Machine Learning Techniques

Shyi-Chyi Cheng (鄭錫齊)
Email:csc@mail.ntou.edu.tw
Tel: 02-24622192-6653

章節目錄

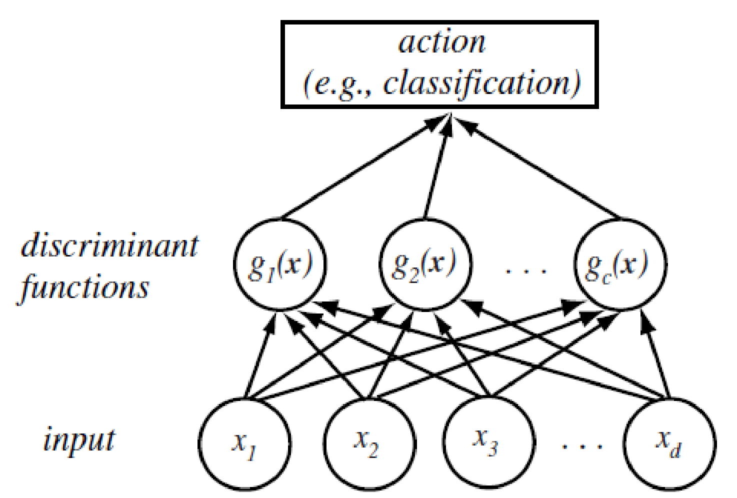
Discriminant Functions

❖ Discriminant functions

$$g_i : R^d \rightarrow R \quad (1 \leq i \leq c)$$

- *Useful way to represent classifiers*
- *One function per category*

Decide ω_i if $g_i(\underline{x}) > g_j(\underline{x})$ for all $j \neq i$



Minimum risk: $g_i(\underline{x}) = -r(\alpha_i \mid \underline{x}) \quad (1 \leq i \leq c)$

Minimum error-rate: $g_i(\underline{x}) = P(\omega_i \mid \underline{x}) \quad (1 \leq i \leq c)$

Discriminant Functions (cont.)

❖ **Decision region**

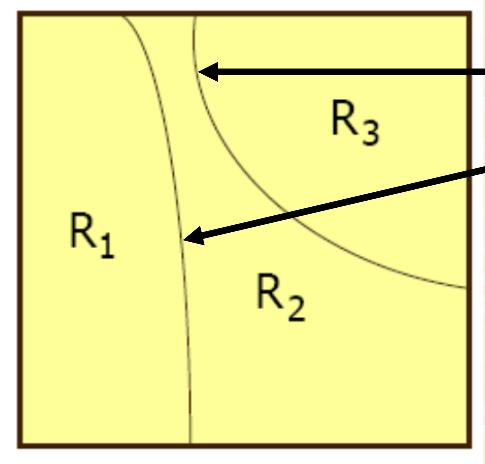
c discriminant functions $g_i(\cdot)\ (1 \le i \le c)$ $\Rightarrow$ c decision regions $R_i \subset \mathrm{R}^d\ (1 \le i \le c)$

$$R_i = \left\{ \underline{x} \mid \underline{x} \in \mathrm{R}^d : g_i(\underline{x}) > g_j(\underline{x})\ \forall j \ne i \right\}$$

$$where\ R_i \cap R_j = \varnothing (i \ne j) \text{ and } \cup_{i=1}^{c} R_i = \mathrm{R}^d$$

❖ **Decision boundary**

surface in feature space where ties occur among several largest discriminant functions



Decision boundary

Linear Discriminant Functions

$$g_i(\underline{x}) = \underline{w}_i^t \underline{x} + w_{i0} \quad (i=1,2,\ldots,c)$$

$\underline{w}_i$: *weight vector* (權值向量, d-dimensional)

w_{i0} : bias/threshold (偏置/閥值)

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$d = 3$, $c = 3$

$$g_1(\underline{x}) = 2x_1 - x_2 + 3x_3 \Rightarrow w_1 = 2;\ w_2 = -1; w_3 = 3;\ w_{1,0} = 0$$

$$g_2(\underline{x}) = x_1 + 3x_2 + 3 \Rightarrow w_1 = 1;\ w_2 = 3; w_3 = 0;\ w_{2,0} = 3$$

$$g_3(\underline{x}) = 5x_1 + 4x_2 + 3x_3 - 5 \Rightarrow w_1 = 5;\ w_2 = 4; w_3 = 3;\ w_{3,0} = -5$$

Linear Discriminant Functions (Cont.)

❖ The two-category case

$$g_1(\underline{x}) = \underline{w}_1^t \underline{x} + w_{10}$$

$$g_2(\underline{x}) = \underline{w}_2^t \underline{x} + w_{20} \quad g(\underline{x}) = g_1(\underline{x}) - g_2(\underline{x})$$

$$\begin{cases} \underline{x} \in \omega_1, \text{ if } g(\underline{x}) > 0 \\ \underline{x} \in \omega_2, \text{ otherwise} \end{cases}$$

$$\begin{aligned} g(\underline{x}) &= g_1(\underline{x}) - g_2(\underline{x}) \\ &= (\underline{w}_1^t \underline{x} + w_{10}) - (\underline{w}_2^t \underline{x} + w_{20}) \\ &= (\underline{w}_1^t - \underline{w}_2^t)\underline{x} + (w_{10} - w_{20}) \\ &= (\underline{w}_1 - \underline{w}_2)^t \underline{x} + (w_{10} - w_{20}) \end{aligned}$$

$$Let \ \underline{w} = \underline{w}_1 - \underline{w}_2$$

$$w_0 = w_{10} - w_{20}$$

$$g(\underline{x}) = \underline{w}^T \underline{x} + w_0$$

It suffices to consider only d+1 parameters ($\underline{w}$ and w_0)
instead of 2(d+1) parameters under two-category

LINEAR CLASSIFIERS

- **The Problem:** Consider a two class task with ω_1, ω_2

  - $$g(\underline{x}) = \underline{w}^T \underline{x} + w_0 = 0 = w_1 x_1 + w_2 x_2 + \ldots + w_l x_l + w_0$$

    $$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_l \end{bmatrix}$$

  - Assume $\underline{x}_1, \underline{x}_2$ on the decision hyperplane :

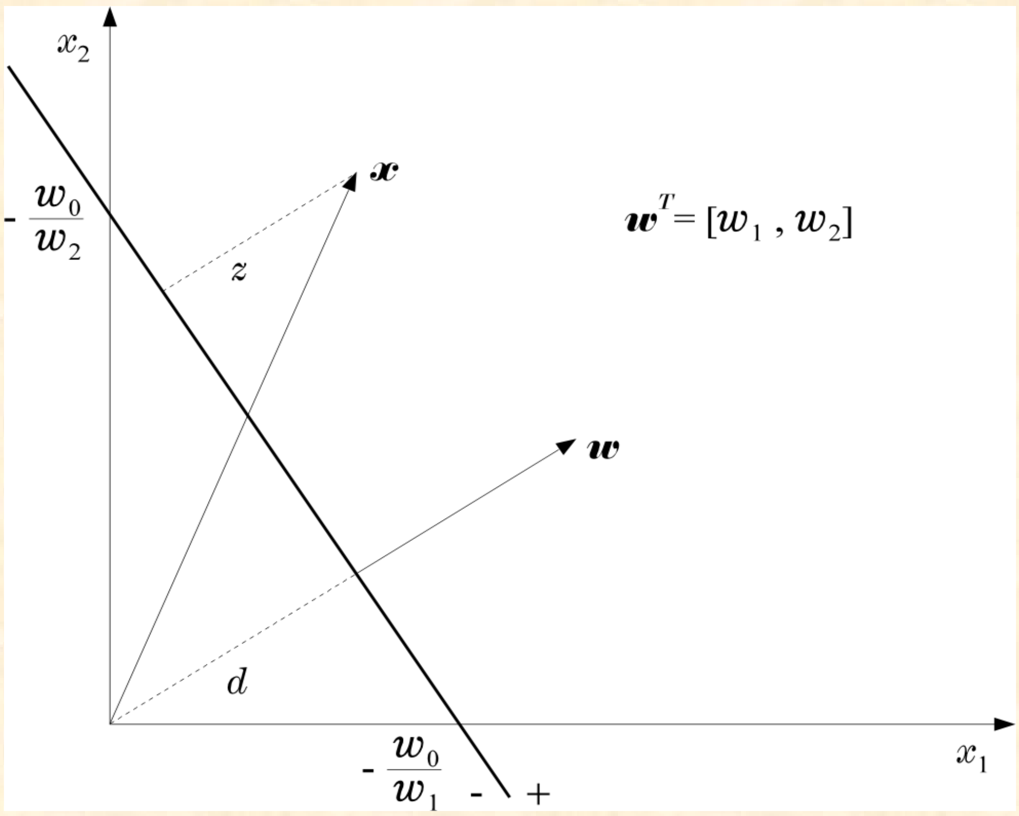
    $$0 = \underline{w}^T \underline{x}_1 + w_0 = \underline{w}^T \underline{x}_2 + w_0 \Rightarrow$$

    $$\underline{w}^T (\underline{x}_1 - \underline{x}_2) = 0 \quad \forall \underline{x}_1, \underline{x}_2$$

➢ Hence:

$\boxed{\underline{w} \perp \underline{x} \text{ on the hyperplane}}$

$$g(\underline{x}) = \underline{w}^T \underline{x} + w_0 = 0$$



$$d = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}, \quad z = \frac{|g(\underline{x})|}{\sqrt{w_1^2 + w_2^2}}$$

Two-Category Case

- **Training set**

$$D^{*}=\left\{\left(\underline{x}_i,\omega_i\right)\mid i=1,2,\ldots,n\right\}\quad\left(\underline{x}_i\in R^{l},\omega_i\in\left\{-1,+1\right\}\right)$$

- **The task**
  - **Determine** $g(\underline{x})=\underline{w}^{T}\underline{x}+w_0$ **which can classify all training examples in D* correctly:**

$$g(\underline{x}_i)=\underline{w}^{T}\underline{x}_i+w_0>0,\ \text{if}\ \omega_i=+1$$

$$g(\underline{x}_i)=\underline{w}^{T}\underline{x}_i+w_0<0,\ \text{if}\ \omega_i=-1$$

$$\omega_i\cdot g(\underline{x}_i)>,\ i=1,2,\ldots,n$$

Two-Category Case (cont.)

