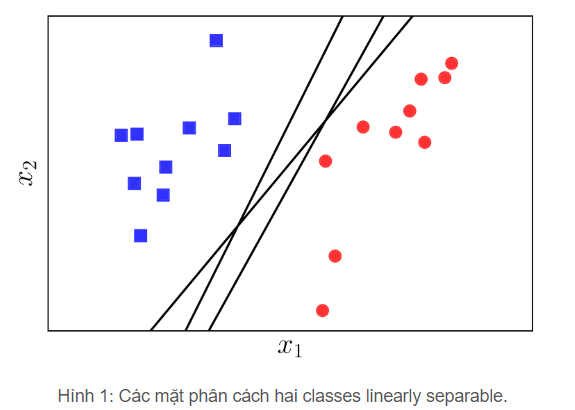
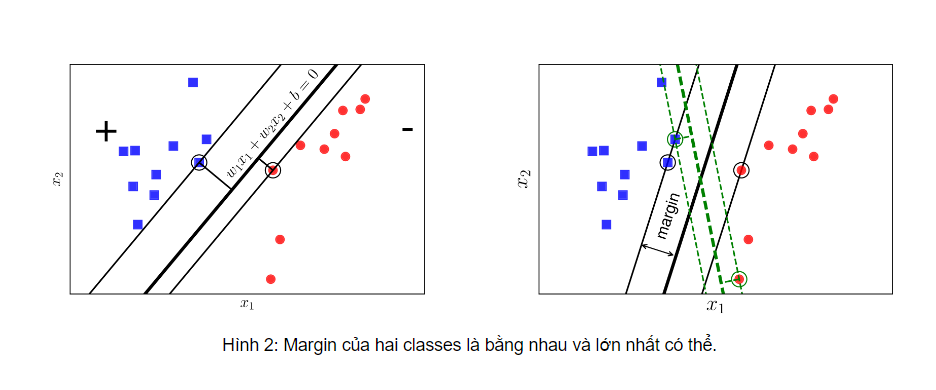
**Convex sets và convex functions**

**Bài toán**  [Perceptron Learning Algorithm (PLA)](https://machinelearningcoban.com/2017/01/21/perceptron/)

****

Câu hỏi đặt ra là: trong vô số các mặt phân chia đó, đâu là mặt phân chia tốt nhất theo một tiêu chuẩn nào đó? Trong ba đường thẳng minh họa trong Hình 1 phía trên, có hai đường thẳng khá lệch về phía class hình tròn đỏ. Điều này là không công bằng cho bên xanh.

Ta sẽ xây dựng thuật toán để xây dựng mặt phẳng nào đó phân chia đều cho cả 2 bên.

****

**Ta có thể thấy trong hình 2a, khoảng cách từ mặt phẳng phân chia đến điểm gần nhất của 2 class là không như nhau.**

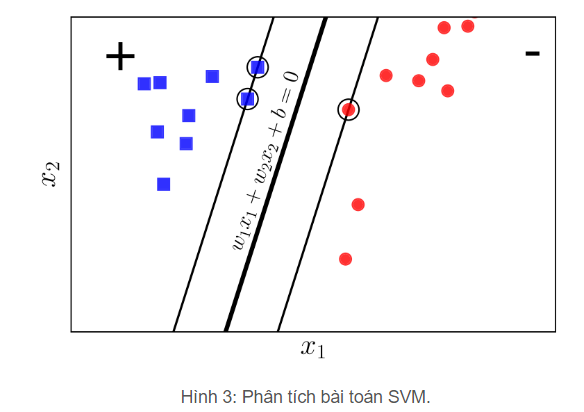
**Trong hinh 2b, khoảng cách từ mặt phẳng được biểu diễn bởi nét màu xanh và màu đen đến điểm gần nhất là như nhau nhưng khoảng cách margin là nhỏ hơn so với mặt phẳng biểu diễn bởi màu đen.**

Bài toán tối ưu trong Support Vector Machine (SVM) chính là bài toán đi 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.

## **2. Xây dựng bài toán tối ưu cho SVM**

Giả sử rằng các cặp dữ liệu của training set là (x1,y1),(x2,y2),…,(xN,yN) với vector xi∈Rd thể hiện đầu vào của một điểm dữ liệu và yi là nhãn của điểm dữ liệu đó. d là số chiều của dữ liệu và N là số điểm dữ liệu. Giả sử rằng nhãn của mỗi điểm dữ liệu được xác định bởi yi=1 (class 1) hoặc yi=−1 (class 2)

Ta xét trường hợp không gian 2 chiều dưới đây:



Giả sử rằng các điểm vuông xanh thuộc class 1, các điểm tròn đỏ thuộc class -1 và mặt wTx+b=w1x1+w2x2+b=0là mặt phân chia giữa hai classes (Hình 3). Hơn nữa, class 1 nằm về phía dương, class -1 nằm về phía âm của mặt phân chia. Nếu ngược lại, ta chỉ cần đổi dấu của w và b.

Ta quan sát thấy một điểm quan trọng sau đây: với cặp dữ liệu (xn,yn) bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

Với mặt phẳng chia như trên, margin được tính là khoảng cách gần nhất từ 1 điểm tới mặt đó.

Margin =

Bài toán tối ưu trong SVM chính là bài toán tìm w và b sao cho margin đạt giá trị lớn nhất:

(w, b) = arg { } = arg {}

Việc giải trực tiếp bài toán này sẽ rất phức tạp. Ta sẽ đưa nó về bài toán đơn giản hơn.

