# Linear Discriminant Function & SVM

> Pattern Recognition Homeworks
>
> Student: thu-zxs

## Solutions

### Problem 1

1.

Consider a support vector $z^{(s)}$ s.t.

$$\beta^T z^{(s)} \geq 1, y^{(s)} = 1$$
$$\beta^T z^{(s)} \leq 1, y^{(s)} = -1$$

Where $\beta$ can be found by letting distance between $z^{(s)}$ and the hyperplane parameterized by $\beta$ to be:

$$r = \begin{cases} \frac{1}{||\beta||}, y^{(s)} = 1 \\ \frac{-1}{||\beta||}, y^{(s)} = -1 \end{cases}$$

then take $\beta_{sep}$ to be $\beta$ we yield the result as:

$$y_i \beta_{sep}^T z_i \geq 1, \forall i$$

2.

$$\begin{aligned} ||\beta_{new} - \beta_{sep}||^2 &= ||\beta_{old} + y_i z_i - \beta_{sep}||^2 \\ &\leq ||\beta_{old} - \beta_{sep}||^2 - || - y_i z_i||^2 = ||\beta_{old} - \beta_{sep}||^2 - 1 \end{aligned}$$

hence when algorithm is converged,

$$0 \leq ||\beta_{old} - \beta_{sep}||^2 - 1 \leq ||\beta_{old-1} - \beta_{sep}||^2 - 2 \leq \ldots \leq ||\beta_{start} - \beta_{sep}||^2 - n$$

So number of steps $n$ satisfies:

$$n \leq ||\beta_{start} - \beta_{sep}||^2$$

### Problem2

$$\phi(x_1) = [1, 0, 0]$$
$$\phi(x_2) = [1, 2, 2]$$

choose to solve the dual problem:

$$
\begin{aligned}
&\min_{\alpha} \frac{1}{2}\sum_{i}^{N}\sum_{j}^{N}\alpha_i\alpha_j y_i y_j(\phi(x_i)\cdot\phi(x_j)) - \sum_{i}^{N}\alpha_i \\
&s.t. \sum_i \alpha_i y_i = 0 \\
&\alpha_i \geq 0, \forall i
\end{aligned}
$$

that is

$$
\begin{aligned}
&\min_{\alpha} \frac{1}{2}(\alpha_1^2 + 9\alpha_2^2 - 2\alpha_1\alpha_2) - \alpha_1 - \alpha_2 \\
&s.t. \alpha_1 = \alpha_2 \\
&\alpha_1 \geq 0, \alpha_2 \geq 0
\end{aligned}
$$

taking derivative of $\alpha_1$ and make it equals to $0$ we have $\alpha_1 = \alpha_2 = \frac{1}{4}$

so,

$$
\begin{aligned}
w^* &= \alpha_1 y_1 \phi(x_1) + \alpha_2 y_2 \phi(x_2) = [0, \frac{1}{2}, \frac{1}{2}] \\
w_0 &= -1
\end{aligned}
$$

a.

$$
w^* = [0, \frac{1}{2}, \frac{1}{2}]
$$

b.

Margin is:

$$
\frac{2}{||w^*||} = 2\sqrt{2}
$$

c.

$$
w = \frac{1}{2}w^* = [0, \frac{1}{4}, \frac{1}{4}]
$$

d.

$$
\begin{aligned}
f(x) &= w_0 + (w^*)^T\phi(x) \\
&= \frac{1}{2}x^2 + \frac{\sqrt{2}}{2}x - 1
\end{aligned}
$$

