Ch 07. Linear Discriminant Function

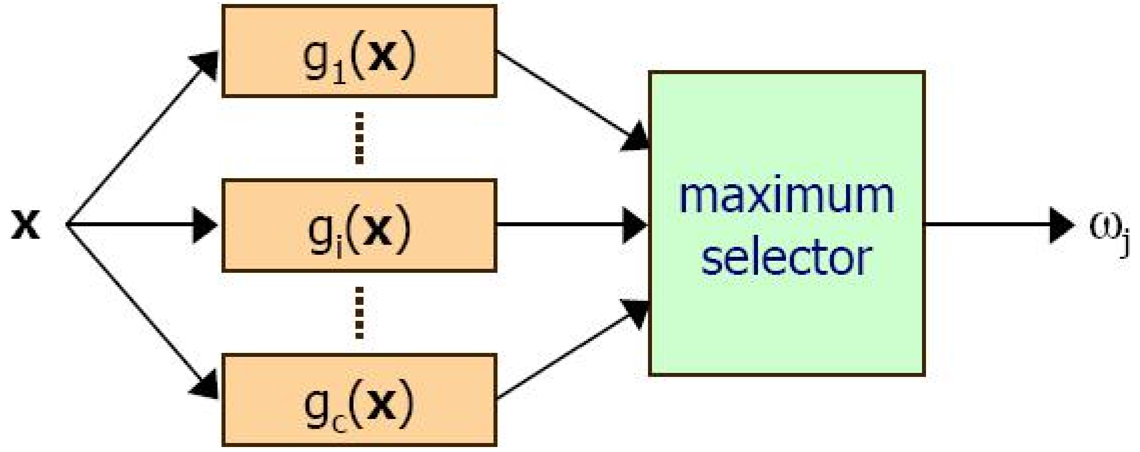
Approaches to Pattern Classification

- **Approach 1**： Estimate class-conditional probability density $p(\mathbf{x}|\omega_i)$
  - Through $p(\mathbf{x}|\omega_i)$ and $P(\omega_i)$, calculate posterior probability $\mathrm{P}(\omega_i|\mathbf{x})$ with Bayes' rule, then make decisions with maximum posterior probability
  - **Two Methods**
    - **Method 1a**： Parameter estimation of probability density
      - Based on parametric description of $p(\mathbf{x}|\omega_i)$
    - **Method 1b**： Non-Parametric estimation of probability density
      - Based on non-parametric description of $p(\mathbf{x}|\omega_i)$
- **Approach 2**： Estimate posterior probability
  - Don't have to estimate $p(\mathbf{x}|\omega_i)$ in advance
- **Approach 3**： Compute discrimination function
  - Don't have to estimate $p(\mathbf{x}|\omega_i)$ or $\mathrm{P}(\omega_i|\mathbf{x})$

Discriminant Function

- The most common expression of classifier is discriminant function $g_i(x)$, $i = 1, \ldots, c$, and each category corresponds to a discriminant function
- Decision rule based on discriminant function

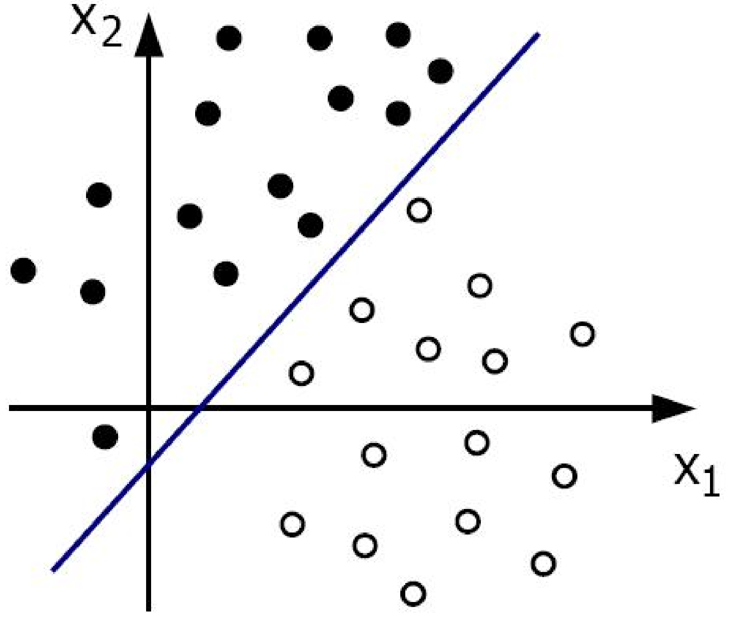
$$\boxed{\text{If } g_j(\mathbf{x}) \geq g_i(\mathbf{x}),\ \forall i \neq j, \text{ then the pattern is } \omega_j}$$



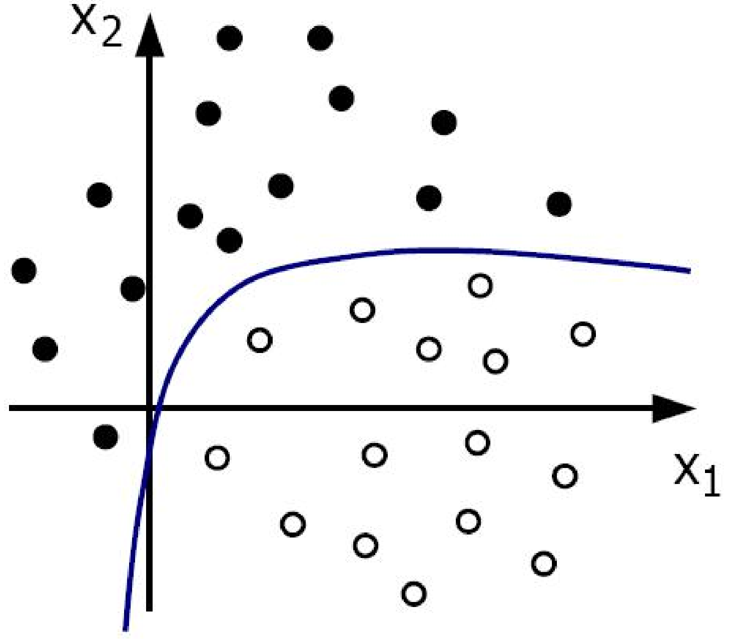
Discriminant Function

- Suppose that the discriminant function form of each category is known
- The parameters in the function can be estimated by using the training sample set
- Discriminant function example

linear discriminant function



quadratic discriminant function



Linear Discriminant Function

- Each discriminant function is a linear combination of the components of the eigenvector **x**

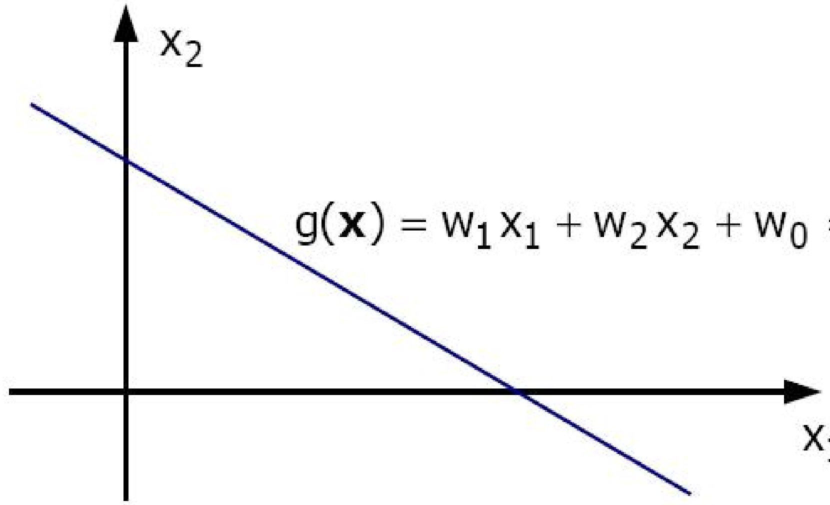
$$g(x) = \mathbf{w}^t\mathbf{x} + w_0 = w_1 \mathrm{x}_1 + w_2 \mathrm{x}_2 + \ldots + w_d \mathrm{x}_d + w_0$$

- For C categories problems, each category i corresponds to a linear discriminant function

$$g_i(x) = \mathbf{w}_i{}^t\mathbf{x} + w_{i0} = w_{i1} \mathrm{x}_1 + w_{i2} \mathrm{x}_2 + \ldots + w_{id} \mathrm{x}_d + w_{i0}$$

