Support Vector Machines Dual Problem

SVM - Bài toán đối ngẫu

Bài toán đối ngẫu là gì ?

- Là thay vì giải trực tiếp trên bài toán gốc
- Thì ta sẽ giải bài toán đó trên một không gian mới
- Không gian mới sẽ lợi thế hơn
- Có thể xuất hiện một số tính chất mà không gian gốc không có

Cho bài toán tối ưu gốc (primal problem):

$$\min_{x} f(x)$$

Với các ràng buộc:

$$g_i(x) \leq 0$$

Bài toán đối ngẫu Lagrange (dual problem) của nó sẽ là:

$$\max_{\alpha} \left[\min_{x} \left(f(x) + \sum_{i=1}^{m} \alpha_i g_i(x) \right) \right]$$

Với các ràng buộc:

$$\alpha_i \geq 0$$

Trường hợp khả tách tuyến tính (không lỗi)

Bài toán gốc

$$\min_{w,b} \frac{1}{2}|w|^2$$

Biến đổi ràng buộc để dùng bài toán đối ngẫu:

$$1 - y^{(i)}\left(x_1^{(i)}w_1 + x_2^{(i)}w_2 + \ldots - b\right) \leq 0$$

Bài toán đối ngẫu

Bài toán đối ngẫu Lagrange (dual problem) sẽ là:

$$\max_{\alpha} \min_{(w,b)} \left(\frac{1}{2}|w|^2 + \sum_{i=1}^{m} \alpha_i(1 - y^{(i)}\left(x_1^{(i)}w_1 + x_2^{(i)}w_2 + \ldots - b\right)\right)$$

Với các ràng buộc:

$$\alpha_i \geq 0$$

Thông qua phép biến đổi:

Bài toán đối ngẫu Lagrange (dual problem) sẽ là:

$$\max_{\alpha} \min_{(w,b)} \left(\frac{1}{2}|w|^2 + \sum_{i=1}^{m} \alpha_i (1 - y^{(i)} \left(x_1^{(i)} w_1 + x_2^{(i)} w_2 + \ldots - b \right) \right)$$

Thế vào bài toán max, bài toán đối ngẫu Lagrange (dual problem) sẽ là:

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} \left(x^{(i)} \cdot x^{(j)} \right) \right)$$

Đưa về dạng ma trận

Đổi dấu, bài toán max lại trở thành bài toán min:

$$\min_{\alpha}\left(\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}\left(x^{(i)}\cdot x^{(j)}\right)-\sum_{i=1}^{m}\alpha_i\right)$$

Với các ràng buộc:

$$\boxed{\alpha_i \geq 0}$$

Trường hợp tách được (không có lỗi)

$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

Biểu diễn dưới dạng ma trận:

$$\min_{\alpha}\left(\frac{1}{2}\alpha^T G\alpha - e^T\alpha\right)$$

Với các ràng buộc:

$I\alpha \geq 0$ I: là ma trận đơn vị

$y^T\alpha = 0$

Nếu có lỗi thì thêm: $I\alpha \leq c$

với G là ma trận có các phần tử

$$G_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{m}y^{(i)}y^{(j)}\left(x^{(i)}, x^{(j)}\right)$$

$$G = (DX)(DX)^T = D(XX^T)D$$

X: là ma trận dữ liệu huấn luyện

D: ma trận đường chéo, chứa các $y^{(i)}$.

e: véc-tơ chứa toàn số 1.

```python
# Giải bài toán quy hoạch toàn phương dùng quadprog
sol = qp.solve_qp(G, a, C, b, meq)
```

Bài toán QP trong quadprog

$$\frac{1}{2}x^T G x - a^T x$$

$$C^T x \geq b$$

Nếu có ràng buộc "=" ta để trên,

ràng buộc ">=" ở dưới và gán

meq = số ràng buộc "="