- **Solution to $(\underline{w}, w_0)$ $(g(\underline{x}) = \underline{w}^T \underline{x} + w_0)$**
  - Minimize a criterion/objective function (準則涵式) $J(\underline{w}, w_0)$ based on the training examples:

$$(\underline{x}_i, \omega_i), i = 1, 2, \ldots, n$$

- **Two questions to answer**
  - **How to define the objective function *J*?**

$$J(\underline{w}, w_0) = -\sum_{i=1,\ldots,n} \omega_i \cdot g(\underline{x}_i)$$

$$J(\underline{w}, w_0) = -\sum_{i=1,\ldots,n} sign[\omega_i \cdot g(\underline{x}_i)]$$

$$J(\underline{w}, w_0) = \sum_{i=1,\ldots,n} [g(\underline{x}_i) - \omega_i]^2$$

  - **How to minimize *J*? Gradient Descent (梯度下降法)?**

Gradient Descent

❖ **Taylor Expansion (泰勒展開式)**

$$f(\underline{x}+\Delta\underline{x}) = f(\underline{x}) + \nabla f(\underline{x})^t \cdot \Delta\underline{x} + O(\Delta\underline{x}^t \cdot \Delta\underline{x})$$

$f : \mathrm{R}^l \rightarrow \mathrm{R}$: a real-valued *l*-variate function

$\underline{x} \in \mathrm{R}^l$: a point in the *l*-dimension Euclidean space

$\Delta\underline{x} \in \mathrm{R}^l$: a small shift in the *l*-dimension Euclidean space

$\nabla f(\underline{x})$: gradient of $f(\square)$ at $\underline{x}$

$O(\Delta\underline{x}^t \cdot \Delta\underline{x})$: the big O order of $\Delta\underline{x}^t \cdot \Delta\underline{x}$

Gradient Descent (cont.)

- **Taylor Expansion (泰勒展開式)**

$$f(\underline{x}+\Delta\underline{x}) = f(\underline{x}) + \nabla f(\underline{x})^t \cdot \Delta\underline{x} + O(\Delta\underline{x}^t \cdot \Delta\underline{x})$$

- **What happens if we set $\Delta\underline{x}$ to be negatively proportional to the gradient at $\underline{x}$ i.e.:**

$$\Delta\underline{x} = -\eta \cdot \nabla f(\underline{x})$$ (η being a *small* positive scalar)

$$\boxed{f(\underline{x}+\Delta\underline{x}) = f(\underline{x}) + \underbrace{\nabla f(\underline{x})^t \cdot \Delta\underline{x}}_{\text{being non-negative}} + \underbrace{O(\Delta\underline{x}^t \cdot \Delta\underline{x})}_{\text{ignored when } O(\Delta\underline{x}^t \cdot \Delta\underline{x}) \text{ is small}}}$$

being *non-negative*

ignored when $\boxed{O(\Delta\underline{x}^t \cdot \Delta\underline{x})}$ is small

$$\Longrightarrow \boxed{f(\underline{x}+\Delta\underline{x}) \le f(\underline{x})!}$$

Gradient Descent (Cont.)

❖ **Basic strategy:** to minimize some *l*-variate function *f*(.), the general gradient descent techniques work in the following *iterative way*:

1. Set learning rate *h* >0 and a small threshold *e* > 0
2. Randomly initialize $\underline{x}_0 \in \mathrm{R}^l$ as the starting point;Set k =0
3. **do** k =k+1
4. $\underline{x}_k = \underline{x}_{k-1} - \eta \cdot \nabla f(\underline{x}_{k-1})$ (*gradient descent step*)
5. **Until** $| f(\underline{x}_k) - f(\underline{x}_{k-1}) | < \varepsilon$
6. Return $\underline{x}_k$ and $f(\underline{x}_k)$

Gradient Descent for Two-Category Linear Discriminant Functions

- **Task revisited:**
  - **Determine** $g(\underline{x}) = \underline{w}^T \underline{x} + w_0$ **which can classify all training examples in D* correctly**
- **The solution**

Choose certain criterion function $J(\underline{w}, w_0)$ defined over **D***	Invoke the standard gradient descent procedure on the $(l+1)$-variate function $J(\ ,\ )$ to determine $(\underline{w}, w_0)$

Gradient Descent for Two-Category Linear Discriminant Functions

- Two examples for *J*(.,.)

$$J(\underline{w}, w_0) = -\sum_{i=1,\ldots,n} \omega_i \cdot g(\underline{x}_i)$$

$$\Rightarrow \nabla J(\underline{w}, w_0) = -\sum_{i=1,\ldots,n} \omega_i \cdot \left[\frac{x_i}{1}\right]$$

$$J(\underline{w}, w_0) = \sum_{i=1,\ldots,n} [g(\underline{x}_i) - \omega_i]^2$$

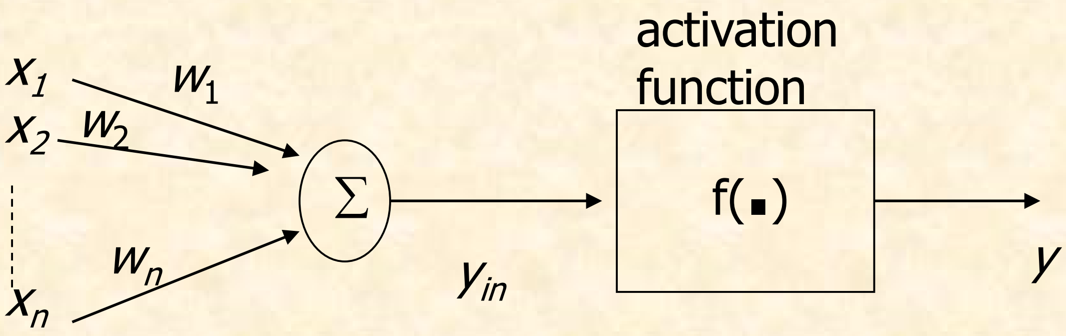
$$\Rightarrow \nabla J(\underline{w}, w_0) = 2 \cdot \sum_{i=1,\ldots,n} (g(\underline{x}_i) - \omega_i) \cdot \left[\frac{x_i}{1}\right]$$

Linear Classifiers Implementation Using Python

Perceptron (感知器)

何謂感知器?

- 感知器是輸入多個訊號之後，再當作一個訊號輸出的裝置
- 感知器是最基本的類神經網路



其中 $y_{in} = \sum_{i=1}^{n} w_i x_i$

又 activation function f(▪) 的定義有下列幾種:

(1) Binary step function

$$f(x) = \begin{cases} 1 & x \geq \theta \\ 0 & x < \theta \end{cases} \qquad \theta : threshold$$

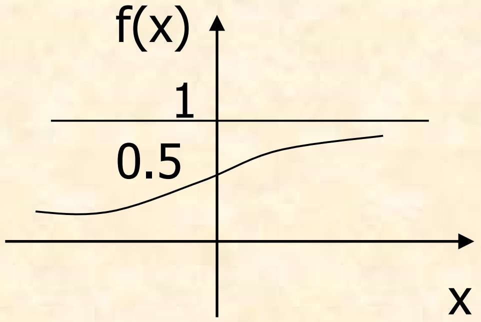
何謂感知器?

(2) Bipolar step function

$$f(x)=\begin{cases}1 & x\geq\theta \\ -1 & x<\theta\end{cases}$$

(3) Binary sigmoid

$$f(x)=\frac{1}{1+\exp(-\sigma x)} \quad \sigma>0$$

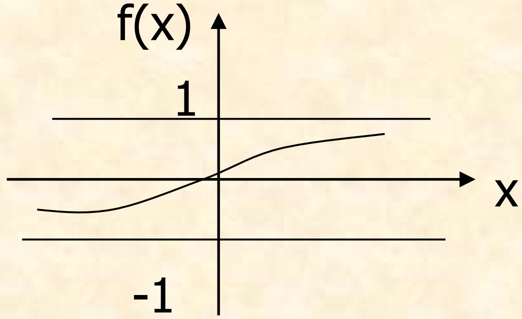


$$\Rightarrow \quad f'(x)=\sigma f(x)[1-f(x)]$$

何謂感知器?

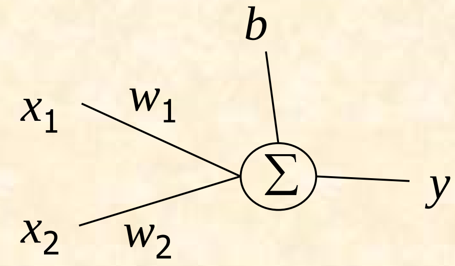
(4) Bipolar sigmoid

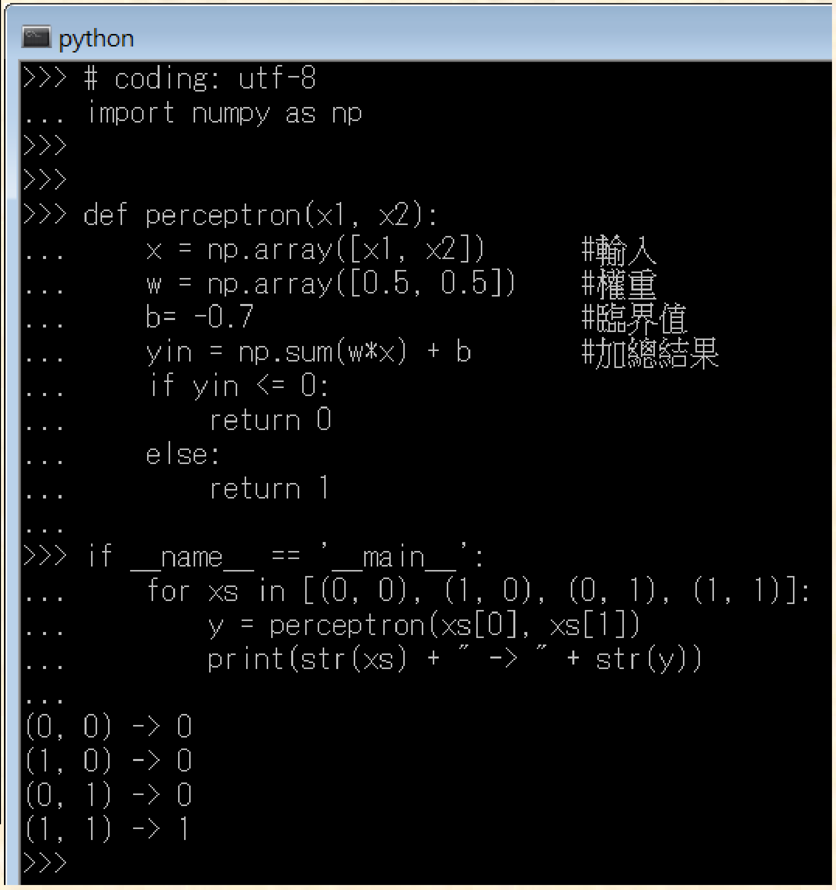
$$f(x)=\frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)} \qquad \sigma>0$$