- Example: Two dimensional case

$$g(x) = w_1 \mathrm{x}_1 + w_2 \mathrm{x}_2 + w_0$$



Linear discriminant functions – two-category case

- Two-category of discriminant functions

$$g_1(\mathbf{x}) = w_{11}\mathrm{x}_1 + w_{12}\mathrm{x}_2 + \ldots + w_{1d}\mathrm{x}_d + w_{10}$$

$$g_2(\mathbf{x}) = w_{21}\mathrm{x}_1 + w_{22}\mathrm{x}_2 + \ldots + w_{2d}\mathrm{x}_d + w_{20}$$

- Can be achieved by a discriminant function

$$\begin{aligned} g(x) &= g_1(x) - g_2(x) \\ &= (w_{11} - w_{21})\mathrm{x}_1 + (w_{12} - w_{22})\mathrm{x}_2 + \ldots + (w_{1d} - w_{2d})\mathrm{x}_d + (w_{10} - w_{20}) \\ &= w_1\mathrm{x}_1 + w_2\mathrm{x}_2 + \ldots + w_d\mathrm{x}_d + w_0 \end{aligned}$$

- Decision rules

Decide $\begin{cases} \omega_1 & if\ g(x) \geq 0 \\ \omega_2 & otherwise \end{cases}$

- Decision surface: $g(x) = 0$

Linear discriminant functions - Multi-category case

- A c-category classifier composed of C linear discriminant functions is called linear machine (线性机)

  $g_i(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} + w_{i0} \qquad i = 1, \dots, c$

- The decision rule of linear machine is

  $$\boxed{\text{Decide } \omega_j\text{, if } g_j(\mathbf{x}) \geq g_i(\mathbf{x}),\ \forall i \neq j}$$

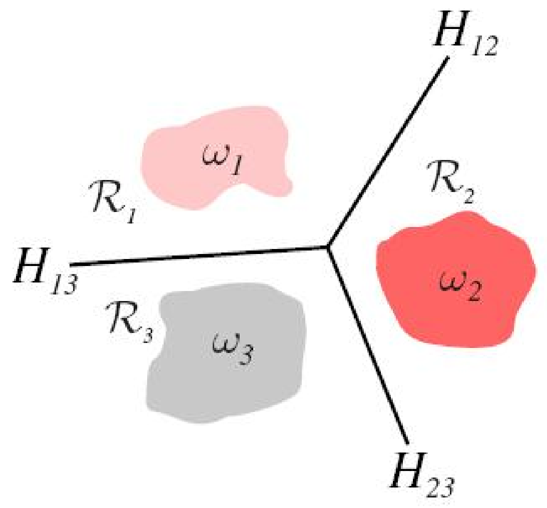
- Suppose $\mathcal{R}_i$ and $\mathcal{R}_j$ are two adjacent decision fields, then the boundary between them is part of **H_{ij}** of hyperplane, **H_{ij}** is determined by the discriminant function corresponding to $\mathcal{R}_i$ and $\mathcal{R}_j$ respectively

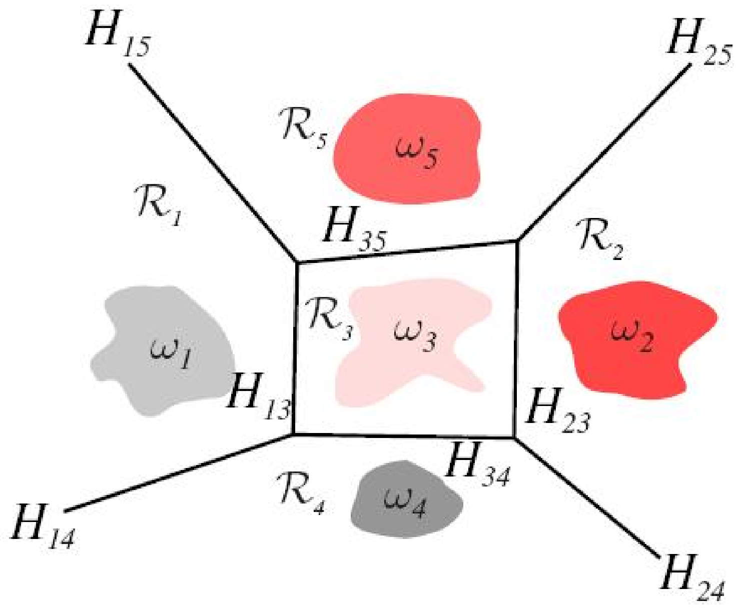
  $g_i(\mathbf{x})$= $g_i(\mathbf{x})$ or $(\mathbf{w}_i - \mathbf{w}_j)^t\mathbf{x} + \left(w_{i0} - w_{j0}\right) = 0$

  The direction of **H_{ij}** is determined by $\mathbf{w}_i - \mathbf{w}_j$

Linear discriminant functions – Multicategory case

- The decision domains and decision surfaces of a linear machine





Generalized Linear Discriminant Function

- Quadratic discriminant function （二次判别函数）

$$g(\mathbf{x}) = \omega_0 + \sum_{i=1}^{d} \omega_i x_i + \sum\nolimits_{i=1}^{d} \sum_{j=1}^{d} \omega_{ij} x_i x_j$$

- Polynomial discriminant function （多项式判别函数）

$$g(\mathbf{x}) = \omega_0 + \sum\nolimits_{i=1}^{d} \omega_i x_i + \sum\nolimits_{i=1}^{d} \sum\nolimits_{j=1}^{d} \omega_{ij} x_i x_j + \sum\nolimits_{i=1}^{d} \sum\nolimits_{j=1}^{d} \sum\nolimits_{k=1}^{d} w_{ijk} x_i x_j x_k$$

Generalized Linear Discriminant Function

- **Generalized linear discriminant function** （广义线性判别函数）

$$g(\mathbf{x}) = \sum_{i=1}^{d} a_i \boldsymbol{y}_i(\mathbf{x}) \qquad g(\mathbf{x}) = \mathbf{a}^t\mathbf{y}$$

  - $\mathbf{a}$ is $\widehat{d}$ dimensional weight vector
  - The component function $\boldsymbol{y}_i(\mathbf{x})$ can be any function of $\mathbf{x}$ and can be regarded as the result of the feature extraction subsystem
  - even though $g(\mathbf{x})$ is not a linear function of x, but it is a linear function of y, called the **generalized linear discriminant function**

Generalized Linear Discriminant Function

- Example 1
  - Quadratic discriminant function

$$g(\mathbf{x}) = a_1 + a_2 x + a_3 x^2$$

  - Define three dimensional vector y

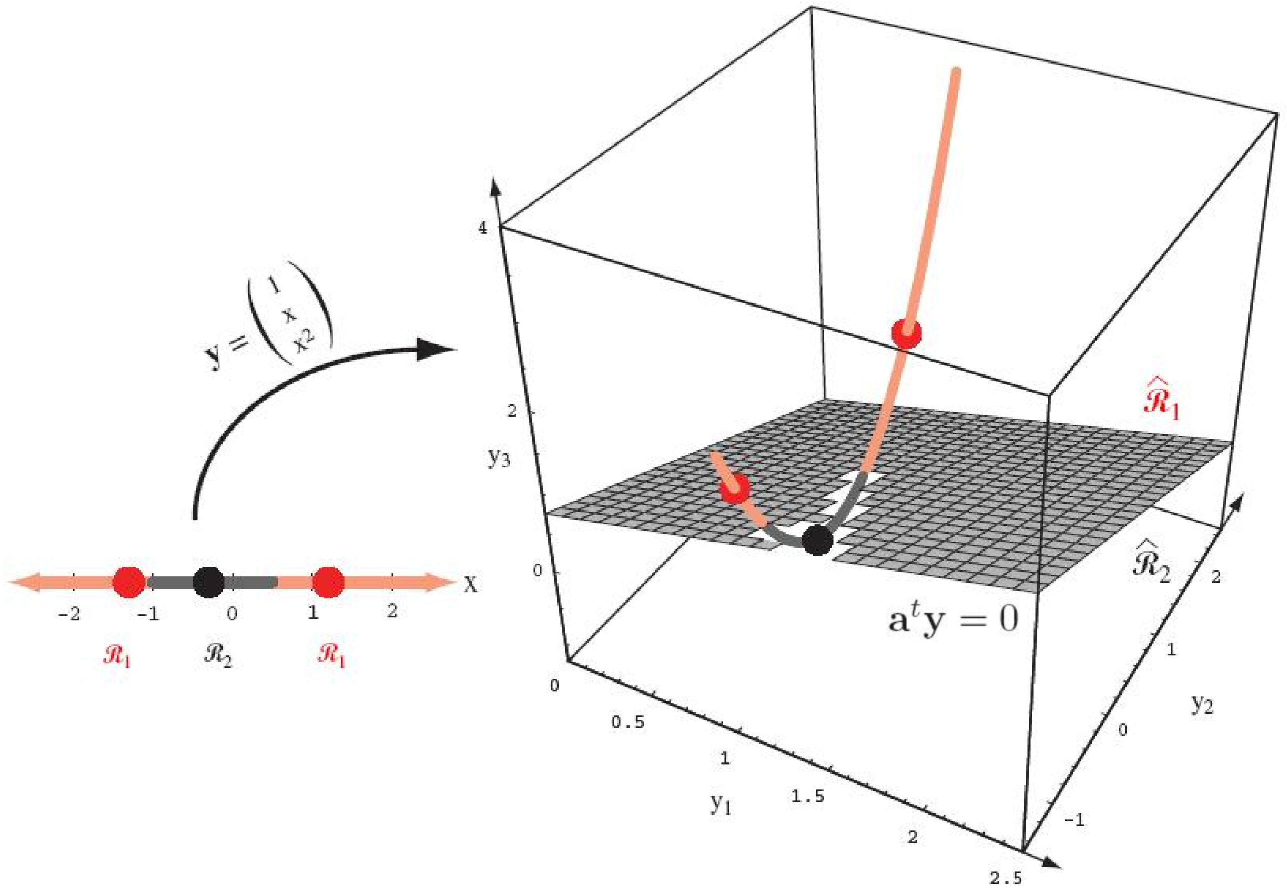
$$\mathbf{y} = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}$$