SVM đối ngẫu (không lỗi):

$$\min_{\alpha}\left(\frac{1}{2}\alpha^T G \alpha - e^T \alpha\right)$$

các ràng buộc:

$$y^T \alpha = 0$$

$$I\alpha \geq 0$$

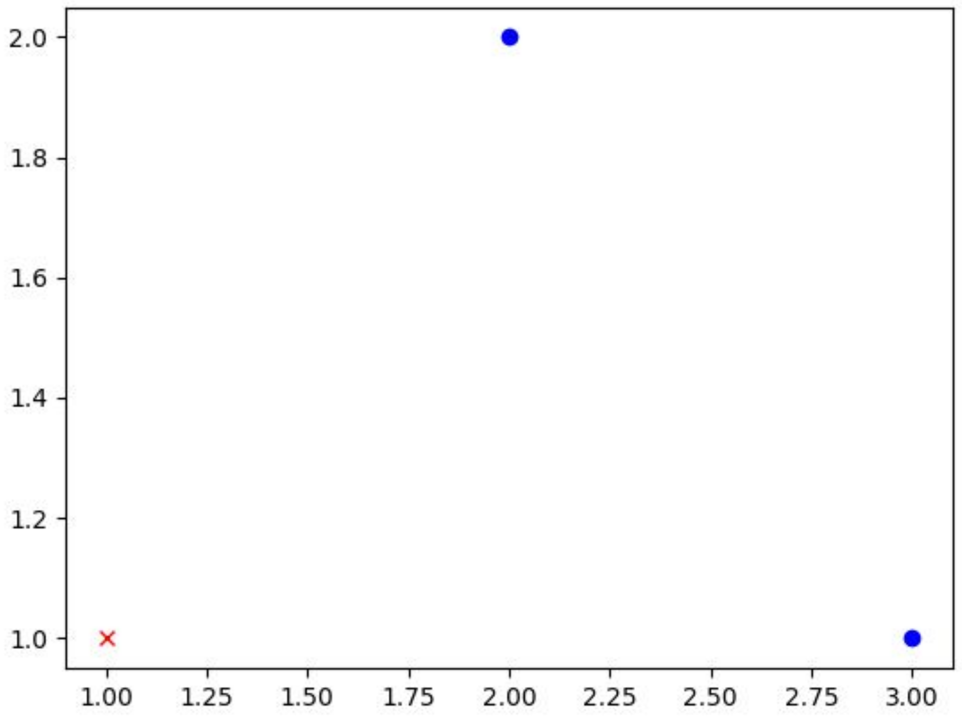
Bài toán khả tách tuyến tính:

Dữ liệu:

```
X = np.matrix([[2.0, 2],
               [3, 1],
               [1, 1],
               ])
Y = np.array([1, 1, -1])
```

Vẽ lên trục tọa độ:

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



Chuẩn bị tham số:

```
# Giải bài toán quy hoạch toàn phương dùng quadprog
sol = qp.solve_qp(G, a, C, b, meq)
```

```
# Tạo một ma trận có đường chéo là Y
D = np.diag(Y)
```

$$D = \begin{pmatrix} +1 & 0 & 0 \\ 0 & +1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

```
# (X * X.T)
XXT = np.dot(X, X.T)
```

```
G = np.dot(np.dot(D, XXT), D)
```

$$G = (DX)(DX)^T = D(XX^T)D$$

Chuẩn bị tham số:

```
# Giải bài toán quy hoạch toàn phương dùng quadprog
sol = qp.solve_qp(G, a, C, b, meq)
```

Biểu diễn dưới dạng ma trận:

$$\min_{\alpha}\left(\frac{1}{2}\alpha^T G\alpha - e^T\alpha\right)$$

e: véc-tơ chứa toàn số 1.