Nhận xét: Nếu thay vector hệ số w bởi kw và b bởi kb với k là một hằng số dương thì mặt phân chia không thay đổi, tức khoảng cách từ từng điểm đến mặt phân chia không đổi, tức margin không đổi.

Giả sử = 1

Chart, scatter chart

Description automatically generated

Với mọi n, ta có:

Vậy bài toán ban đầu có thể đưa về bài toán tối ưu có rang buộc sau đây:

(w, b) = arg

Subject to : ∀n=1,2,…,N

Bài toán trên tương đương bài toán:

(w, b) = arg   
subject to: 1 - ≤ 0 , ∀n=1,2,…,N 

Chúng ta sẽ giải bài toán đối ngẫu của bài toán này.

3.Bài toán đối ngẫu cho SVM

3.1 Kiểm tra tiêu chuẩn Slater

Điều kiện Slater nói rằng: Nếu tồn tại w, b thỏa mãn: 1 - thì strong duality thỏa mãn.

Việc kiểm tra này tương đối đơn giản. Vì ta biết rằng luôn luôn có một (siêu) mặt phẳng phân chia hai classes nếu hai class đó là linearly separable, tức bài toán có nghiệm, nên feasible set của bài toán tối ưu (3)(3) phải khác rỗng. Tức luôn luôn tồn tại cặp (w0,b0) sao cho 1 -

⇔ 2 -

Chọn và , ta có 1 -

Vậy bài toán thỏa mãn điều kiện Slater.

### **3.2. Lagrangian của bài toán SVM**

Lagrangian của bải toán là L(w,b,λ) = (4)  
với λ=[λ1,λ2,…,λN] và λn≥0, ∀n=1,2,…,N≥0.

### **3.3. Hàm đối ngẫu Lagrange**

Hàm đối ngẫu Lagrange được định nghĩa là: g(λ)= L(w,b,λ) với λ⪰0.

Giải hệ phương trình đạo hàm của L(w,b,λ) theo w và b bằng 0, ta có:

= 0 => w = (5)

(6)

Thay (5)(5) và (6)(6) vào (4)(4) ta thu được g(λ)

g(λ)= -

Xét ma trận: V=[y1x1,y2x2,…,yNxN]

và vector 1=[1,1,…,1]T, ta có thể viết lại g(λ)dưới dạng:

g(λ)=V λ + λ

3.4 Bài toán đối ngẫu Lagrange

λ=arg g(λ)

subject to: λ⪰0

= 0

## **4. Lập trình tìm nghiệm cho SVM**

Trước tiên chúng ta gọi các modules cần dùng và tạo dữ liệu giả (dữ liệu này chính là dữ liệu tôi dùng trong các hình phía trên nên chúng ta biết chắc rằng hai classes là linearly separable):

**from** \_\_future\_\_ **import** print\_function

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** scipy.spatial.distance **import** cdist

np.random.seed(22)

means **=** [[2, 2], [4, 2]]

cov **=** [[.3, .2], [.2, .3]]

N **=** 10

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*

Tiếp theo, chúng ta giải bài toán (9)(9) bằng CVXOPT:

**from** cvxopt **import** matrix, solvers

*# build K*

V **=** np.concatenate((X0.T, **-**X1.T), axis **=** 1)

K **=** matrix(V.T.dot(V)) *# see definition of V, K near eq (8)*

p **=** 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*

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(K, p, G, h, A, b)

l **=** np.array(sol['x'])

**print**('lambda = ')

**print**(l.T)

Kết quả:

lambda =

[[ 8.54018321e-01 2.89132533e-10 1.37095535e+00 6.36030818e-10

4.04317408e-10 8.82390106e-10 6.35001881e-10 5.49567576e-10

8.33359230e-10 1.20982928e-10 6.86678649e-10 1.25039745e-10

2.22497367e+00 4.05417905e-09 1.26763684e-10 1.99008949e-10

2.13742578e-10 1.51537487e-10 3.75329509e-10 3.56161975e-10]]

Ta nhận thấy rằng hầu hết các giá trị của lambda đều rất nhỏ, tới 10−910−9 hoặc 10−1010−10. Đây chính là các giá trị bằng 0 nhưng vì sai số tính toán nên nó khác 0 một chút. Chỉ có 3 giá trị khác 0, ta dự đoán là sẽ có 3 điểm là support vectors.

Ta đi tìm support set S rồi tìm nghiệm của bài toán

epsilon **=** 1e-6 *# just a small number, greater than 1e-9*

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)

b **=** np.mean(yS.T **-** w.T.dot(XS))

**print**('w = ', w.T)

**print**('b = ', b)

w = [[-2.00984381 0.64068336]]

b = 4.66856063387

### **4.2. Tìm nghiệm theo thư viện**

Chúng ta sẽ sử dụng hàm [sklearn.svm.SVC](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) ở đây. Các bài toán thực tế thường sử dụng thư viện [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/) được viết trên ngôn ngữ C, có API cho Python và Matlab.

Nếu dùng thư viện thì sẽ như sau:

**from** sklearn.svm **import** SVC

y1 **=** y.reshape((2**\***N,))

X1 **=** X.T *# each sample is one row*

clf **=** SVC(kernel **=** 'linear', C **=** 1e5) *# just a big number*

clf.fit(X1, y1)

w **=** clf.coef\_

b **=** clf.intercept\_

**print**('w = ', w)

**print**('b = ', b)

w = [[-2.00971102 0.64194082]]

b = [ 4.66595309]