  - Then

$$g(\mathbf{x}) = a_1 y_1 + a_2 y_2 + a_2 y_3 = \mathbf{a}^t \mathbf{y}$$

Generalized Linear Discriminant Function

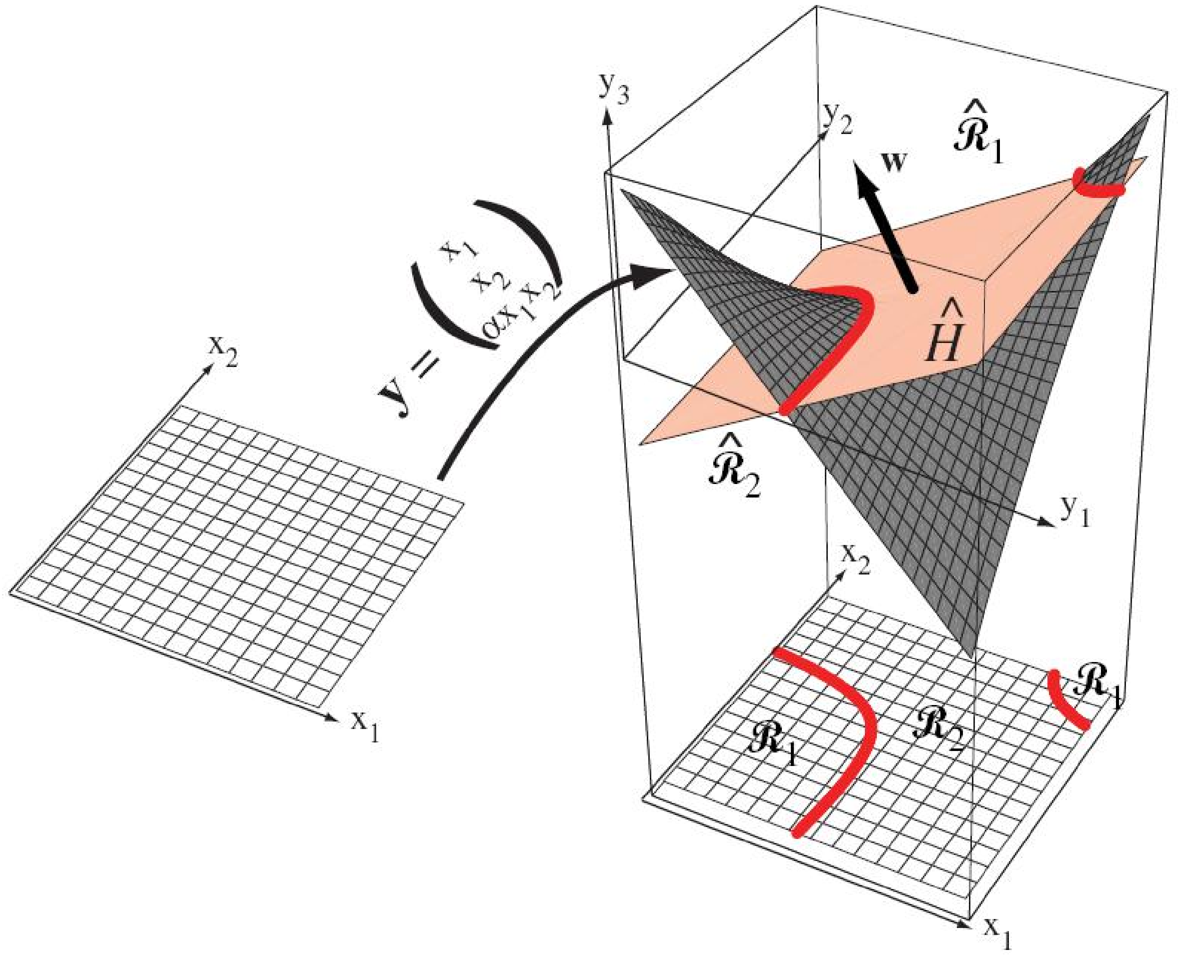
- Example 1
  - When $\mathbf{a} = (-1,1,2)^t$



$\mathbf{y} = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}$
x
-2
-1
0
1
2
$\mathcal{R}_1$
$\mathcal{R}_2$
$\mathcal{R}_1$
4
2
0
y_3
$\widehat{\mathcal{R}}_1$
$\widehat{\mathcal{R}}_2$
$\mathbf{a}^t\mathbf{y} = 0$
0
0.5
1
1.5
2
2.5
y_1
-1
0
1
2
y_2

Generalized Linear Discriminant Function

- Example 2



Generalized Linear Discriminant Function

- Linear Discriminant Function

$$g(\mathbf{x}) = \omega_0 + \sum_{i=1}^{d} \omega_i x_i = \sum_{i=0}^{d} \omega_i x_i$$

  - Augmented feature vector （增广特征向量）$\mathbf{y}$
  - Augmented weight vector （增广权向量）$\mathbf{a}$

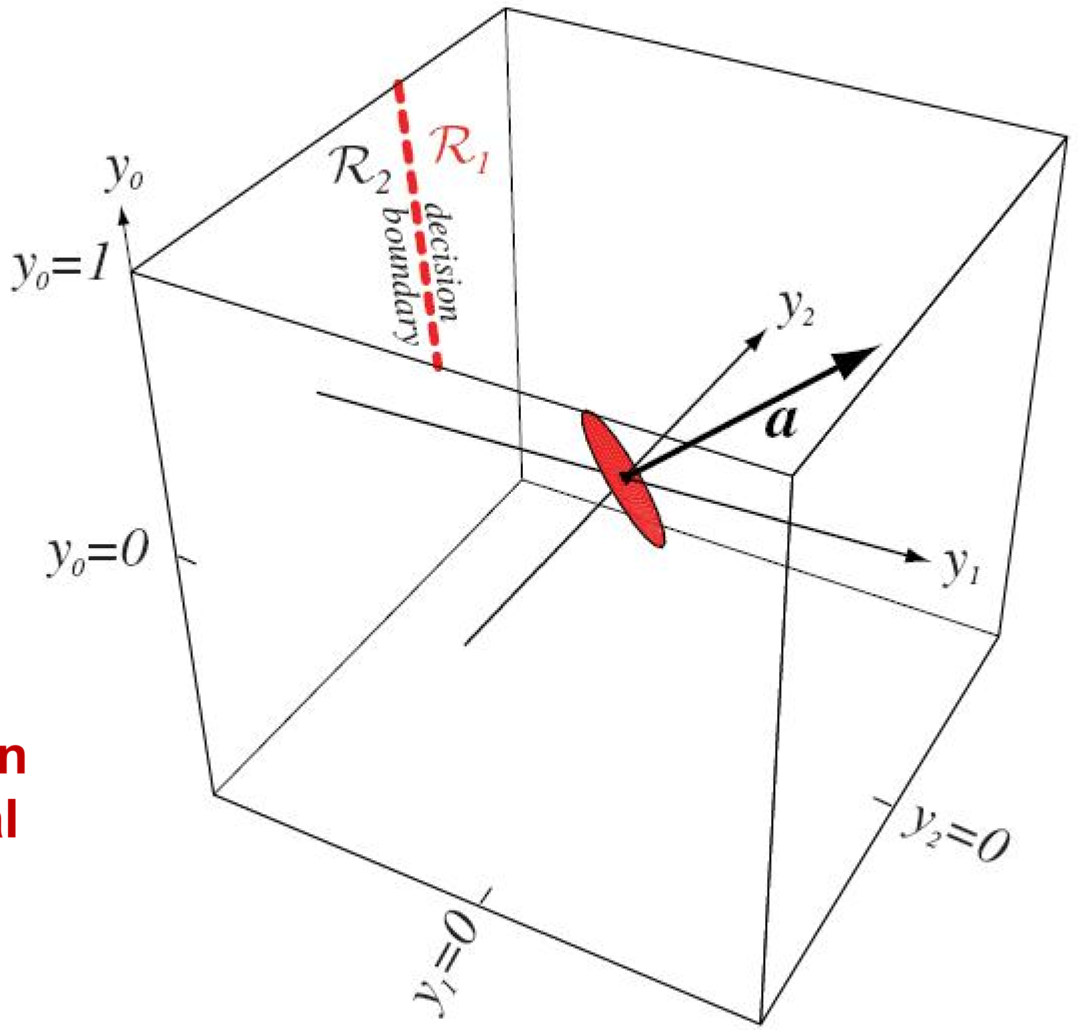
$$\mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \qquad a = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_d \end{bmatrix} = \begin{bmatrix} \omega_0 \\ \mathbf{w} \end{bmatrix}$$