利用Python執行感知器

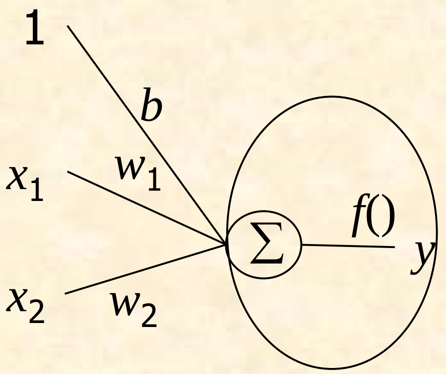
```
# coding: utf-8
import numpy as np


def perceptron(x1, x2):
    x = np.array([x1, x2])      #輸入
    w = np.array([0.5, 0.5])   #權重
    b= -0.7                     #臨界值
    yin = np.sum(w*x) + b     #加總結果
    if yin <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = perceptron(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

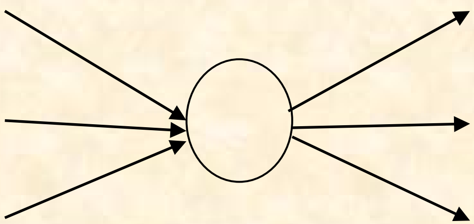




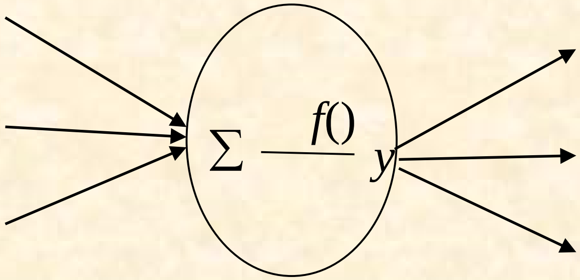
Activation Function (活化函數)

❖ 活化函數$f(.)$決定是否激發神經元的加總輸出





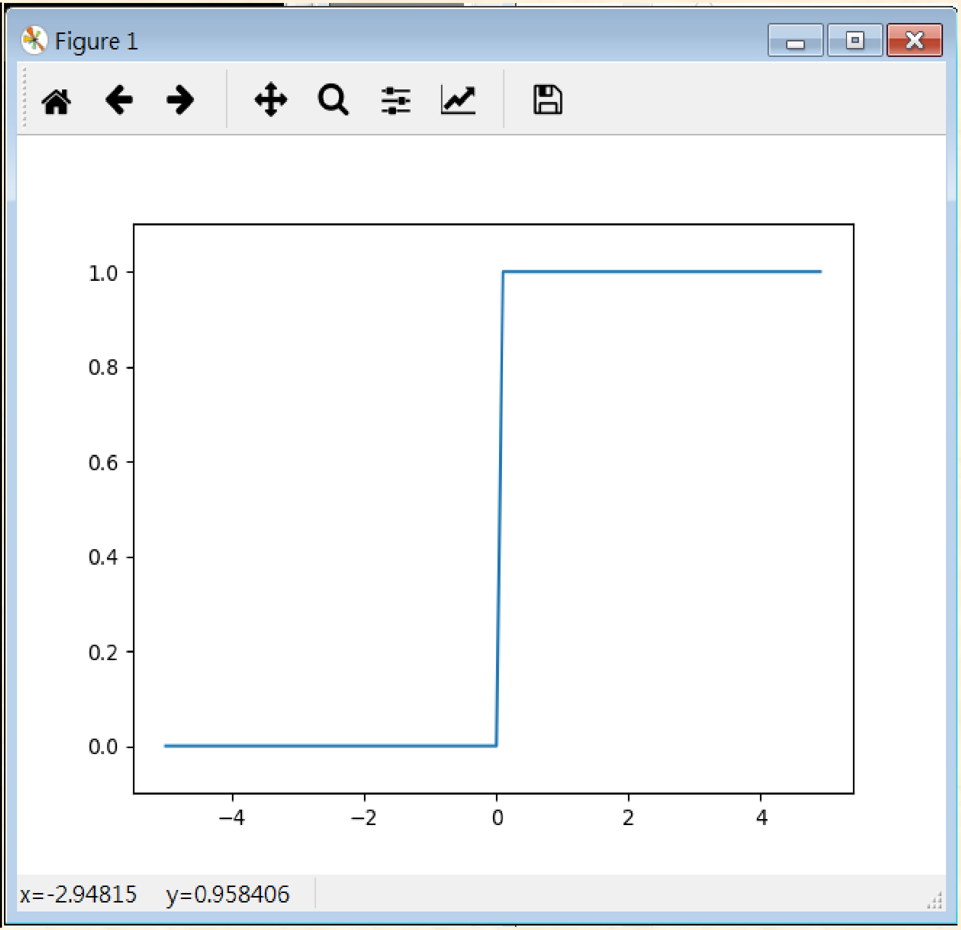
神經元



活化函數是感知器進入神經網路的橋樑

執行階梯活化函數

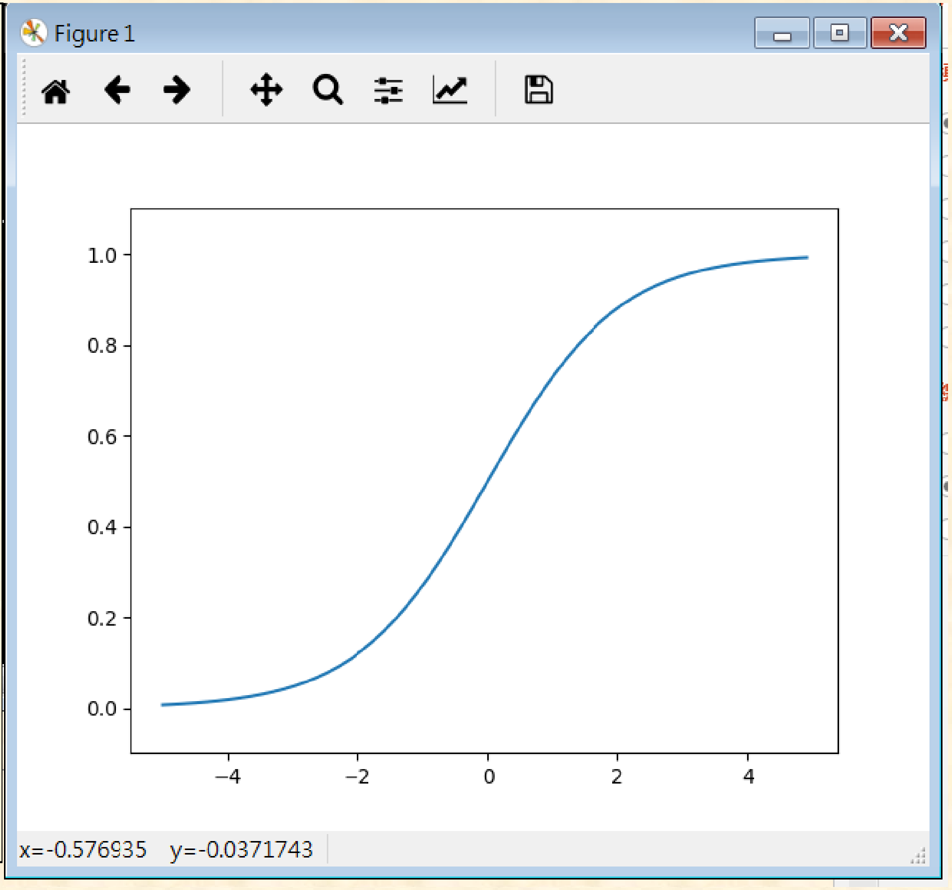
```
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def step_function(x):
    return np.array(x > 0,
dtype=np.int)

X = np.arange(-5.0, 5.0, 0.1)
Y = step_function(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)  # 設定y軸的範圍
plt.show()
```



Python執行結果

執行Sigmoid活化函數

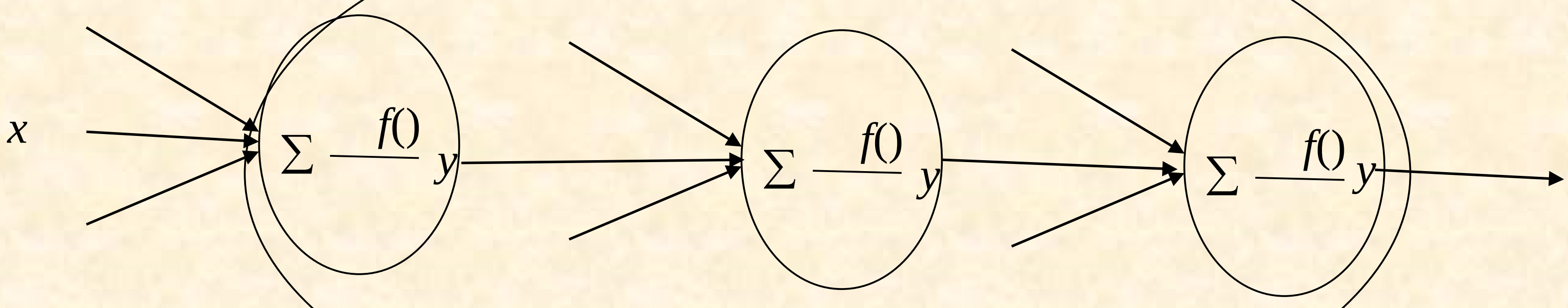
```
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```



Python執行結果

非線性活化函數

- 階梯及Sigmoid都屬於非線性函數
- 神經網路必須使用非線性函數，否則多層的神經網路分類功效會很差
- 假如使用線性函數如: $f(x) = cx$，c是一個常數
  - 架設3層神經網路