# Programming

1. Random generate sample and visualizing:

```
def generate_2cls_data(w, scale=5, n=100):
```

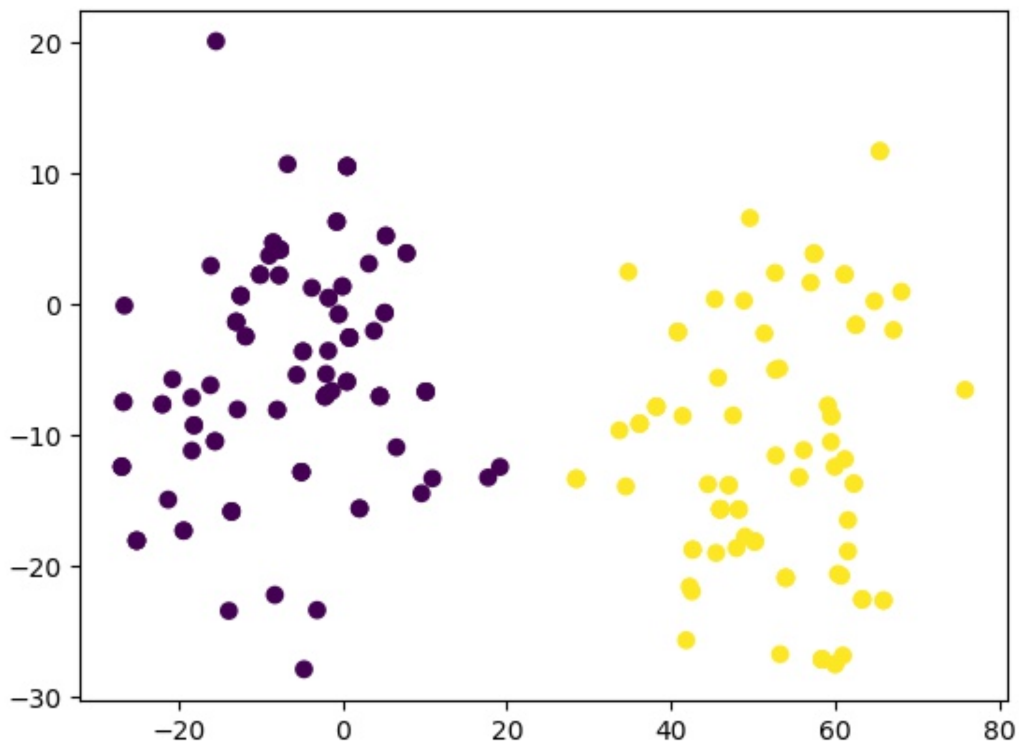
```
    """ w is the normal vector of the hyperplane
    """
    X1 = scale*(2*(np.random.randn(n,2)-0.5))
    margin_point_positive = np.argmax(X1.dot(w), axis=0)
    margin_distance_positive = np.max(X1.dot(w), axis=0)[0]

    X2 = scale*(2*(np.random.randn(n,2)-0.5))
    margin_point_negative = np.argmin(X2.dot(w), axis=0)
    margin_distance_negative = np.min(X2.dot(w), axis=0)[0]

    norm_w = np.linalg.norm(w)
    w_norm = w/norm_w
    dis_vec = (X1[margin_point_positive] - X2[margin_point_negative])
    mv_axis0 = dis_vec.dot(np.array([[1],[0]]))[0]
    mv_axis1 = dis_vec.dot(np.array([[0],[1]]))[0]
    X2[:,0] += mv_axis0*(1.2)
    X2[:,1] += mv_axis1*(1.2)

    return np.vstack((X1, X2)), np.hstack((-np.ones(n, dtype=int),
np.ones(n, dtype=int)))
```

```
X, Y = generate_2cls_data(w=np.array([[0.5],[-0.1]]), scale=5)
rand_idx = np.random.choice(range(X.shape[0]), X.shape[0])
X = X[rand_idx, :]
Y = Y[rand_idx]
plt.scatter(X[:, 0], X[:, 1], marker='o', c=Y)
```



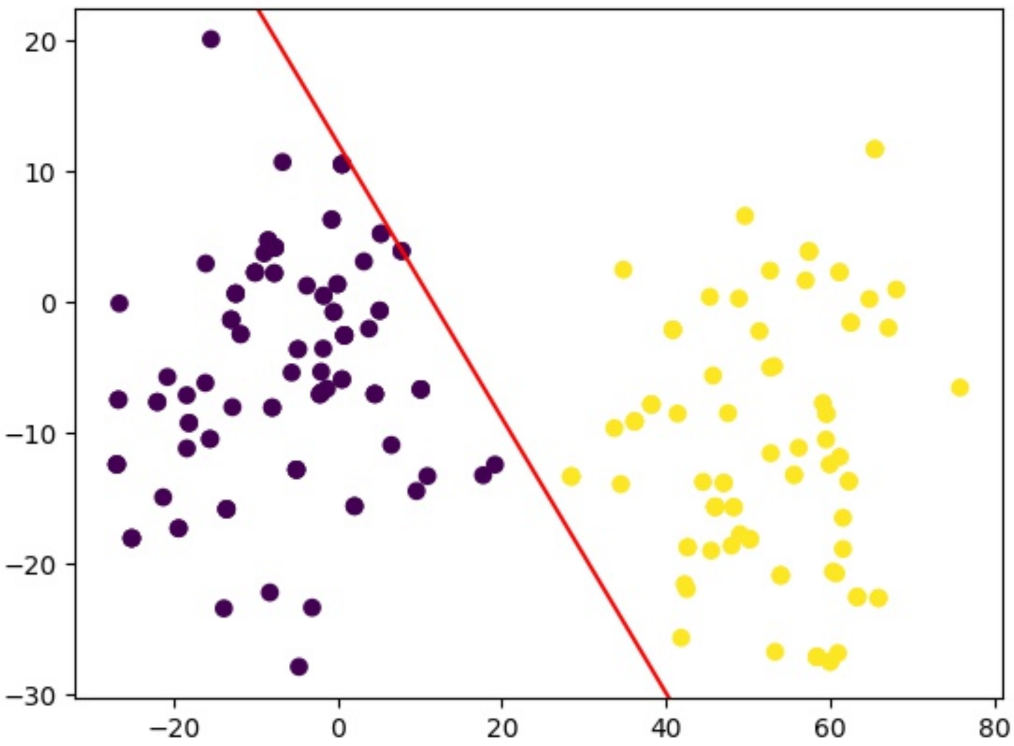

2. classic perceptron algorithm

```
def perceptron(w0, X, Y):
    w = w0
    k = 0
    n = X.shape[0]
    X = np.hstack((X, np.ones((n,1))))
    all_correct = np.zeros(n, dtype=bool)
    while not np.all(all_correct):
        if not Y[k]*X[k,:].dot(w) > 0:
            w = w + Y[k]*X[k,:][:, np.newaxis]
        k = (k+1)%n
        all_correct = (Y*np.squeeze(X.dot(w)))>0
    return w
```

```
w1 = perceptron(np.array([[-0.1], [-0.5], [3]]), X, Y)
print(w1)
w1 = np.squeeze(w1)
p1 = [0, -w1[2]/w1[1]]
p2 = [-w1[2]/w1[0], 0]
l1 = newline(p1,p2,'r')
```

get the result of $\alpha$:

```
[[  8.0209055 ]
 [  7.62012004]
 [-93.        ]]
```