  - Transform the finding of the weight vector $\mathbf{w}$ and the weight threshold ω_0 into finding weight vector $\mathbf{a}$

Generalized Linear Discriminant Function

The decision surface $\mathbf{a}^t\mathbf{y} = 0$ must pass through the origin of the augmented space

By adjusting a, a certain linear decision surface in the original space (y_0=1 plane) can be projected



Summary

- Discriminant function
  - Decision rule based on discriminant function

$$\boxed{\text{If } g_j(\mathbf{x}) \geq g_i(\mathbf{x}),\ \forall i \neq j, \text{ then the pattern is } \omega_j}$$

  - Linear discriminant function
  - Quadratic discriminant function
  - Polynomial discriminant function
- Generalized linear discriminant function

$$g(\mathbf{x}) = \sum_{i=1}^{d} a_i \boldsymbol{y}_i(\mathbf{x}) \qquad g(\mathbf{x}) = \mathbf{a}^t\mathbf{y}$$

Two-Category Linearly Separable Case

- Suppose there is a set $y_1, \ldots, y_n$ of n samples, and these samples are used to determine the weight vector $\mathbf{a}$ in the discriminant function

$$g(\mathbf{x}) = \mathbf{a}^t \mathbf{y}$$

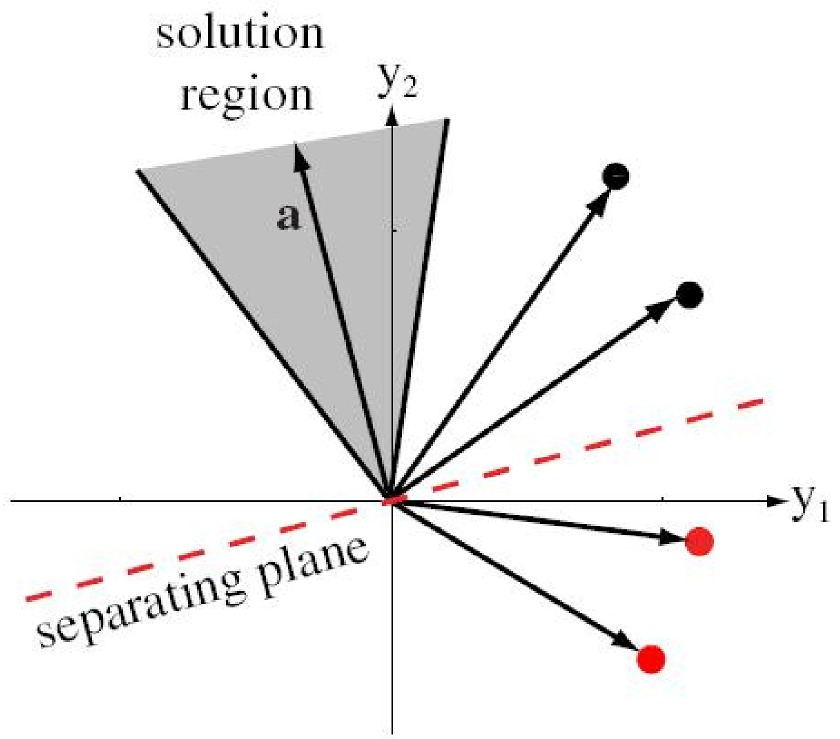
- If there is a weight vector (i.e., y is a hyperplane) in the space, all these samples can be classified correctly, then the samples is called "**linear separable**" (线性可分)
- Normalization of the two-category problem
  - For sample y_i, if $\mathbf{a}^t \boldsymbol{y}_i > 0$, then mark it as ω_1, if $\mathbf{a}^t \boldsymbol{y}_i < 0$, then mark it as ω_2
  - Normalization:
    Take negative ($\boldsymbol{y}_i \leftarrow - \boldsymbol{y}_i$) of all marked samples belonging to ω_2, then the problem can be simplified as: find a weight vector $\mathbf{a}$ that has $\mathbf{a}^t \boldsymbol{y}_i > 0$ for all samples

$\mathbf{a}$: separating vector (分离向量) or solution vector (解向量)

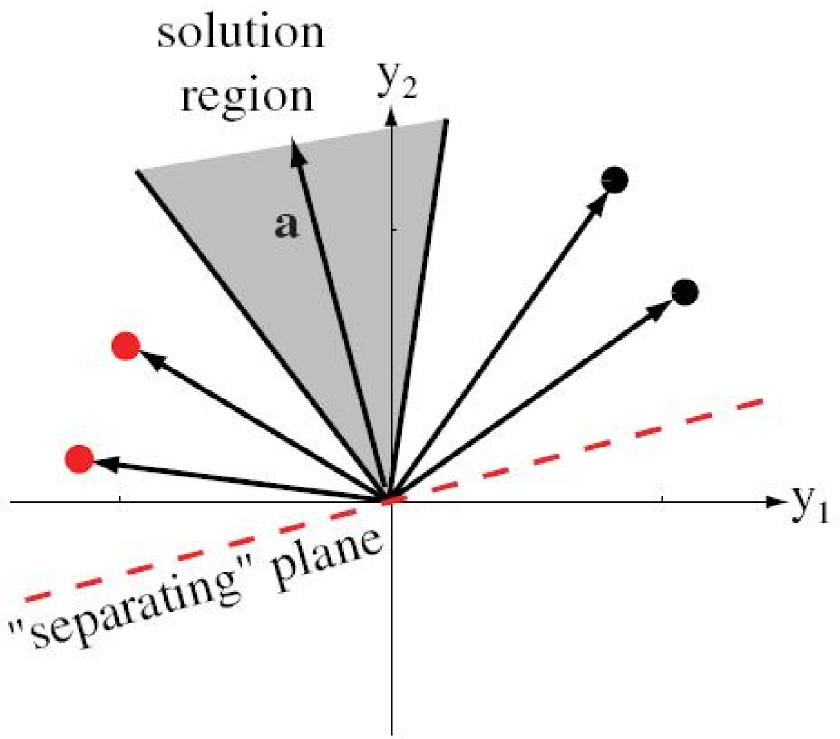
Weight Space and Solution Region

- All the possible weight vectors make up the weight space (权空间)
- The solution vector is a point in the weight space
- For each $\boldsymbol{y}_i$, $\mathbf{a}^t\boldsymbol{y}_i = 0$ determines a hyperplane that crosses the origin in the weighted space, and $\boldsymbol{y}_i$ is its normal vector
- The solution vector must be on the positive side of the hyperplane determined by each training sample, that is, the solution vector must be in the overlapping region of n positive half spaces determined by n samples, which is called “solution region“ (解区域), in which any vector is a solution vector