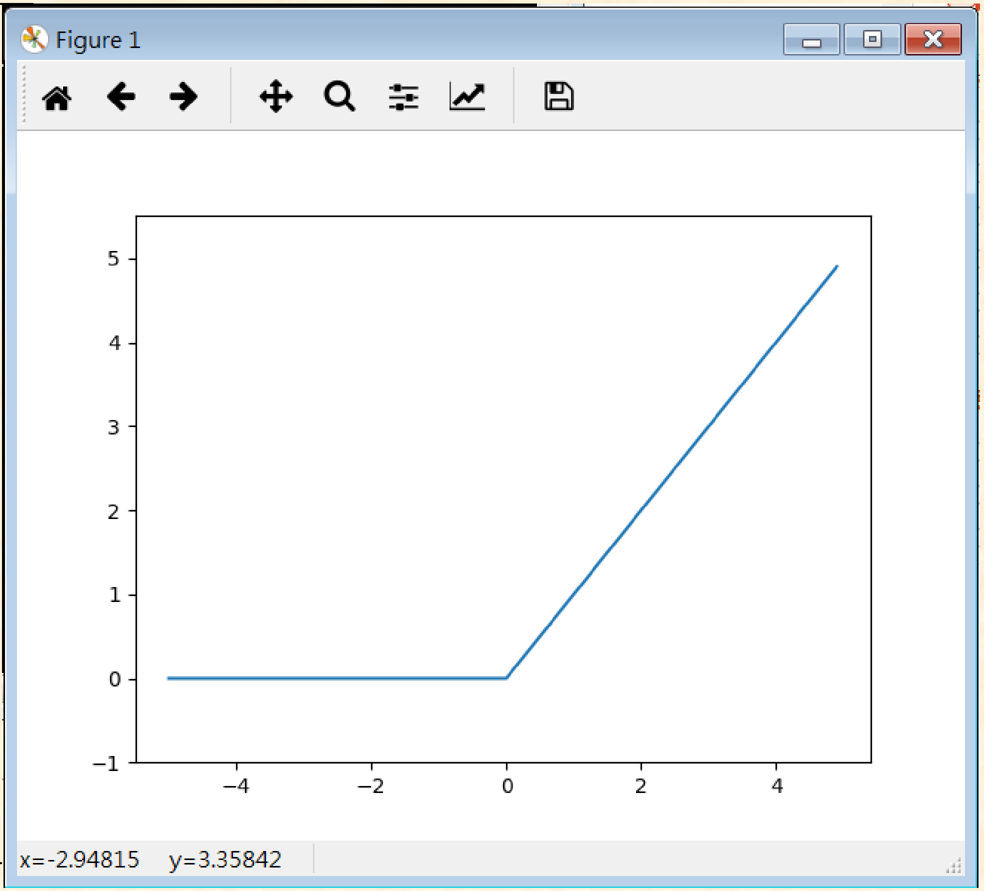


$$f' = f(f(f(cx))) = c^3x$$

效果跟只有1層之神經網路一樣

深度學習常使用ReLU活化函數: $f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$

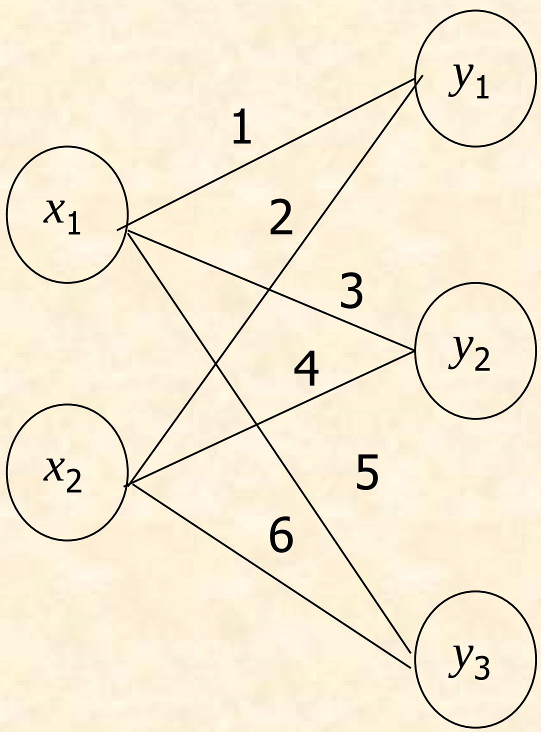
```
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def relu(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.ylim(-1.0, 5.5)
plt.show()
```



Python執行結果

神經網路的乘積

- 使用NumPy矩陣執行神經網路



$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

X	W	=	Y
2	2x3		3

一致　　一致

神經網路的乘積

❖ 使用NumPy矩陣執行神經網路

```
# coding: utf-8
import numpy as np
X = np.array([1, 2])
X.shape
W = np.array([[1, 3, 5],[2, 4, 6]])
print(W)
W.shape
Y= np.dot(X, W)
print(Y)
```

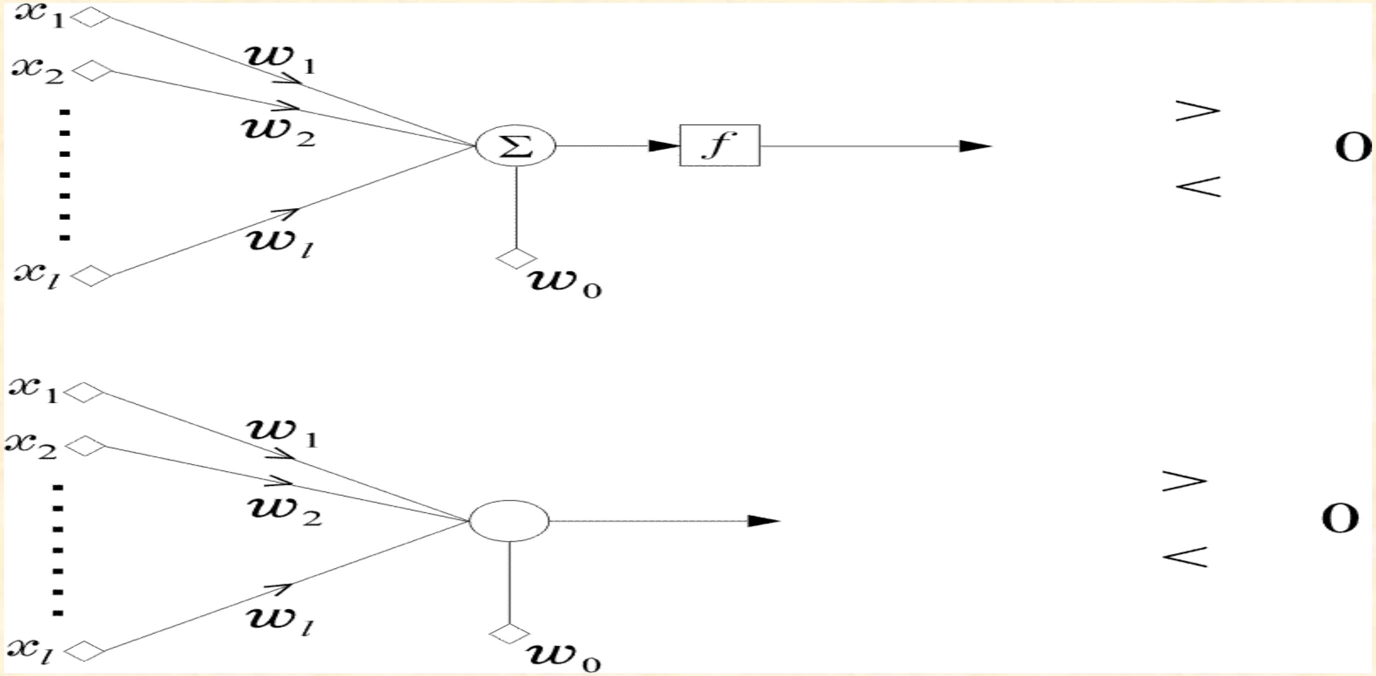
python

```
>>>
>>>
>>> # coding: utf-8
... import numpy as np
>>> X = np.array([1, 2])
>>> X.shape
(2,)
>>> W = np.array([[1, 3, 5],[2, 4, 6]])
>>> print(W)
[[1 3 5]
 [2 4 6]]
>>> W.shape
(2, 3)
>>> Y= np.dot(X, W)
>>> print(Y)
[ 5 11 17]
>>>
```

Training Algorithms for Linear Classifiers

Perceptron (感知器)

Training Perceptron



$w_i's$ synapses or synaptic weights

w_0 threshold

- The network is called perceptron or neuron.
- It is a learning machine that learns from the training vectors via the perceptron algorithm.

The Perceptron Algorithm

- Assume linearly separable classes, i.e.,

$$\exists\ \underline{w}^*:\ \underline{w}^{*T}\ \underline{x} > 0\ \ \forall \underline{x} \in \omega_1$$

$$\underline{w}^{*T}\ \underline{x} < 0\ \ \forall \underline{x} \in \omega_2$$

- The case $\underline{w}^{*T}\ \underline{x} + w_0^*$
  falls under the above formulation, since

  - $\underline{w}' \equiv \begin{bmatrix} \underline{w}^* \\ w_0^* \end{bmatrix},\ \underline{x}' = \begin{bmatrix} \underline{x} \\ 1 \end{bmatrix}$

  - $\underline{w}^{*T}\ \underline{x} + w_0^* = \underline{w}'^{T}\ \underline{x}' = 0$

The Perceptron Algorithm (cont.)

- Our goal: Compute a solution, i.e., a hyperplane $\underline{w}$, so that

$$\underline{w}^T \underline{x} (><) 0 \quad \underline{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases}$$

  - The steps
    - Define a cost function to be minimized.
    - Choose an algorithm to minimize the cost function.
    - The minimum corresponds to a solution.

- The Cost Function

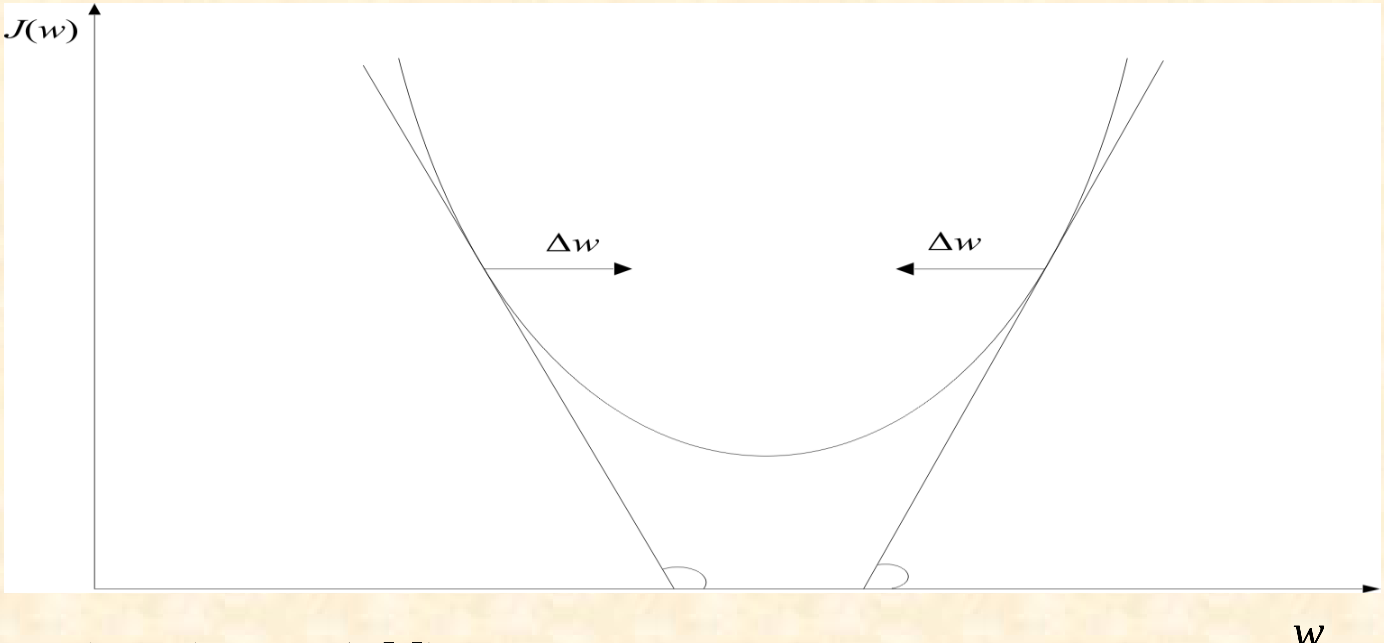
$$J(\underline{w}) = \sum_{\underline{x} \in Y} (\delta_x \underline{w}^T \underline{x})$$

- where Y is the subset of the vectors wrongly classified by $\underline{w}$. When Y=O (empty set) a solution is achieved and
- $J(\underline{w}) = 0$
- $\delta_x = -1$ if $\underline{x} \in Y$ and $\underline{x} \in \omega_1$

  $\delta_x = +1$ if $\underline{x} \in Y$ and $\underline{x} \in \omega_2$
- $J(\underline{w}) \geq 0$

- $J(\underline{w})$ is piecewise linear (WHY?)