Bài toán QP trong quadprog

$$\frac{1}{2}x^T Gx - a^T x$$

```
# Vector toan so 1
a = np.array(m * [1.0])
```

Chuẩn bị tham số:

```python
# Giải bài toán quy hoạch toàn phương dùng quadprog
sol = qp.solve_qp(G, a, C, b, meq)
```

Nếu có ràng buộc "=" ta để trên,

ràng buộc ">=" ở dưới và gán

meq = số ràng buộc "="

các ràng buộc:

$$y^T\alpha = 0$$

$$I\alpha \geq 0$$

```python
# Ma tran co duong cheo = 1
I = np.diag(m * [1.0])
# ghep rang buoc
C = np.vstack([Y, I]).T
# vector b ma tran 0
b = np.array((m+1) * [0.0])
meq = 1
```

Tìm alpha

```
sol = qp.solve_qp(G, a, C, b, meq)
alpha = sol[0]
print(alpha)
```

```
[1.00000095e+00 4.99999486e-11 1.00000191e+00]
```

Tính W

```
w = sum(np.dot(np.diag(alpha), np.dot(D, X)), 0).T
print(w[0,0])
print(w[1,0])
```

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

Tính b

```
k = np.argmax(alpha)
# bias
b = np.dot(X[k], w) - Y[k]
b = b[0, 0]
```

Để tính độ lệch (bias) b, chọn 1 **véc-tơ hỗ trợ k** bất kỳ có α_i < c:

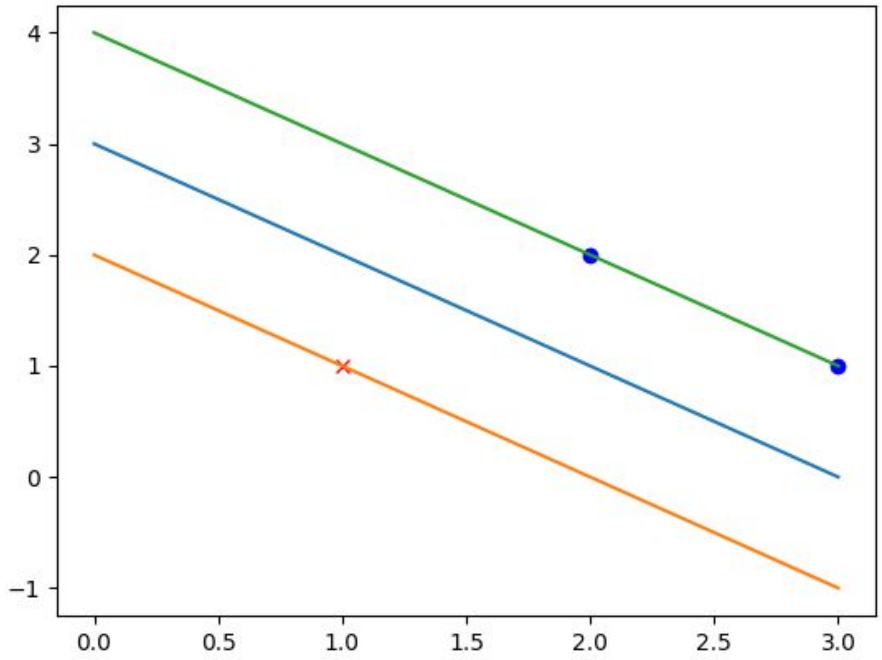
$$b = w^T x^{(k)} - y^{(k)}$$

$$b = \left(w_1 x_1^{(k)} + w_2 x_2^{(k)} + \ldots\right) - y^{(k)}$$

Vẽ đường thẳng:

```
# Ve duong thang
x1 = np.array([0, 3])
d1 = (b - w[0, 0] * x1) / w[1, 0]
# x1 + x2 - b = -1
d2 = ((b-1) - w[0, 0] * x1) / w[1, 0]
# x1 + x2 - b = 1
d3 = ((b+1) - w[0, 0] * x1) / w[1, 0]
```