3. margin perceptron algorithm

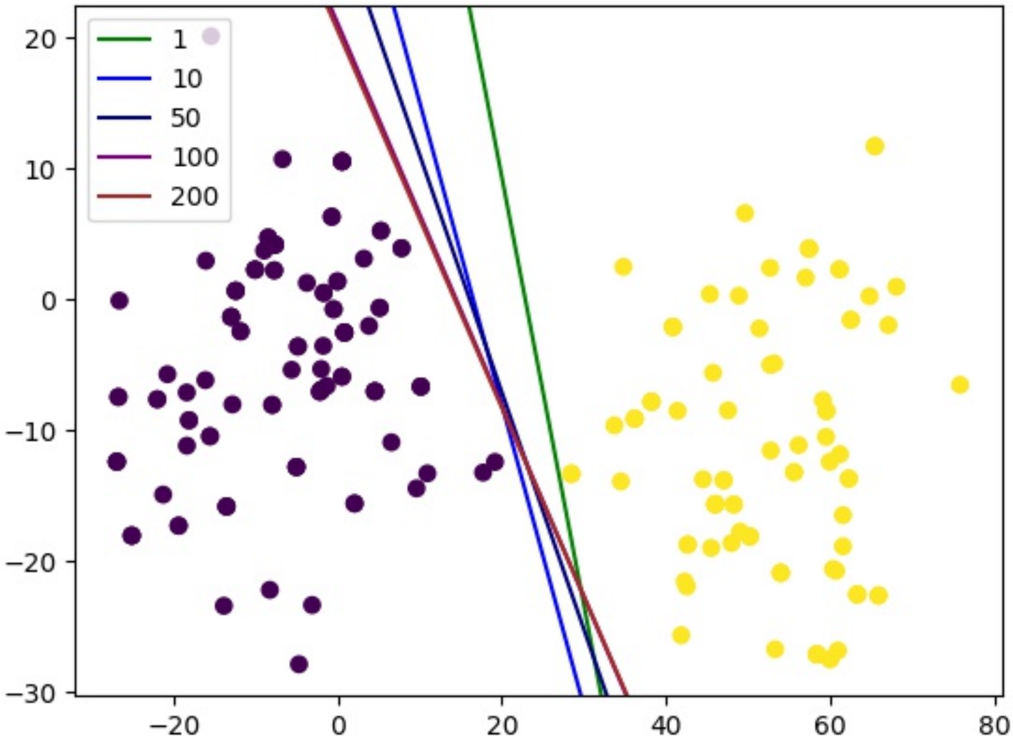
```
def perceptron_margin(w0, X, Y, margin):
    w = w0
    k = 0
    n = X.shape[0]
    X = np.hstack((X, np.ones((n,1))))
    all_correct_margin = np.zeros(n, dtype=bool)
    while not np.all(all_correct_margin):
        if not Y[k]*X[k,:].dot(w) > margin:
            w = w + Y[k]*X[k,:][:, np.newaxis]
        k = (k+1)%n
        all_correct_margin = (Y*np.squeeze(X.dot(w))) > margin
    return w
```

```
ax = plt.gca()
ax.lines.remove(l1)
colors = ['green', 'blue', 'navy', 'purple', 'brown']
ls = []
margins = [1, 10, 50, 100, 200]
for i,k in enumerate(margins):
```

```
    w2 = perceptron_margin(np.array([[-0.1], [-0.5], [3]]), X, Y,
margin=k)
    print("margin: {}\n{}".format(k, w2))
    w2 = np.squeeze(w2)
    p3 = [0, -w2[2]/w2[1]]
    p4 = [-w2[2]/w2[0], 0]
    l = newline(p3,p4,colors[i])
    ls.append(l)
plt.legend(ls, np.array(margins, dtype=np.str))
```

get the result of $\alpha$(s):

```
margin: 1
[[   3.58857155]
 [   1.09576698]
 [-82.        ]]
margin: 10
[[   5.75051003]
 [   2.48918374]
 [-95.        ]]
margin: 50
[[  15.84975964]
 [   8.739276  ]
 [-255.        ]]
margin: 100
[[  27.75299081]
 [  18.98105444]
 [-401.        ]]
margin: 200
[[  56.72367228]
 [  39.32919139]
 [-809.        ]]
```




with the increase of margin, the algorithm took more steps to converge, and yield a more robust and compact seperation hyperplane (or dividing the support vector more evenly)