➢ The Algorithm
- The philosophy of the gradient descent is adopted.



$$\underline{w}(\text{new}) = \underline{w}(\text{old}) + \Delta\underline{w}$$

$$\Delta\underline{w} = -\mu \frac{\partial J(\underline{w})}{\partial \underline{w}} \Big|\, \underline{w} = \underline{w}(\text{old})$$

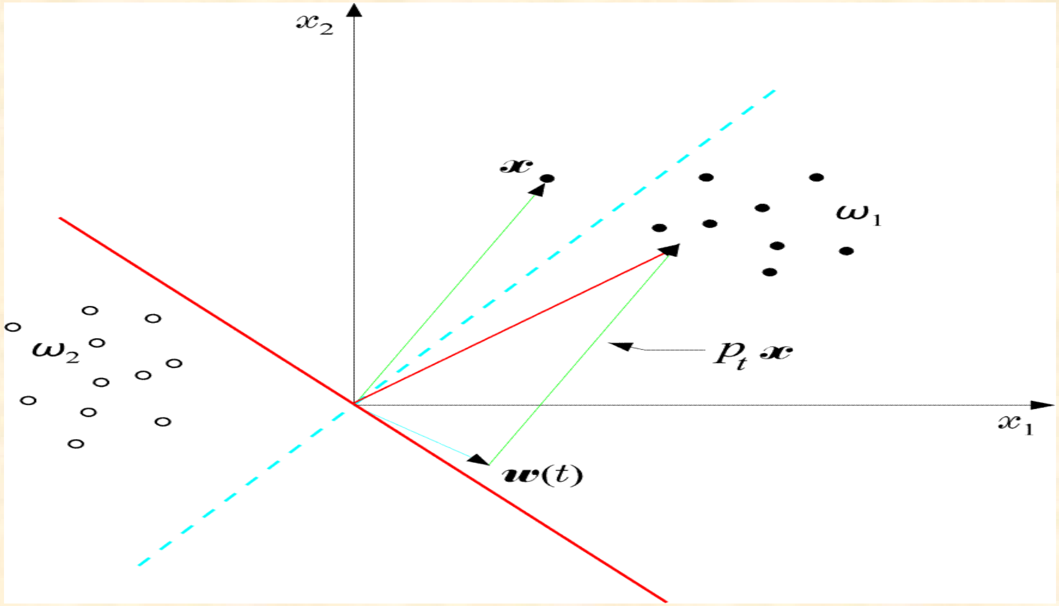
- Wherever valid

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} \Big(\sum_{\underline{x} \in Y} \delta_x \underline{w}^T \underline{x} \Big) = \sum_{\underline{x} \in Y} \delta_x \underline{x}$$

- $$\boxed{\underline{w}(t+1) = \underline{w}(t) - \rho_t \sum_{\underline{x} \in Y} \delta_x \underline{x}}$$

This is the celebrated Perceptron Algorithm.

- An example:



$$\underline{w}(t+1) = \underline{w}(t) + \rho_t \underline{x}$$
$$= \underline{w}(t) - \rho_t \delta_x \underline{x} \quad (\delta_x = -1)$$

- The perceptron algorithm **converges** in a **finite** number of iteration steps to a solution if

$$\lim_{t\to\infty} \sum_{k=0}^{t} \rho_k \to \infty, \lim_{t\to\infty} \sum_{k=0}^{t} \rho_k{}^2 < +\infty$$

$$\text{e.g.,: } \rho_t = \frac{c}{t}$$

❖ A useful variant of the perceptron algorithm

$$\underline{w}(t+1) = \underline{w}(t) + \rho \underline{x}_{(t)}, \quad \begin{array}{c} \underline{w}^T(t)\underline{x}_{(t)} \leq 0 \\ \underline{x}_{(t)} \in \omega_1 \end{array}$$

$$\underline{w}(t+1) = \underline{w}(t) - \rho \underline{x}_{(t)}, \quad \begin{array}{c} \underline{w}^T(t)\underline{x}_{(t)} \geq 0 \\ \underline{x}_{(t)} \in \omega_2 \end{array}$$

$$\underline{w}(t+1) = \underline{w}(t) \quad \text{otherwise}$$

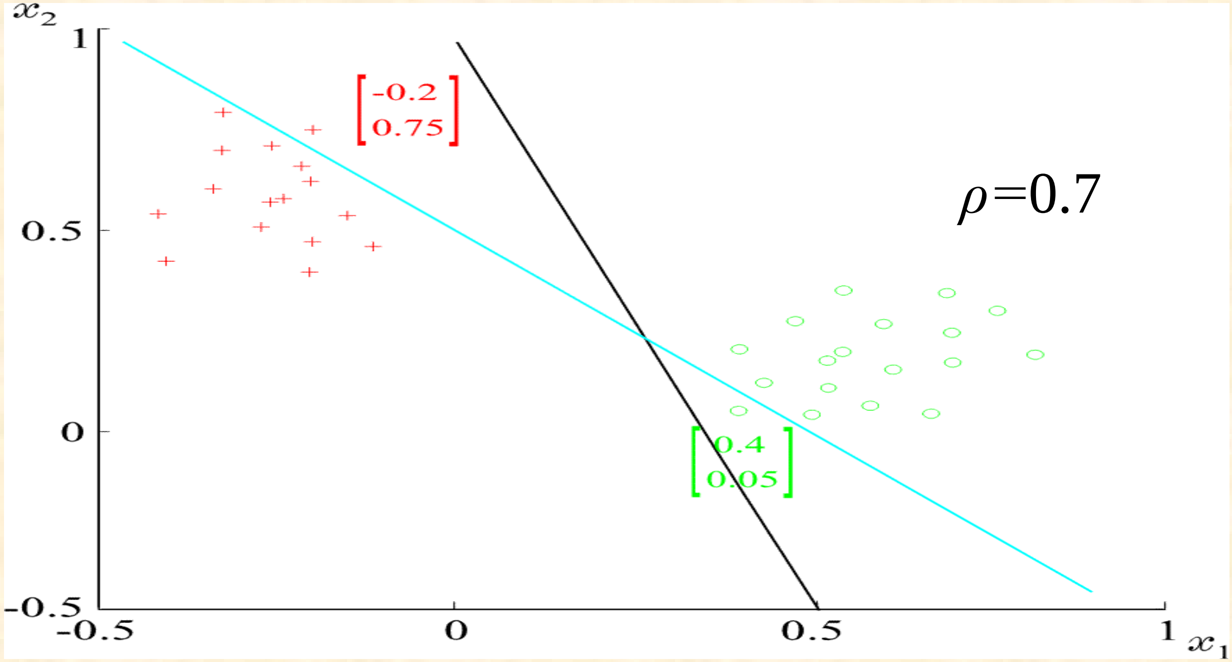
- It is a reward and punishment type of algorithm.

- **Example:** At some stage t the perceptron algorithm results in

$$w_1 = 1,\ w_2 = 1,\ w_0 = -0.5$$

$$x_1 + x_2 - 0.5 = 0$$

The corresponding hyperplane is



$$\underline{w}(t+1) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix} - 0.7(-1)\begin{bmatrix} 0.4 \\ 0.05 \\ 1 \end{bmatrix} - 0.7(+1)\begin{bmatrix} -0.2 \\ 0.75 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.42 \\ 0.51 \\ -0.5 \end{bmatrix}$$

Least Squares Methods

- If classes are linearly separable, the perceptron output results in ± 1
- If classes are NOT linearly separable, we shall compute the weights, $w_1, w_2, \ldots, w_0,$ so that the difference between
  - The actual output of the classifier, $\underline{w}^T \underline{x}$, and
  - The desired outputs, e.g.,

$$+1 \text{ if } \underline{x} \in \omega_1$$

$$-1 \text{ if } \underline{x} \in \omega_2$$

to be SMALL.

Remarks on Least Squares Methods

- SMALL, in the mean square error sense, means to choose $\underline{w}$ so that the cost function:

$$J(\underline{w}) \equiv E[(y - \underline{w}^T \underline{x})^2] \text{ becomes minimum.}$$

$$\hat{\underline{w}} = \arg\min_{\underline{w}} J(\underline{w})$$

y is the corresponding desired response.

- Minimizing

$J(\underline{w})$ w.r.t. to $\underline{w}$ results in :

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} E[(y - \underline{w}^T x)^2] = 0$$

$$= 2E[\underline{x}(y - \underline{x}^T \underline{w})] \Rightarrow$$

$$E[\underline{x}\underline{x}^T]\underline{w} = E[\underline{x}y] \Rightarrow$$

$$\boxed{\hat{\underline{w}} = R_x^{-1} E[\underline{x}y]}$$

where R_x is the autocorrelation matrix

$$R_x \equiv E[\underline{x}\underline{x}^T] = \begin{bmatrix} E[x_1 x_1] & E[x_1 x_2] \ldots & E[x_1 x_l] \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\ E[x_l x_1] & E[x_l x_2] \ldots & E[x_l x_l] \end{bmatrix}$$

and $E[\underline{x}y] = \begin{bmatrix} E[x_1 y] \\ \ldots \\ E[x_l y] \end{bmatrix}$ the crosscorrelation vector.

Multi-class generalization

- The goal is to compute M linear discriminant functions:

$$g_i(\underline{x}) = \underline{w}_i^T \underline{x}$$

according to the MSE.

- Adopt as desired responses y_i:

$$y_i = 1 \quad \text{if} \quad \underline{x} \in \omega_i$$

$$y_i = 0 \quad \text{otherwise}$$

- Let $\qquad \underline{y} = \left[y_1, y_2, \ldots, y_M\right]^T$

- And the matrix $\quad W = \left[\underline{w}_1, \underline{w}_2, \ldots, \underline{w}_M\right]$

- The goal is to compute W:

$$\hat{W} = \arg\min_{W} E\left[\left\| \underline{y} - W^T \underline{x} \right\|^2\right] = \arg\min_{W} E\left[\sum_{i=1}^{M} \left(y_i - \underline{w}_i^T \cdot \underline{x}\right)^2\right]$$

- The above is equivalent to a number M of MSE minimization problems. That is:

  Design each $\underline{w}_i$ so that its desired output is 1 for $\underline{x} \in \omega_i$ and 0 for any other class.

➢ Remark: The MSE criterion belongs to a more general class of cost function with the following important property:

- The value of $g_i(\underline{x})$ is an estimate, in the MSE sense, of the a-posteriori probability $P(\omega_i \mid \underline{x})$, provided that the desired responses used during training are $y_i = 1, \underline{x} \in \omega_i$ and 0 otherwise.