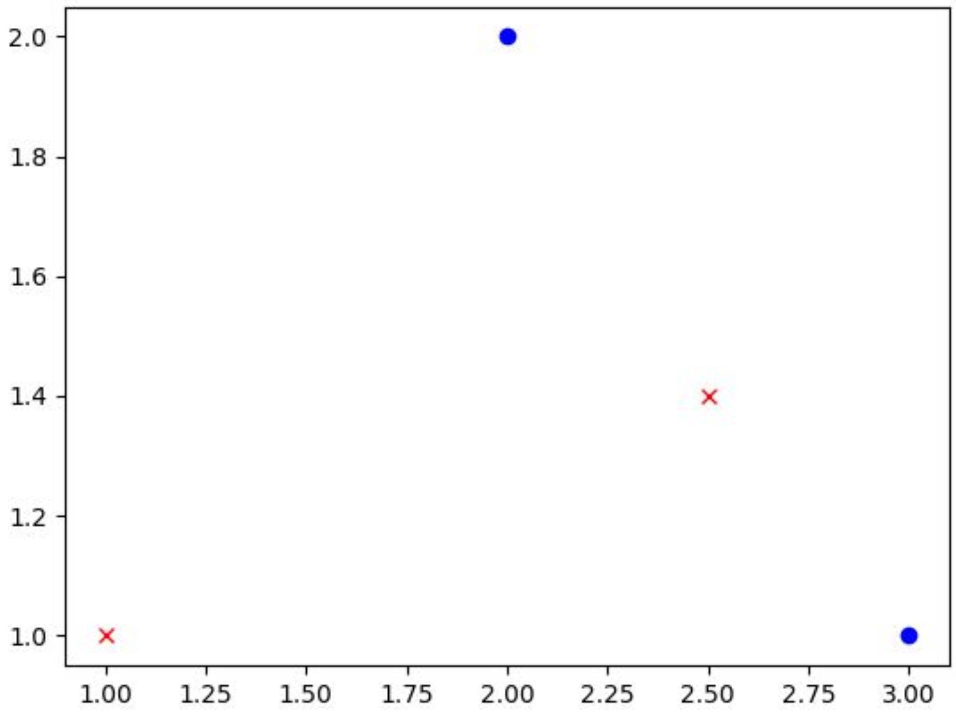
```
plt.plot(x1, d1)
plt.plot(x1, d2)
plt.plot(x1, d3)
plt.show()
```



Bài toán có lỗi

```
X = np.matrix([[2.0, 2],
               [3, 1],
               [1, 1],
               [2.5, 1.4]
               ])
Y = np.array([1, 1, -1, -1])
```

```
# len(Y)
m = len(Y)
```



Bổ sung ràng buộc:

```
# Ma tran co duong cheo = 1
I = np.diag(m * [1.0])
# ghep rang buoc
C = np.vstack([Y, I, -I]).T
# vector b ma tran 0
c = 100
# c lớn thì lỗi nhỏ, lề nhỏ
# c nhỏ thì lỗi nhiều, lề lớn
b = np.array((m + 1) * [0.0] + m * [-c])
```

