BÀI THỰC HÀNH PHẦN SOFT MARGIN SUPPORT VECTOR MACHINE

Ví dụ 1: Chúng ta vẫn bắt đầu với một ví dụ trên dữ liệu nhân tạo, tương tự như phần HARD MARGIN. 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 Soft Margin Vector Machine và tiến hành thực hiện phân lớp trên bộ dữ liệu nói trên.

Cách 1. Tự xây dựng các bước giải bài toán ràng buộc: Phương pháp giải bài toán Lagrange đối ngẫu g(λ) để tìm λ, sau đó thay vào tính w, w_0 (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):

```
# generate data
# list of points
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(21)
from matplotlib.backends.backend_pdf import PdfPages


means = [[2, 2], [4, 1]]
cov = [[.3, .2], [.2, .3]]
N = 10
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X1[-1, :] = [2.7, 2]
X = np.concatenate((X0.T, X1.T), axis = 1)
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1)
```

Phần hiển thị dữ liệu trên hình vẽ để có quan sát trực quan, các bạn tham khảo lại phần Hard Margin và tự xây dựng.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(5, 5))


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([-1 , 6, -1, 6])
plt.show()
```

Tiếp theo chúng ta giải bài toán tối ưu - g(λ) → min để tính λ trước bằng thư viện CVXOPT. Ở đây chúng ta thực hiện ương tự như việc giải bài toán Hard Margin SVM, chỉ khác rằng ta có thêm ràng buộc về chặn trên của các nhân thử Lagrange ($0 \leq \lambda \leq C$), Ở đây ta chọn C=100:

```
#!pip install cvxopt
from cvxopt import matrix, solvers
C = 100
# build K
V = np.concatenate((X0.T, -X1.T), axis = 1)
K = matrix(V.T.dot(V))

p = matrix(-np.ones((2*N, 1)))
# build A, b, G, h
G = matrix(np.vstack((-np.eye(2*N), np.eye(2*N))))

h = matrix(np.vstack((np.zeros((2*N, 1)), C*np.ones((2*N, 1)))))
A = matrix(y.reshape((-1, 2*N)))
b = matrix(np.zeros((1, 1)))
solvers.options['show_progress'] = False
sol = solvers.qp(K, p, G, h, A, b)

l = np.array(sol['x'])
print('lambda = \n', l.T)
```

Tương tự phần Hard Margin, trong các thành phần của lambda tìm được, có rất nhiều thành phần nhỏ tới 1e-6 hay 1e-7. Đây chính là các lambda_i = 0. Có rất nhiều phần tử xấp xỉ 9.99e+01, đây chính là các lambda_i bằng với C = 100, tương ứng với các support vectors không nằm trên margins, các sai số nhỏ xảy ra do tính toán. Các giá trị còn lại nằm giữa 0 và 100 là các giá trị tương ứng với các điểm nằm chính xác trên hai margins.

Tiếp theo, ta cần tính w và b theo công thức. Trước đó ta cần tìm tập hợp các điểm support và những điểm nằm trên margins.Trong đoạn chương trình dưới đây ta lọc các giá trị nhỏ hơn 10^{-6} sau đó tính w, w_0 theo công thức lý thuyết.

```
S = np.where(l > 1e-5)[0] # support set
S2 = np.where(l < .999*C)[0]

M = [val for val in S if val in S2] # intersection of two lists

XT = X.T # we need each column to be one data point in this alg
VS = V[:, S]
lS = l[S]
yM = y[M]
```

```
XM = XT[:, M]


w_dual = VS.dot(lS).reshape(-1, 1)
b_dual = np.mean(yM.T - w_dual.T.dot(XM))
print(w_dual.T, b_dual)
```

Cuối cùng, chúng ta vẽ kết quả để quan sát trực quan. Phần này các bạn cần tham khảo code của bài Hard Margin (lưu ý tên biến tham số trong bài này là w_dual)

Cách 2. Xây dựng phương pháp giải bài toán tối ưu không ràng buộc: Trong phương pháp này, chúng ta cần tính gradient của hàm mất mát. Để chắc chắn công thức tính gradient trong lý thuyết là đúng, chúng ta cần kiểm chứng này bằng cách so sánh với numerical gradient.

Chú ý rằng trong phương pháp này, ta cần dùng tham số lam = 1/C (xem lại công thức lý thuyết).