Weight Space and Solution Region



Before Normalized



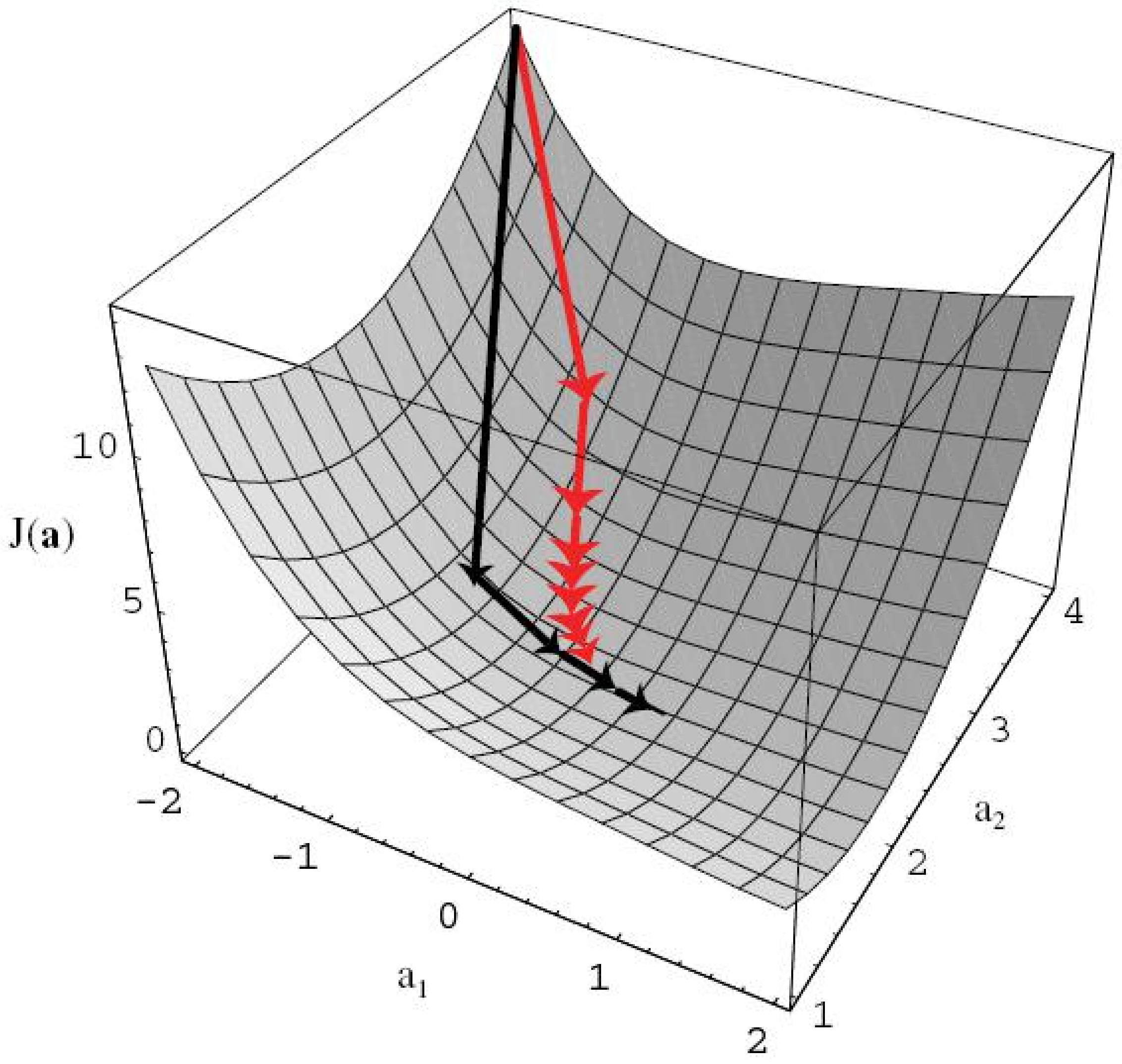
After Normalized

Gradient Descent Algorithm

- The vector can be solved by minimizing a certain criterion function J(a)
- The gradient descent method is used to solve the function minimization problem
- Basic idea:
  - Randomly choose a weight vector a(1) as the initial value
  - Compute the gradient vector $\nabla J(\mathrm{a}(1))$ of J(a) in a(1), and its negative direction represents the direction in which J(a) drops the fastest from a(1)
  - The next value a(2) is obtained by moving a distance in the negative direction of the gradient from a(1)
  - Iterative formula

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \overset{\textbf{Learning rate (学习率)}}{\boxed{\eta(k)}}\nabla J(\mathbf{a}(k))$$

Gradient Descent Algorithm



Gradient Descent Algorithm

- Basic gradient descent algorithm

```
1 begin initialize $\mathbf{a}$, criterion $\theta$, $\eta(\cdot)$, $k = 0$
2     do $k \leftarrow k + 1$
3           $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\nabla J(\mathbf{a})$
4     until $\eta(k)\nabla J(\mathbf{a}) < \theta$
5 return $\mathbf{a}$
6 end
```

Gradient Descent Algorithm

- The choice of learning rate
  - If $\eta(k)$ is too small, the algorithm converges very slowly
  - If $\eta(k)$ is too large, the algorithm might overshoot (过冲) or even fail to converge
- Can you find the optimal learning rate for each iteration?
  - $J(\mathbf{a}(k+1))$ can reach lowest point along the negative direction of the gradient

Gradient Descent Algorithm

- Optimal learning rate
  - The second order expansion of the criterion function near a(k)

$$J(\mathbf{a}) \simeq J\big(\mathbf{a}(k)\big) + \nabla J^t\big(\mathbf{a} - \mathbf{a}(k)\big) + \frac{1}{2}\big(\mathbf{a} - \mathbf{a}(k)\big)^t \mathbf{H}(\mathbf{a} - \mathbf{a}(k))$$

where ∇J is the gradient vector of J(a) in a (k), and $\mathbf{H}$ is Hessian matrix, i.e. the second partial derivative of J(a) in a(k)

$$h_{ij} = \frac{\partial^2 J}{\partial a_i \partial a_j}$$

Put into $\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J\big(\mathbf{a}(k)\big)$ get

$$J\big(\mathbf{a}(k+1)\big) \simeq J\big(\mathbf{a}(k)\big) - \eta(k)\|\nabla J\|^2 + \frac{1}{2}\eta^2(k)\nabla J^t \mathbf{H}\nabla J$$

when $\eta(k) = \boxed{\frac{\|\nabla J\|^2}{\nabla J^t \mathbf{H}\nabla J}}$, $J\big(\mathbf{a}(k+1)\big)$ is the minimum

optimal learning rate

Newton Descent Algorithm

- Directly find a to minimize

$$J(\mathbf{a}) \simeq J\big(\mathbf{a}(k)\big) + \nabla J^t\big(\mathbf{a} - \mathbf{a}(k)\big) + \frac{1}{2}\big(\mathbf{a} - \mathbf{a}(k)\big)^t \mathbf{H}(\mathbf{a} - \mathbf{a}(k))$$