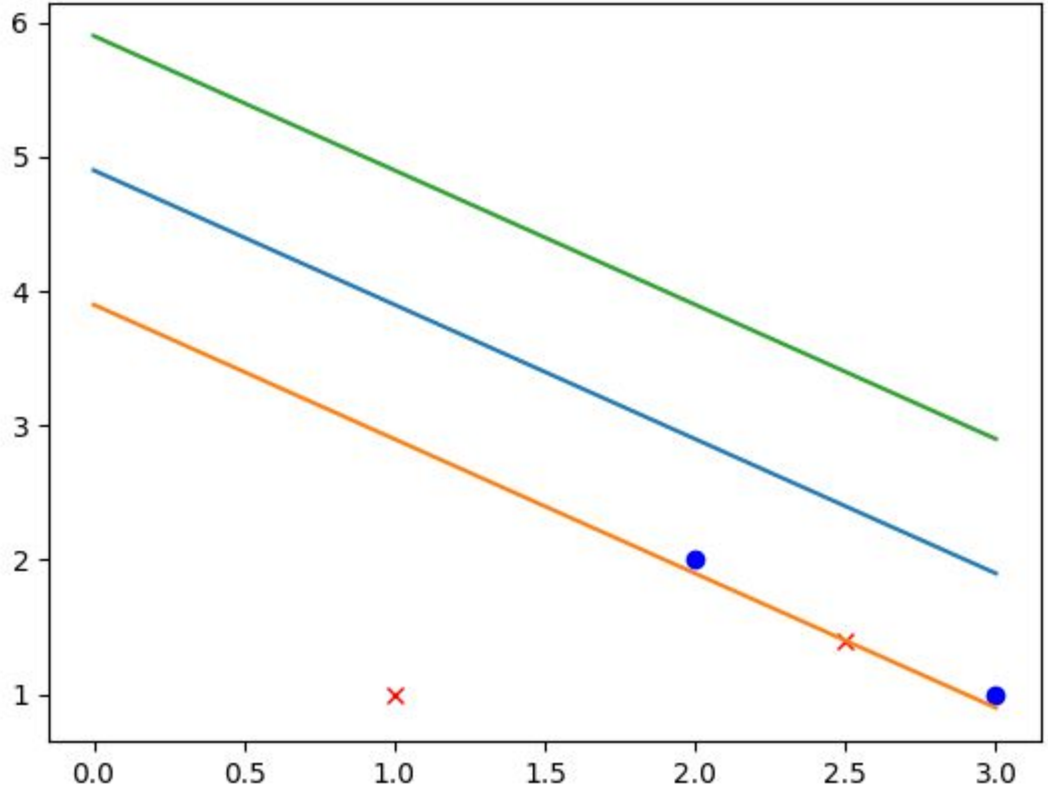
các ràng buộc:

$$y^T\alpha = 0$$

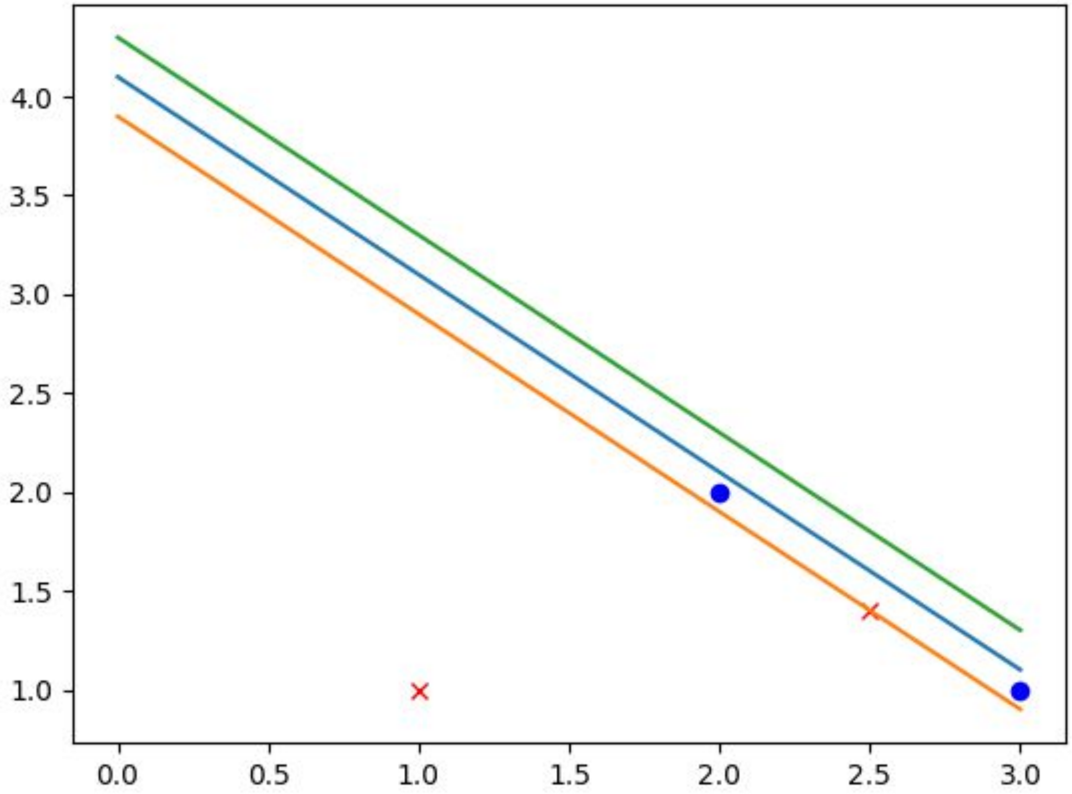
$$I\alpha \geq 0$$

$$\boxed{\begin{aligned} I\alpha &\leq c \\ \Leftrightarrow \quad -I\alpha &\geq -c \end{aligned}}$$