```
X0_bar = np.vstack((X0.T, np.ones((1, N)))) # extended data
X1_bar = np.vstack((X1.T, np.ones((1, N)))) # extended data


Z = np.hstack((X0_bar, - X1_bar)) # as in (22)
lam = 1./C


def cost(w):
    u = w.T.dot(Z) # as in (23)
    return (np.sum(np.maximum(0, 1 - u)) + \
        .5*lam*np.sum(w*w)) - .5*lam*w[-1]*w[-1] # no bias


def grad(w):
    u = w.T.dot(Z) # as in (23)
    H = np.where(u < 1)[1]
    ZS = Z[:, H]
    g = (-np.sum(ZS, axis = 1, keepdims = True) + lam*w)
    g[-1] -= lam*w[-1] # no weight decay on bias
    return g


eps = 1e-6
def num_grad(w):
    g = np.zeros_like(w)
    for i in range(len(w)):
        wp = w.copy()
        wm = w.copy()
        wp[i] += eps
```

```
        wm[i] -= eps
        g[i] = (cost(wp) - cost(wm))/(2*eps)
    return g

w0 = np.random.randn(X0_bar.shape[0], 1)
g1 = grad(w0)
g2 = num_grad(w0)
diff = np.linalg.norm(g1 - g2)
print('Gradient different: %f' %diff)
```

Tiếp theo, chúng ta xây dựng phương thức lặp Gradient Descent và giải bài toán tối ưu để tìm tham số w, b:

```
def grad_descent(w0, eta):
    w = w0
    it = 0
    while it < 100000:
        it = it + 1
        g = grad(w)
        w -= eta*g
        if (it % 10000) == 1:
            print('iter %d' %it + ' cost: %f' %cost(w))
        if np.linalg.norm(g) < 1e-5:
            break
    return w
w0 = np.random.randn(X0_bar.shape[0], 1)
w = grad_descent(w0, 0.001)
w_hinge = w[:-1].reshape(-1, 1)
b_hinge = w[-1]
print(w_hinge.T, b_hinge)
```

Cách 3. 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 = 100) # use the same C
```

```
# 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('b = ', w0)
```

Hãy so sánh bộ trọng số tối ưu (kết quả) của cả ba 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 theo mỗi phương pháp.

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 soft margin để phân loại cho dữ liệu này.

Ví dụ 3 (Bài tập tự giải): Lọc thư rác:

Tập dữ liệu Spambase cung cấp 4601 mẫu thư điện tử tiếng Anh dạng thư rác và không phải thư rác cho trong link: http://archive.ics.uci.edu/ml/datasets/Spambase .

Tập dữ liệu này thường được dùng để đánh giá hiệu quả của các thuật toán lọc thư rác tự động. Một số tính chất của tập dữ liệu này:

Kích thước mẫu: 4601; Kiểu của đặc trưng: số nguyên và số thực. Số đặc trưng: 57.

Mô tả danh sách các đặc trưng: 48 số thực trong đoạn [0, 100] thuộc kiểu word_freq_WORD là phần trăm từ trong thư là WORD, tức là 100*(số lần từ WORD xuất hiện trong thư) / tổng số từ trong thư;

Mỗi "từ" là bất kì một chuỗi kí tự nào, có thể là từ theo nghĩa thông thường hoặc một kí hiệu (token)

Một số từ thuộc 48 từ được xét: make, address, all, 3d, our, money, technology, conference..

6 số thực trong đoạn [0, 100] thuộc kiểu char_freq_WORD là phần trăm kí tự trong thư là CHAR, tức là 100*(số lần kí tự CHAR xuất hiện trong thư) / tổng số kí tự trong thư. Sáu kí tự được xét là ;, (, [, !, \$ và $\#$.}

1 số thực trong đoạn [1,…] thuộc kiểu capital_run_length_average là độ dài trung bình của các chuỗi chứa toàn các kí tự hoa trong thư.

1 số nguyên trong đoạn [1,…] thuộc kiểu capital_run_length_longest là độ dài của chuỗi dài nhất chứa toàn

các kí tự hoa.

1 số nguyên trong đoạn [1,…] thuộc kiểu capital_run_length_total là tổng độ dài của các chuỗi chứa toàn các kí tự hoa, tức là tổng số kí tự hoa trong thư.

Nếu thư là thư rác thì nó được đánh dấu thuộc lớp 1, không phải thư rác thì thuộc lớp 0. Để áp dung SVM ta cần đổi về 1 và -1.

Bài tập

Lập trình cho mô hình SVM softmargin với C tùy chọn, trong đó dữ liệu được chia 70% là training và 30% là test.