  that denotes $\mathbf{a}$ as $\mathbf{a}(k+1)$

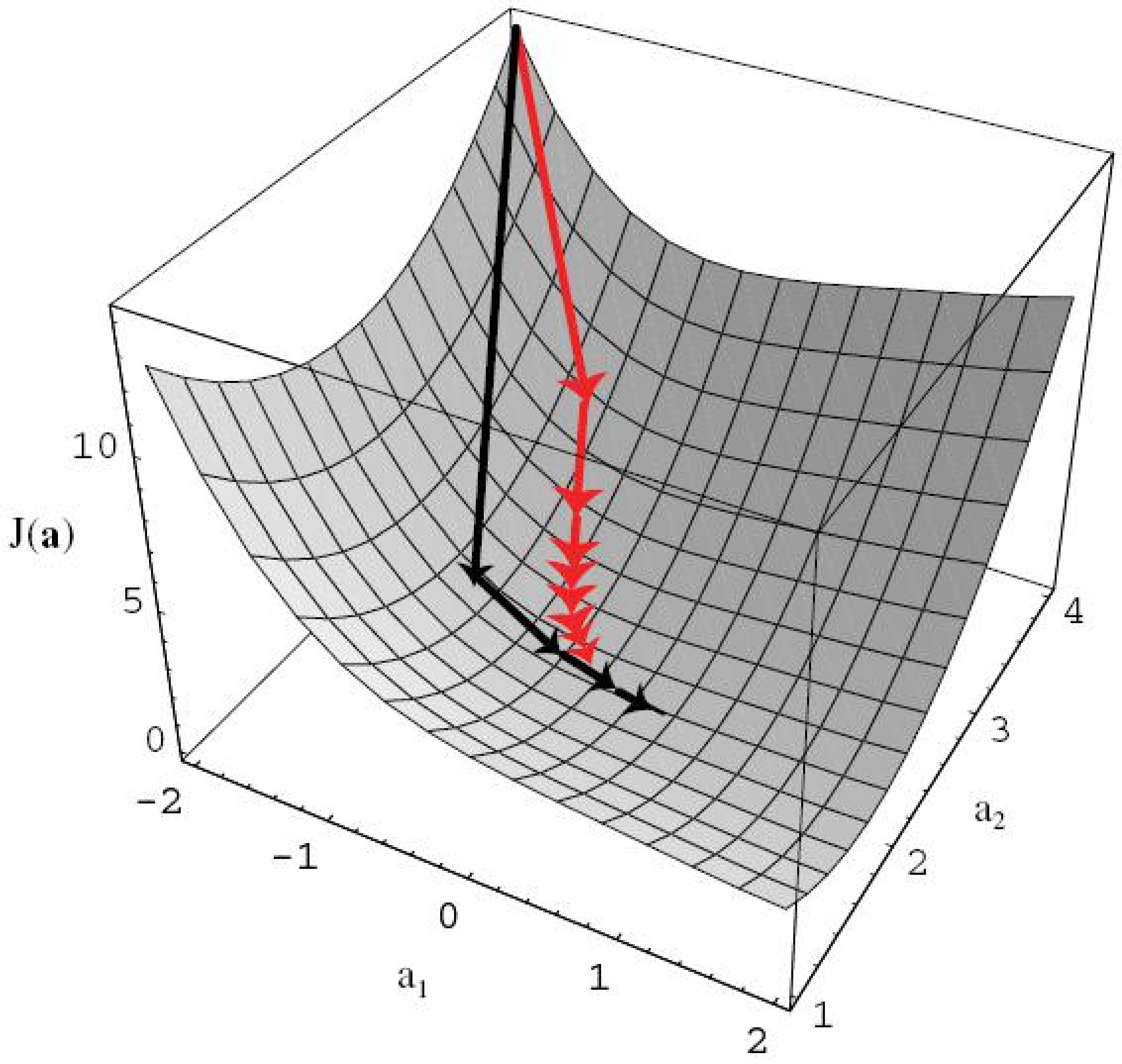
- Iterative formula $\mathbf{a}(k+1) = \mathbf{a}(k) - \mathbf{H}^{-1}\,\nabla J$
- Algorithm

```
1 begin initialize $\mathbf{a}$, criterion $\theta$
2     do
3           $\mathbf{a} \leftarrow \mathbf{a} - \mathbf{H}^{-1}\nabla J(\mathbf{a})$
4     until $\mathbf{H}^{-1}\nabla J(\mathbf{a}) < \theta$
5 return $\mathbf{a}$
6 end
```

Newton Descent Algorithm

- **Advantage**
  - The Newton descent method gives a better step size at each step than the gradient descent method
- **Disadvantage**
  - When the Hessian matrix **H** is singular, the Newton descent method cannot be applied
  - Even if **H** is not singular, the computational complexity of computing the inverse matrix of **H** is $O(d^3)$

Newton Descent Algorithm



The Choice of Learning Rate in Practice

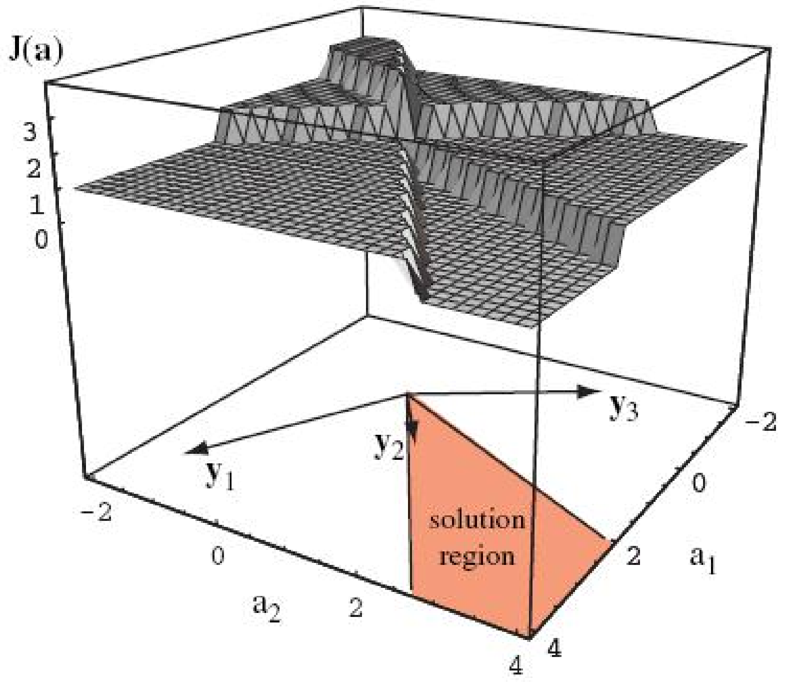
- Directly set $\eta(k)$ to some a certain enough small constant η
- Although it takes more steps to adjust than using the optimal learning rate for each step
- However, since there is no need to calculate the optimal learning rate, the total time overhead is often smaller
- **The most common choice in practice**

Perceptron Criterion Function

- How to construct the criterion function J(**a**) to solve the vector?
  - Let J(**a**) be equal to the number of samples misclassified by the decision surface determined by **a**

    Piecewise constant function, not be conducive to gradient search

    (gradient is often 0)

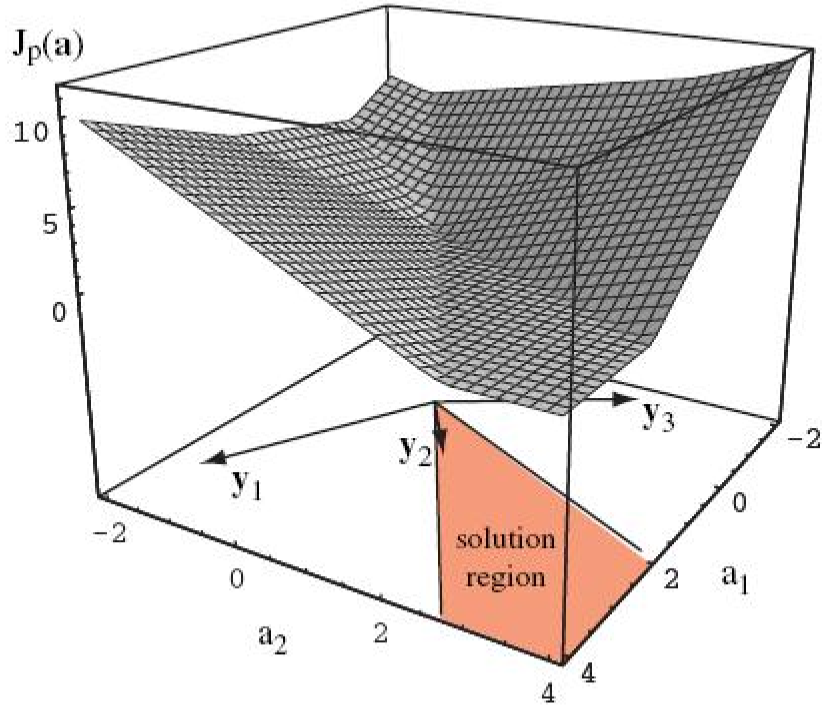


Perceptron Criterion Function

- How to construct the criterion function J(a) to solve the vector?
  - Perceptron criterion function （感知器准则函数）

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^t \mathbf{y})$$

    - where $\mathcal{Y}(\mathbf{a})$ is the sample set misclassified by **a**
    - The perceptron criterion function is proportional to the sum of the distance from the wrong sample to the decision surface



Perceptron Criterion Function Minimization

- The gradient of the perceptron criterion function

$$\nabla J_p = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{y})$$

- Iterative formula of gradient descent method

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} (-\mathbf{y})$$

where $\mathcal{Y}_k$ represents the sample set misclassified by $\mathbf{a}(k)$

Perceptron Criterion Function Minimization

- Batch perceptron algorithm

```
1 begin initialize $\mathbf{a}$, $\eta(\cdot)$, criterion $\theta$, $k = 0$
2     do $k \leftarrow k + 1$
3         $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$
4     until $|\eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}| < \theta$
5 return $\mathbf{a}$
6 end
```

Batch processing: The weight vector **a** is corrected every time, and all training samples need to be considered in "batch"

Perceptron Criterion Function Minimization

- Single sample correction
  - In the batch perceptron algorithm, all samples are examined for each correction
  - Single sample correction examines only one wrong sample at a time
  - The input sample is considered sequentially. Once a sample is found to be wrong, the current weight vector is corrected immediately
  - In order to ensure that each sample can appear in the sequence for an infinite number of times, the training samples can be kept circulating in the sequence until the algorithm converges

$$\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_1, \boldsymbol{y}_2, \dots$$

