVM",
# if C is large enough, method is near to hard margin
clf.fit(X1, y1)
w = clf.coef_
w0 = clf.intercept_
print('w = ', w)
print('W0 = ', w0)
```

Hãy so sánh bộ trọng số tối ưu (kết quả) của hai phương pháp.

**Bài tập thực hành 1:** Xem lại các bài ví dụ phần trước, sử dụng bộ trọng số W đã tính được và dự đoán các phân lớp y ứng với dữ liệu X trên chính tập dữ liệu ngẫu nhiên vừa tạo. Đưa ra độ chính xác Accuracy và ma trận nhầm lẫn Confusion Matrix.

**Bài tập thực hành 2**: Hãy sửa phần đọc dữ liệu trong Ví dụ 1 ở trên, sau đó áp dụng cho phần phân loại phân loại bệnh nhân ung thư vú của Đại học Wisconsin–Madison, Hoa Kỳ. Dữ liệu có sẵn trong thư viện sklearn.

Dữ liệu có 569 bản ghi (mẫu), với 30 thuộc tính. Bệnh nhân được chia làm hai loại: u lành tính (B – Begnin)

có 357 mẫu và u ác tính (M – Malignant) có 212 mẫu.

Đoạn chương trình thực hiện việc đọc dữ liệu như dưới đây:

```
from sklearn import datasets
cancer_data = datasets.load_breast_cancer()
# show to test record 5th
print(cancer_data.data[5])
print(cancer_data.data.shape)
#target set
print(cancer_data.target)
from sklearn.model_selection import train_test_split
cancer_data = datasets.load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer_data.data, cancer_data.target,
test_size=0.3,random_state=109)
```

Tiếp theo, hãy sử dụng các phương pháp SVM đã có để áp dụng cho dữ liệu này.

**Ví dụ 2 (Bài tập tự giải)**: Trong bài tập này chúng ta sử dụng dữ liệu sóng thủy âm Sonar. Tệp dữ liệu có 60 cột ứng với 60 thuộc tính (trường) không có tiêu đề, là tham số của các mẫu sóng âm phản hồi; cột thứ 61 là đầu ra phân loại (y), với ký tự "R" nghĩa là Rock; ký tự "M" nghĩa là Mine (vật thể kim loại hình trụ). Toàn bộ tệp có 208 bản ghi. Thông tin thêm về dữ liệu có thể tìm hiểu tại link https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks) hoặc có thể lấy trong tệp Sonar.all-data.csv đính kèm.

Tham khảo lại phần hướng dẫn cho mô hình perceptron để nắm các thao tác đọc dữ liệu, định dạng lại dữ liệu (chuyển M thành class 1 và chuyển R thành class -1).

Đọc dữ liệu và sử dụng mô hình SVM với hard margin để phân loại cho dữ liệu này