- **Mean square error regression**: Let $\underline{y} \in \Re^M$, $\underline{x} \in \Re^\ell$ be jointly distributed random vectors with a joint pdf $p(\underline{x}, \underline{y})$
  - The goal: Given the value of $\underline{x}$, estimate the value of $\underline{y}$. In the pattern recognition framework, given $\underline{x}$ one wants to estimate the respective label $y = \pm 1$.
  - The MSE estimate $\hat{\underline{y}}$ of $\underline{y}$, given $\underline{x}$, is defined as:

$$\hat{\underline{y}} = \arg\min_{\tilde{y}} E\left[\left\| y - \tilde{y} \right\|^2\right]$$

  - It turns out that:

$$\hat{\underline{y}} = E\left[\underline{y} \mid \underline{x}\right] \equiv \int_{-\infty}^{+\infty} \underline{y}\, p(\underline{y} \mid \underline{x})\, d\underline{y}$$

  The above is known as the regression of $\underline{y}$ given $\underline{x}$ and it is, in general, a non-linear function of $\underline{x}$. If $p(\underline{x}, \underline{y})$ is Gaussian the MSE regressor is linear.

- SMALL in the sum of error squares sense means

  - $$J(\underline{w}) = \sum_{i=1}^{N} (y_i - \underline{w}^T \underline{x}_i)^2$$

    $(y_i, \underline{x}_i)$: training pairs that is, the input $\underline{x}_i$ and its corresponding class label y_i (± 1).

  - $$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} \sum_{i=1}^{N} (y_i - \underline{w}^T \underline{x}_i)^2 = 0 \Rightarrow$$

$$\boxed{(\sum_{i=1}^{N} \underline{x}_i \underline{x}_i^T)\underline{w} = \sum_{i=1}^{N} \underline{x}_i y_i}$$

❖ Pseudoinverse Matrix

- Define

$$X = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \dots \\ \underline{x}_N^T \end{bmatrix} \quad \text{(an } Nxl \text{ matrix)}$$

$$\underline{y} = \begin{bmatrix} y_1 \\ \dots \\ y_N \end{bmatrix} \quad \text{corresponding desired responses}$$

- $X^T = [\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N]$ (an lxN matrix)
- $X^T X = \sum_{i=1}^{N} \underline{x}_i \underline{x}_i^T$
- $X^T \underline{y} = \sum_{i=1}^{N} \underline{x}_i y_i$

Thus $$(\sum_{i=1}^{N} \underline{x}_i^T \underline{x}_i)\underline{\hat{w}} = (\sum_{i=1}^{N} \underline{x}_i y_i)$$

$$(X^T X)\underline{\hat{w}} = X^T \underline{y} \Rightarrow$$

$$\underline{\hat{w}} = (X^T X)^{-1} X^T \underline{y}$$

$$= X^{\neq} \underline{y}$$

$$\boxed{X^{\neq} \equiv (X^T X)^{-1} X^T}$$ Pseudoinverse of X

- Assume $N=l \quad \Rightarrow \quad X$ square and invertible. Then

$$(X^T X)^{-1} X^T = X^{-1} X^{-T} X^T = X^{-1} \Rightarrow$$

$$\boxed{X^{\neq} = X^{-1}}$$

- Assume $N>l$. Then, in general, there is no solution to satisfy all equations simultaneously:

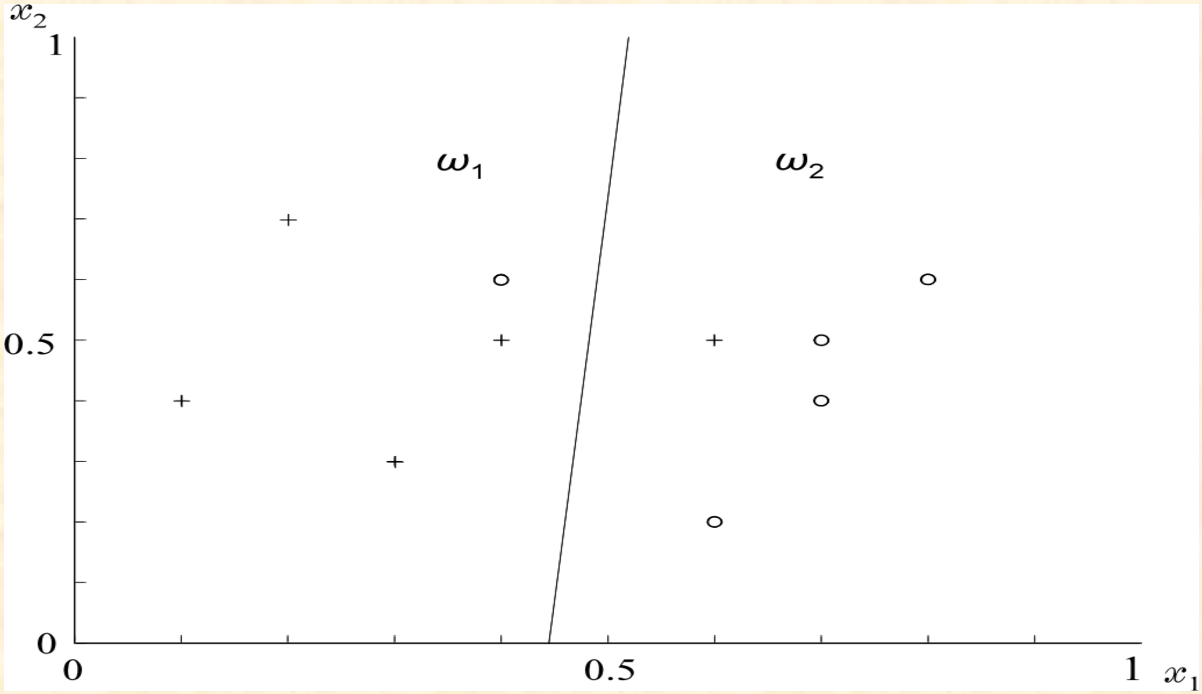
$$X\underline{w}=\underline{y}: \quad \begin{array}{c} \underline{x}_1^T\underline{w}=y_1 \\ \underline{x}_2^T\underline{w}=y_2 \\ \ldots. \\ \underline{x}_N^T\underline{w}=y_N \end{array} \qquad N \text{ equations } > l \text{ unknowns}$$

- The "solution" $\underline{w}=X^{\neq}\underline{y}$ corresponds to the minimum sum of squares solution.

- Example:

$$\omega_1 : \begin{bmatrix} 0.4 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 0.2 \\ 0.7 \end{bmatrix}, \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$$

$$\omega_2 : \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix}, \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix}$$



$$X = \begin{bmatrix} 0.4 & 0.5 & 1 \\ 0.6 & 0.5 & 1 \\ 0.1 & 0.4 & 1 \\ 0.2 & 0.7 & 1 \\ 0.3 & 0.3 & 1 \\ 0.4 & 0.6 & 1 \\ 0.6 & 0.2 & 1 \\ 0.7 & 0.4 & 1 \\ 0.8 & 0.6 & 1 \\ 0.7 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} = \underline{y}$$

- $$X^T X = \begin{bmatrix} 2.8 & 2.24 & 4.8 \\ 2.24 & 2.41 & 4.7 \\ 4.8 & 4.7 & 10 \end{bmatrix}, X^T \underline{y} = \begin{bmatrix} -1.6 \\ 0.1 \\ 0.0 \end{bmatrix}$$

$$\underline{w} = (X^T X)^{-1} X^T \underline{y} = \begin{bmatrix} -3.13 \\ 0.24 \\ 1.34 \end{bmatrix}$$

The Bias – Variance Dilemma

A classifier $g(\underline{x})$ is a learning machine that tries to predict the class label y of $\underline{x}$. In practice, a finite data set D is used for its training. Let us write $g(\underline{x}; D)$. Observe that:

- For some training sets, $D = \{(y_i, \underline{x}_i), i = 1, 2, ..., N\}$, the training may result to good estimates, for some others the result may be worse.

- The average performance of the classifier can be tested against the MSE optimal value, in the mean squares sense, that is:

$$E_D\left[\left(g(\underline{x}; D) - E[y \mid \underline{x}]\right)^2\right]$$

where E_D is the mean over all possible data sets D.

- The above is written as:

$$E_D\left[\left(g(\underline{x};D)-E[y\mid\underline{x}]\right)^2\right]=$$

$$\left(E_D\left[g(\underline{x};D)\right]-E[y\mid\underline{x}]\right)^2+E_D\left[\left(g(\underline{x};D)-E_D\left[g(\underline{x};D)\right]\right)^2\right]$$

- In the above, the first term is the contribution of the bias and the second term is the contribution of the variance.

- For a finite D, there is a trade-off between the two terms. Increasing bias it reduces variance and vice verse. This is known as the bias-variance dilemma.

- Using a complex model results in low-bias but a high variance, as one changes from one training set to another. Using a simple model results in high bias but low variance.

LOGISTIC DISCRIMINATION

- Let an M-class task, $\omega_1, \omega_2, \ldots, \omega_M$. In logistic discrimination, the logarithm of the likelihood ratios are modeled via linear functions, i.e.,

$$\ln\left(\frac{P(\omega_i \mid \underline{x})}{P(\omega_M \mid \underline{x})}\right) = w_{i,0} + \underline{w}_i^T \underline{x}, \ \ i = 1, 2, \ldots, M\text{-}1$$

- Taking into account that

$$\sum_{i=1}^{M} P(\omega_i \mid \underline{x}) = 1$$

it can be easily shown that the above is equivalent with modeling posterior probabilities as:

$$P(\omega_M \mid \underline{x}) = \frac{1}{1 + \sum_{i=1}^{M-1} \exp\left(w_{i,0} + \underline{w}_i^T \underline{x}\right)}$$

$$P(\omega_i \mid \underline{x}) = \frac{\exp\left(w_{i,0} + \underline{w}_i^T \underline{x}\right)}{1 + \sum_{i=1}^{M-1} \exp\left(w_{i,0} + \underline{w}_i^T \underline{x}\right)}, \imath = 1,2,\ldots \mathrm{M}-1$$

- For the two-class case it turns out that

$$P(\omega_2 \mid \underline{x}) = \frac{1}{1 + \exp\left(w_0 + \underline{w}^T \underline{x}\right)}$$

$$P(\omega_1 \mid \underline{x}) = \frac{\exp\left(w_0 + \underline{w}^T \underline{x}\right)}{1 + \exp\left(w_0 + \underline{w}^T \underline{x}\right)}$$

- The unknown parameters $\underline{w}_i, w_{i,0}, \ i=1, 2, \ldots, M\text{-}1$ are usually estimated by maximum likelihood arguments.

- Logistic discrimination is a useful tool, since it allows linear modeling and at the same time ensures posterior probabilities to add to one.