Perceptron Criterion Function Minimization

- Fixed incremental single sample correction
  - Suppose $\eta(k) = 1$
  - Sample sequence number
    - The subscript indicates the sample number

$$\downarrow \boldsymbol{y}_1, \boldsymbol{y}_2, \downarrow \boldsymbol{y}_3, \downarrow \boldsymbol{y}_1, \downarrow \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_1, \downarrow \boldsymbol{y}_2, \ldots$$

    - The superscript indicates the wrong sample number

$$\boldsymbol{y}^1, \boldsymbol{y}^2, \boldsymbol{y}^3, \boldsymbol{y}^4, \boldsymbol{y}^5, \ldots$$

i.e. $\boldsymbol{y}_1, \boldsymbol{y}_3, \boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_2, \ldots.$

    - Iterative formula

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \boldsymbol{y}^k$$

Perceptron Criterion Function Minimization

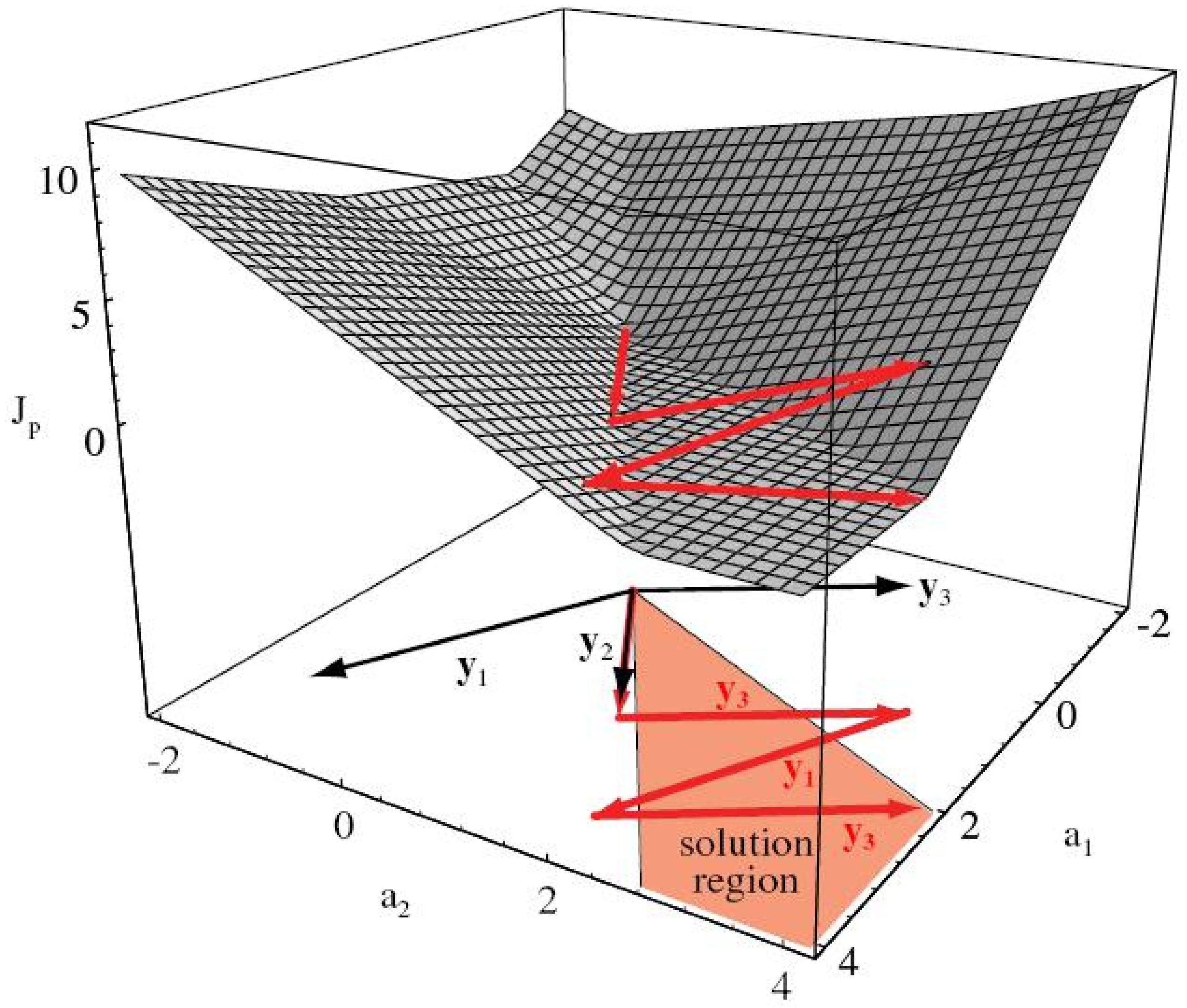
- Fixed incremental single sample perceptron algorithm

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \boldsymbol{y}^k$$

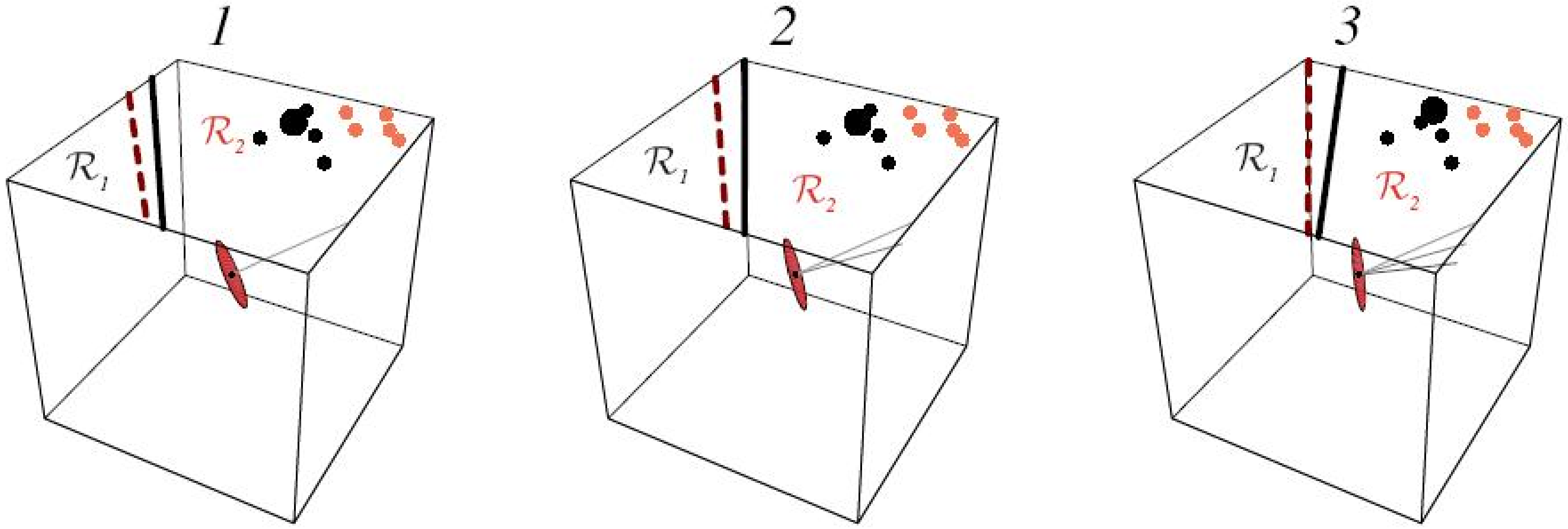
```
1. begin initialize a, $j \leftarrow 0,\ k \leftarrow 0$
2.        do $j \leftarrow j+1$
3.            $i = ((j-1) \bmod n) + 1$
4.            if $\mathbf{y}_i$ is misclassified by a
5.            then $k \leftarrow k+1;\ \mathbf{y}^k = \mathbf{y}_i;\ \mathbf{a} \leftarrow \mathbf{a} + \mathbf{y}^k$
6.        until $k$ is kept unchanged for $n$ consecutive rounds
7.     return a
8. end
```