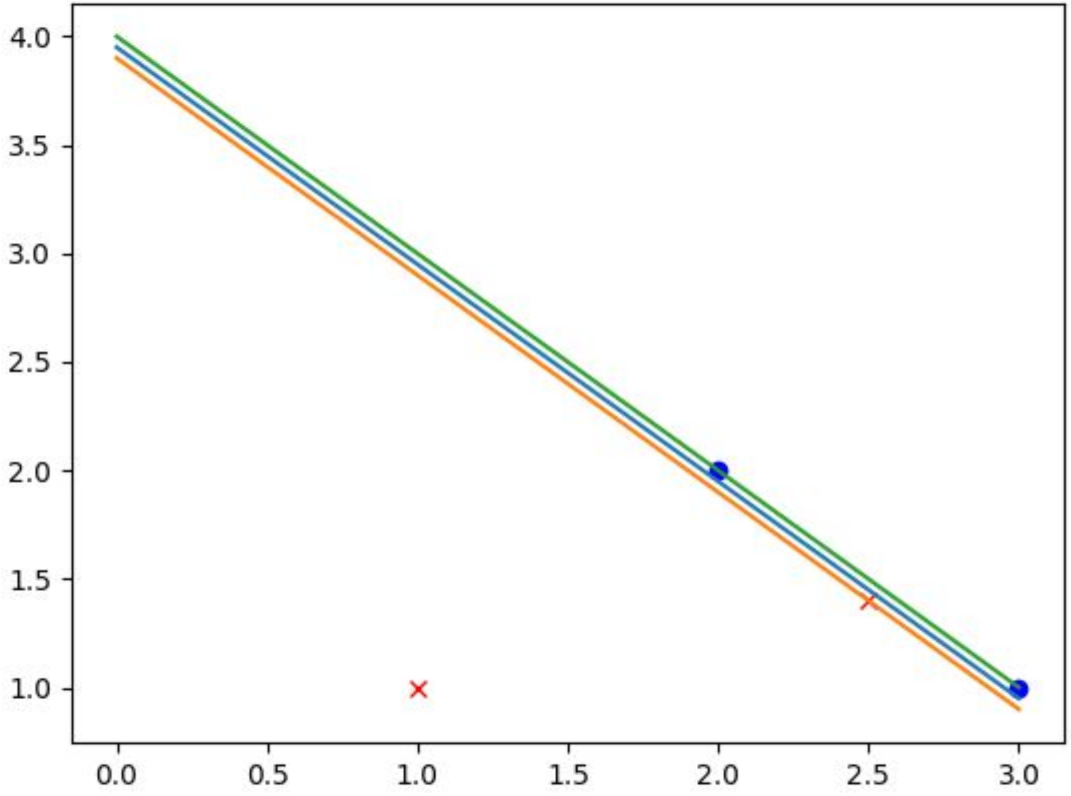
c = 10



c = 100



c = 1000



Kernel

```python
def RBF_kernel(xi, xj, gama=1.0):
    diff = xi - xj
    dist = np.dot(diff, diff.T)
    return np.exp(-gama * dist)
```

```python
kernel = RBF_kernel
# Tạo một ma trận có đường chéo là Y
D = np.diag(Y)
m = len(Y)
K = np.zeros((m, m))
for i in range(m):
    for j in range(m):
        K[i][j] = kernel(X[i], X[j])
G = np.dot(np.dot(D,K),D)
```