Training Algorithms for Linear Classifiers

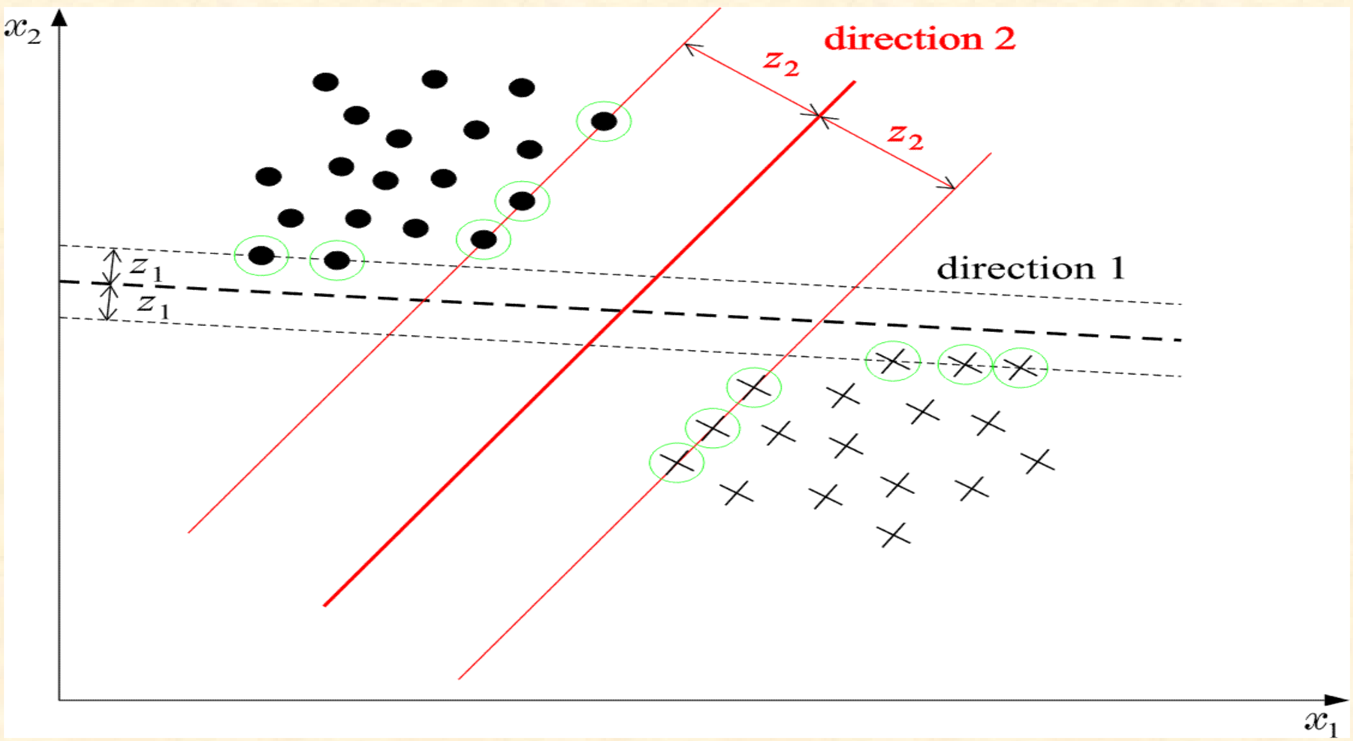
Support Vector Machine (SVM)

Support Vector Machines

- The goal: Given two linearly separable classes, design the classifier

$$g(\underline{x}) = \underline{w}^T \underline{x} + w_0 = 0$$

that leaves the maximum margin from both classes.



- Margin: Each hyperplane is characterized by:
  - Its direction in space, i.e., $\underline{w}$
  - Its position in space, i.e., w_0
  - For EACH direction, $\underline{w}$, choose the hyperplane that leaves the SAME distance from the nearest points from each class. The margin is twice this distance.

- The distance of a point $\underline{\hat{x}}$ from a hyperplane is given by:

$$z_{\hat{x}} = \frac{g(\underline{\hat{x}})}{\|\underline{w}\|}$$

- Scale, $\underline{w}, \underline{w}_0$, so that at the nearest points, from each class, the discriminant function is ± 1:

$$|g(\underline{x})| = 1 \ \{g(\underline{x}) = +1 \text{ for } \omega_1 \text{ and } g(\underline{x}) = -1 \text{ for } \omega_2\}$$

- Thus the margin is given by:

$$\frac{1}{\|\underline{w}\|} + \frac{1}{\|\underline{w}\|} = \frac{2}{\|\underline{w}\|}$$

- Also, the following is valid

$$\underline{w}^T \underline{x} + w_0 \geq 1 \ \forall \underline{x} \in \omega_1$$

$$\underline{w}^T \underline{x} + w_0 \leq -1 \ \forall \underline{x} \in \omega_2$$

- SVM (linear) classifier

$$g(\underline{x}) = \underline{w}^T \underline{x} + w_0$$

- Minimize

$$J(\underline{w}) = \frac{1}{2} \|\underline{w}\|^2$$

- Subject to

$$y_i(\underline{w}^T \underline{x}_i + w_0) \geq 1, \; i = 1,2,\ldots,N$$

$$y_i = 1, \text{for } \underline{x}_i \in \omega_i \,,$$

$$y_i = -1, \text{for } \underline{x}_i \in \omega_2$$

- The above is justified since by minimizing $\|\underline{w}\|$ the margin $\frac{2}{\|w\|}$ is maximised.

- The above is a quadratic optimization task, subject to a set of linear inequality constraints. The Karush-Kuhh-Tucker conditions state that the minimizer satisfies:

  - (1) $\frac{\partial}{\partial \underline{w}} \mathrm{L}(\underline{w}, w_0, \underline{\lambda}) = \underline{0}$
  - (2) $\frac{\partial}{\partial w_0} L(\underline{w}, w_0, \underline{\lambda}) = 0$
  - (3) $\lambda_i \geq 0, i = 1,2,\ldots, N$
  - (4) $\lambda_i \left[y_i (\underline{w}^T \underline{x}_i + w_0) - 1 \right] = 0, i = 1,2,\ldots, N$
  - Where $L(\bullet,\bullet,\bullet)$ is the Lagrangian

$$L(\underline{w}, w_0, \underline{\lambda}) \equiv \frac{1}{2} \underline{w}^T \underline{w} - \sum_{i=1}^{N} \lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0)]$$

- The solution: from the above, it turns out that:

  - $$\underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$$

  - $$\sum_{i=1}^{N} \lambda_i y_i = 0$$

➢ Remarks:

- The Lagrange multipliers can be either zero or positive. Thus,

  – $$\underline{w} = \sum_{i=1}^{N_s} \lambda_i y_i \underline{x}_i$$

  where $N_s \leq N_0$, corresponding to positive Lagrange multipliers.

  – From constraint (4) above, i.e.,

  $$\lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0) - 1] = 0, \quad i = 1,2,\ldots,N$$

  the vectors contributing to $\underline{w}$ satisfy

  $$\underline{w}^T \underline{x}_i + w_0 = \pm 1$$

- These vectors are known as SUPPORT VECTORS and are the closest vectors, from each class, to the classifier.

- Once $\underline{w}$ is computed, w_0 is determined from conditions (4).

- The optimal hyperplane classifier of a support vector machine is UNIQUE.

- Although the solution is unique, the resulting Lagrange multipliers are not unique.

➢ Dual Problem Formulation

- The SVM formulation is a convex programming problem, with
  - Convex cost function
  - Convex region of feasible solutions
- Thus, its solution can be achieved by its dual problem, i.e.,
  - maximize$_{\underline{\lambda}}$ $L(\underline{w}, w_0, \underline{\lambda})$
  - subject to

$$\underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\underline{\lambda} \geq \underline{0}$$

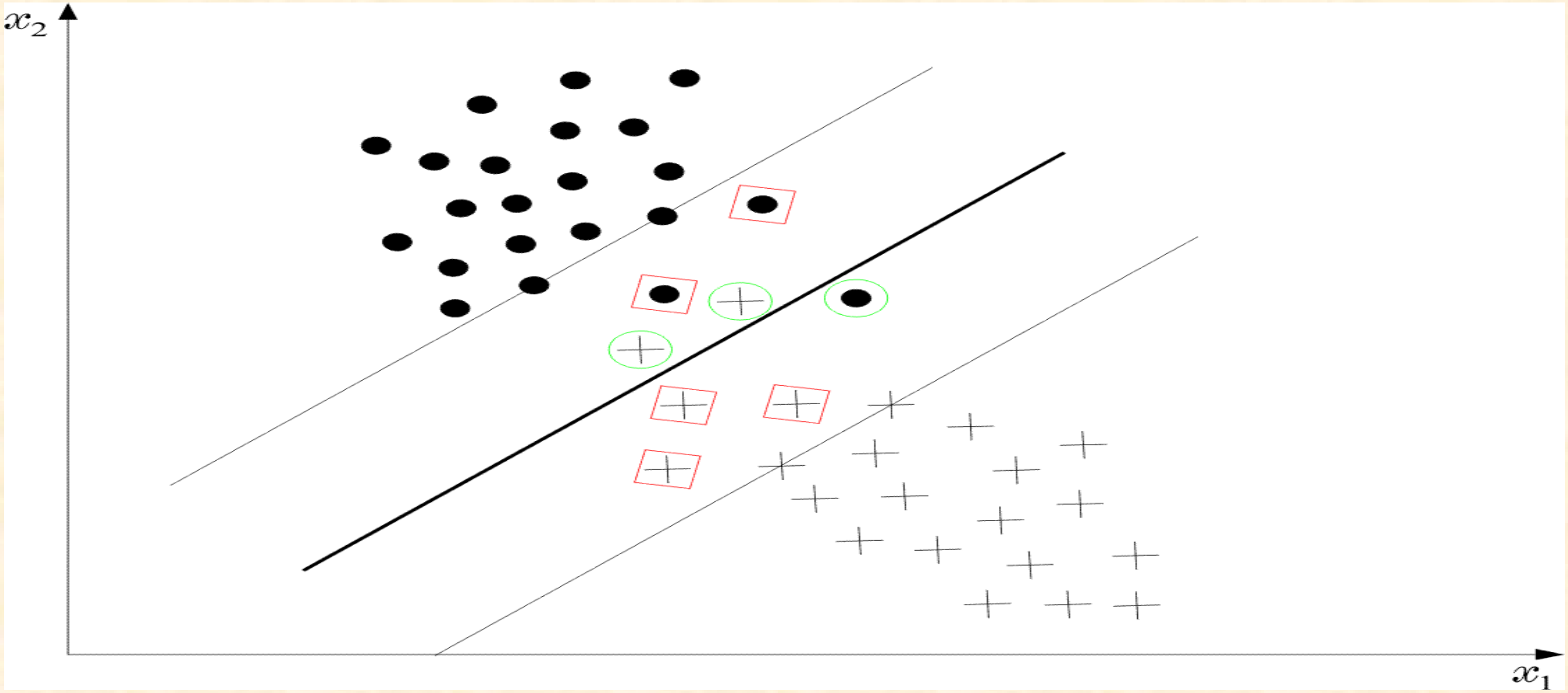
- Combine the above to obtain:
  - maximize $\underset{\underline{\lambda}}{}$ $$\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{ij}\lambda_i\lambda_j y_i y_j \underline{x}_i^T \underline{x}_j\right)$$
  - subject to

$$\sum_{i=1}^{N}\lambda_i y_i = 0$$

$$\underline{\lambda} \geq \underline{0}$$

- Remarks:
  - Support vectors enter via inner products.

- Non-Separable classes



In this case, there is no hyperplane such that:

$$\underline{w}^T \underline{x} + w_0 (><) 1, \ \forall \underline{x}$$

- Recall that the margin is defined as twice the distance between the following two hyperplanes:

$$\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}} + \boldsymbol{w}_0 = 1$$

$$\text{and}$$

$$\underline{\boldsymbol{w}}^T \underline{\boldsymbol{x}} + \boldsymbol{w}_0 = -1$$

- The training vectors belong to one of three possible categories

1) Vectors outside the band which are correctly classified, i.e.,

$$y_i(\underline{w}^T \underline{x} + w_0) > 1$$

2) Vectors inside the band, and correctly classified, i.e.,

$$0 \leq y_i(\underline{w}^T \underline{x} + w_0) < 1$$

3) Vectors misclassified, i.e.,

$$y_i(\underline{w}^T \underline{x} + w_0) < 0$$

- All three cases above can be represented as:

$$y_i(\underline{w}^T \underline{x} + w_0) \geq 1 - \xi_i$$

1) $\rightarrow \xi_i = 0$
2) $\rightarrow 0 < \xi_i \leq 1$
3) $\rightarrow 1 < \xi_i$

ξ_i are known as slack variables.

- The goal of the optimization is now two-fold:
  - Maximize margin
  - Minimize the number of patterns with $\xi_i > 0$.

    One way to achieve this goal is via the cost

$$J(\underline{w}, w_0, \underline{\xi}) = \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

where C is a constant and

$$I(\xi_i) = \begin{Bmatrix} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{Bmatrix}$$

  - $I(.)$ is not differentiable. In practice, we use an approximation. A popular choice is:
  - $J(\underline{w}, w_0, \underline{\xi}) = \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N} \xi_i$
  - Following a similar procedure as before we obtain:

➢ KKT conditions

$$(1)\ \ \underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$$

$$(2)\ \ \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$(3)\ \ C - \mu_i - \lambda_i = 0, i = 1,2,\ldots, N$$

$$(4)\ \ \lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0) - 1 + \xi_i] = 0, \quad i = 1,2,\ldots, N$$

$$(5)\ \ \mu_i \xi_i = 0, \quad i = 1,2,\ldots, N$$

$$(6)\ \ \mu_i, \lambda_i \geq 0, \quad i = 1,2,\ldots, N$$

➢ The associated dual problem

Maximize $\underset{\underline{\lambda}}{}$ $\quad (\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \underline{x}_i^T \underline{x}_j)$

subject to

$$0 \leq \lambda_i \leq C, \; i = 1,2,\ldots,N$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

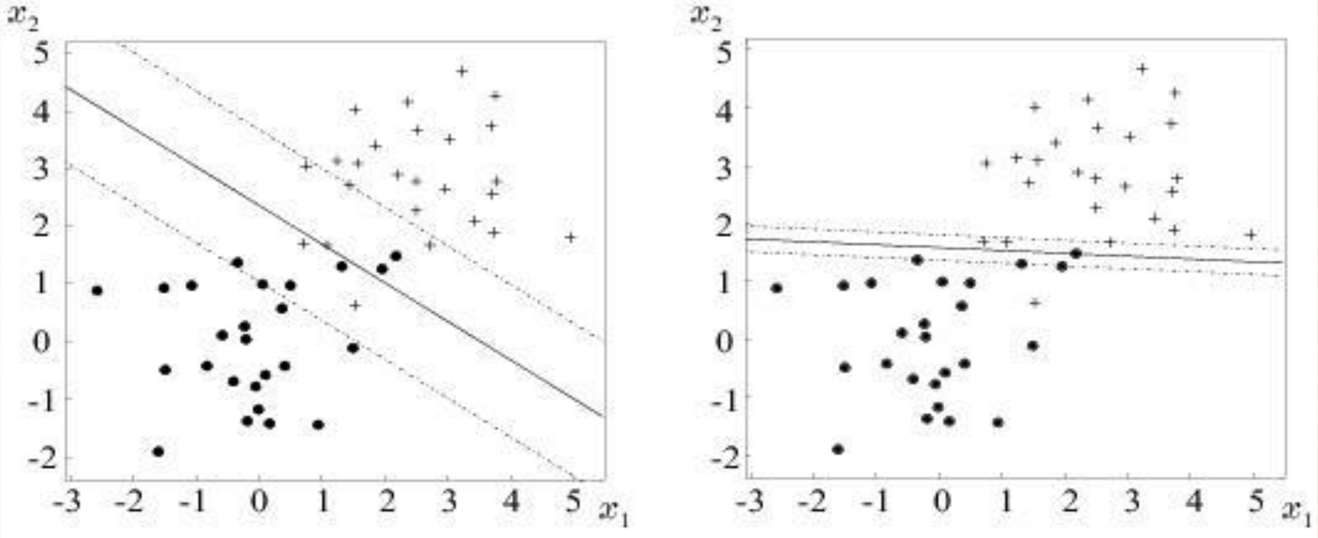
➢ Remarks: The only difference with the separable class case is the existence of C in the constraints.

- <u>Training the SVM</u>: A major problem is the high computational cost. To this end, decomposition techniques are used. The rationale behind them consists of the following:
  - Start with an arbitrary data subset (working set) that can fit in the memory. Perform optimization, via a general purpose optimizer.
  - Resulting support vectors remain in the working set, while others are replaced by new ones (outside the set) that violate severely the KKT conditions.
  - Repeat the procedure.
  - The above procedure guarantees that the cost function decreases.
  - Platt’s SMO algorithm chooses a working set of two samples, thus analytic optimization solution can be obtained.

- Multi-class generalization

  Although theoretical generalizations exist, the most popular in practice is to look at the problem as M two-class problems (one against all).

- Example:



(a) (b)

- Observe the effect of different values of C in the case of non-separable classes.

使用Scikit-learn 套件執行SVM分類器

- **Scikit-learn** 套件是**Python**用以執行機器學習各種演算法的外部套件
- 目前Scikit-learn同時支援Python 2及 3
- 執行步驟

(一)引入函式並準備**Linaer SVM**分類器

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
h = .02  # step size in the mesh

names =["Linear SVM"]
classifiers = [
  SVC(kernel="linear", C=0.025),
]
```

使用Scikit-learn 套件執行SVM分類器

(二)準備測試資料

```
# make_classification:進行分類
# n_features =2: 2維特徵
# n_informative =2: 2個類別
X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1)
# 加入適度雜訊
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
# linearly_separable: 訓練集
linearly_separable = (X, y)
#資料集合
datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable
            ]
```

使用Scikit-learn 套件執行SVM分類器

(三)測試分類器並作圖

1.外迴圈：資料迴圈。首先畫出資料分佈，接著將資料傳入分類器迴圈

```
for ds in datasets:
    # preprocess dataset, split into training and test part
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
    # and testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1
```

使用Scikit-learn 套件執行SVM分類器

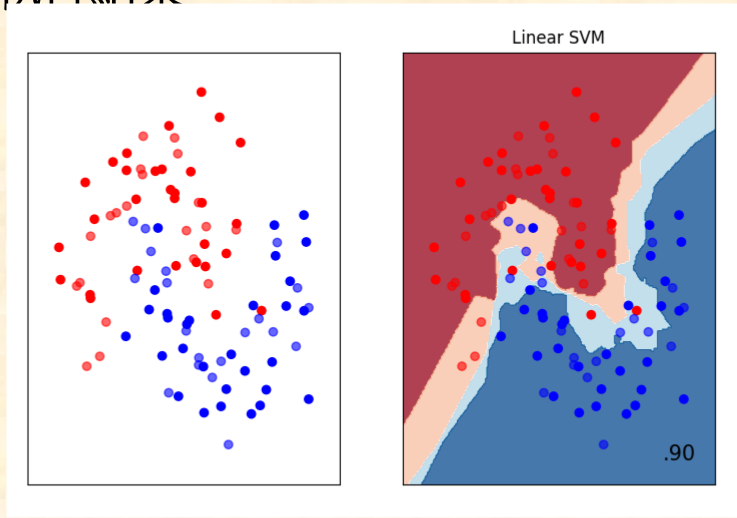
(三)測試分類器並作圖

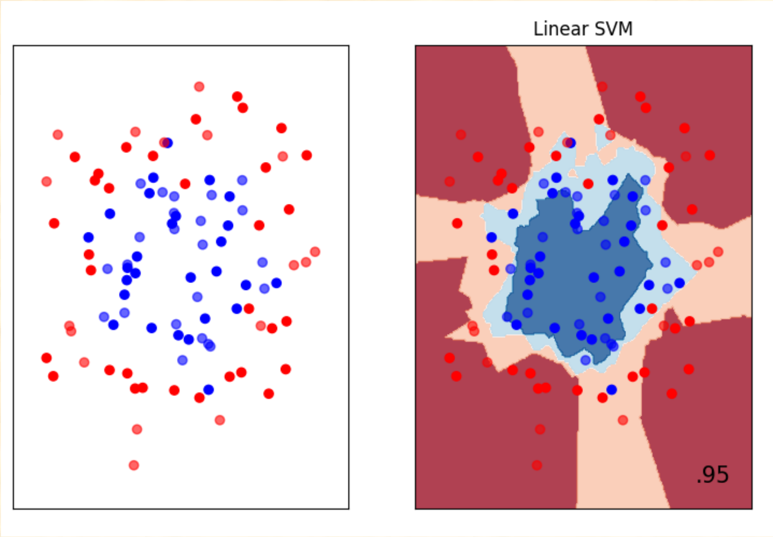
2. 內迴圈：分類器迴圈。測試分類準確度並繪製分類邊界及區域

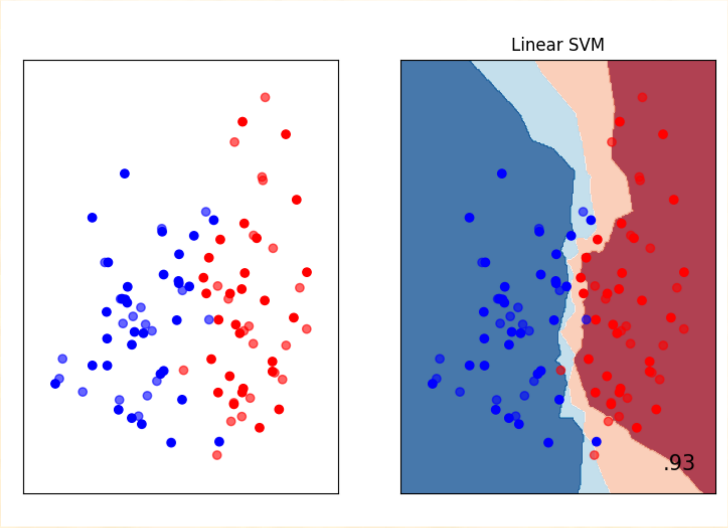
```
for name, clf in zip(names, classifiers):
        ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, m_max]x[y_min, y_max].
        if hasattr(clf, "decision_function"):
            Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

        # Plot also the training points
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
        # and testing points
        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
                   alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title(name)
        ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),
                size=15, horizontalalignment='right')
        i += 1

figure.subplots_adjust(left=.02, right=.98)
plt.show()
```

使用Scikit-learn 套件執行SVM分類器

(三)執行結果







Any Questions?