Fixed-Increment
Single-Sample Perceptron

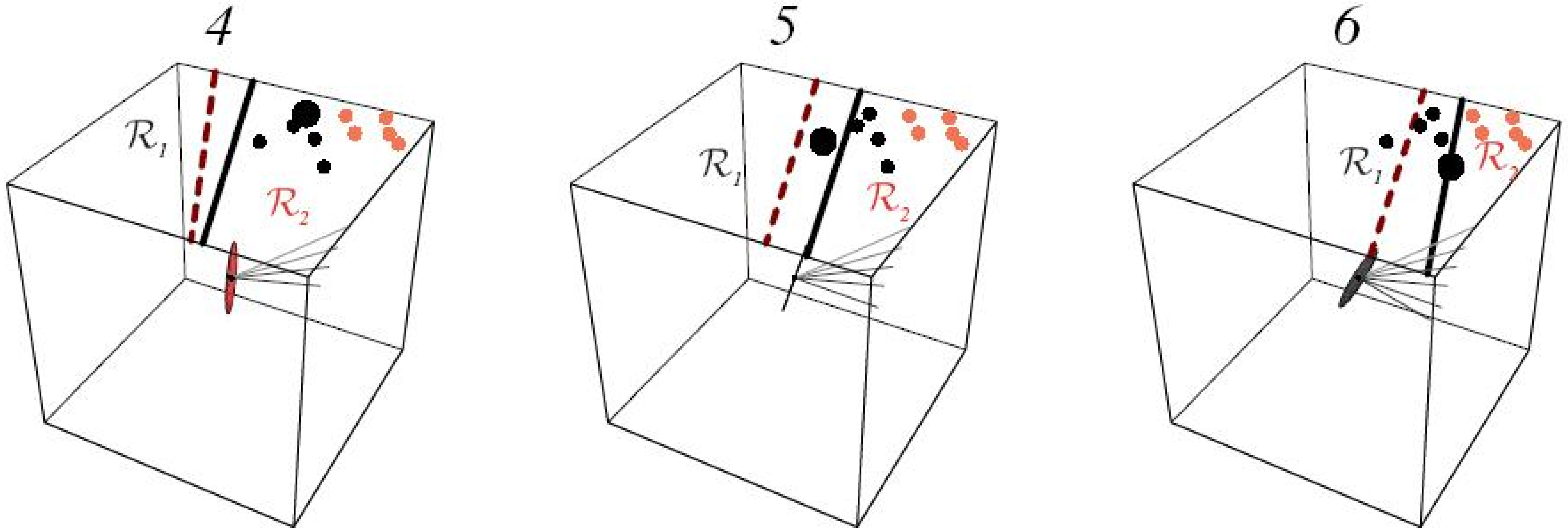
Perceptron Criterion Function Minimization



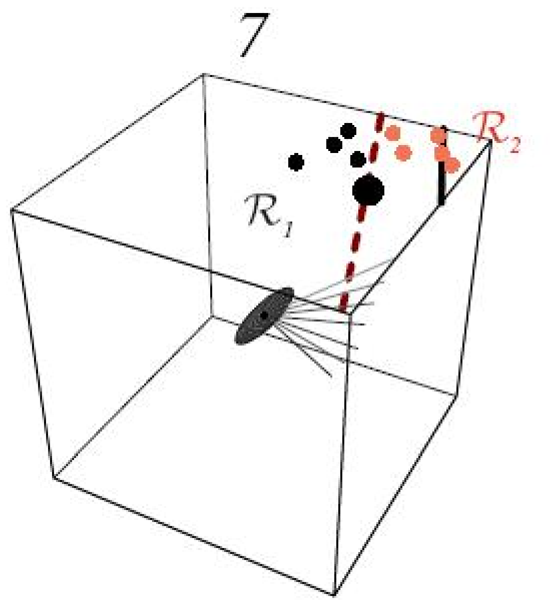
Perceptron Criterion Function Minimization

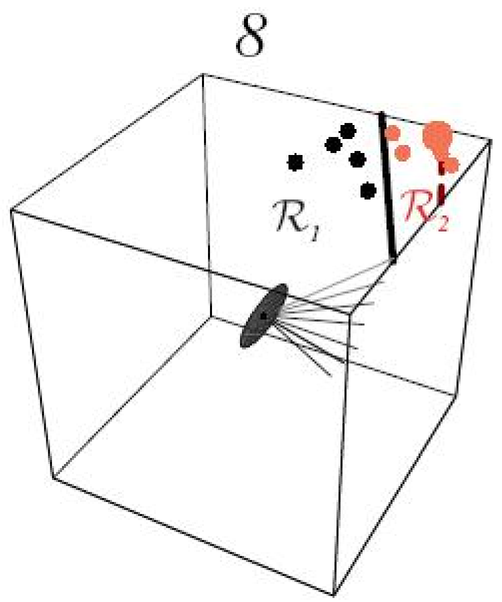


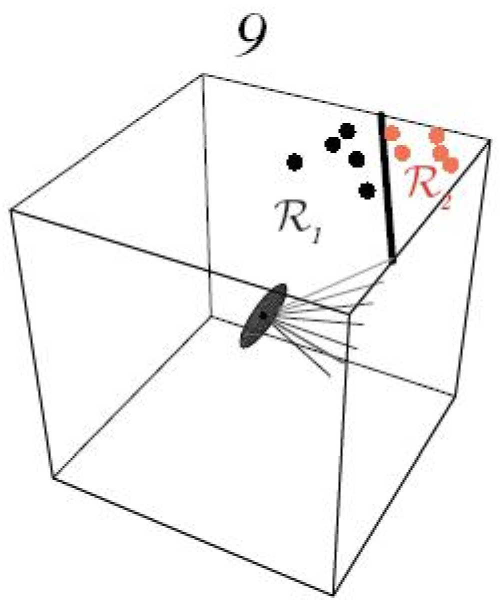
Perceptron Criterion Function Minimization



Perceptron Criterion Function Minimization







Perceptron Criterion Function Minimization

- Variable incremental perceptron algorithm with margin

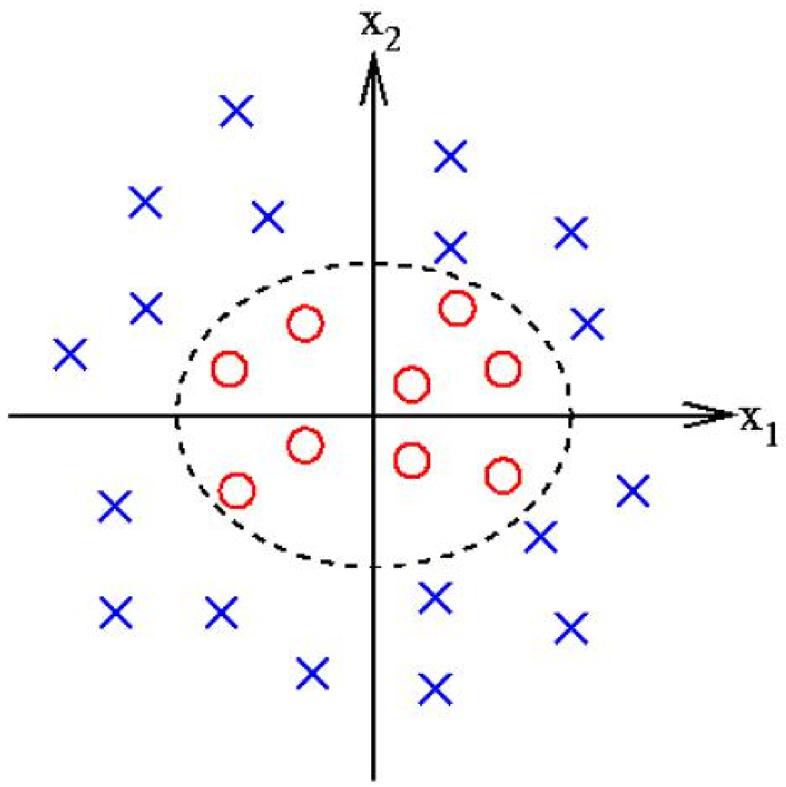
$$\mathbf{a}(k+1) = \mathbf{a}(k) + \boldsymbol{y}^k$$

```
1. begin initialize a, margin $b$, $\eta(\cdot)$, $j \leftarrow 0$, $k \leftarrow 0$
2.        do  $j \leftarrow j + 1$
3.              $i = ((j-1) \bmod n) + 1$
4.              if  $\mathbf{a}^t\mathbf{y}_i \leq b$
5.              then  $k \leftarrow k + 1$;  $\mathbf{y}^k = \mathbf{y}_i$;  $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\mathbf{y}^k$
6.        until  $k$ is kept unchanged for $n$ consecutive rounds
7.     return a
8. end
```

Variable-Increment Perceptron with Margin

Linearly Inseparable Case

- The perceptron algorithms can be summarized as “Error-Correction Procedure” (误差校正方法) in essence
- If the problem itself is linearly inseparable, the correction process may never end



Minimum Squared Error

- A criterion function that includes all samples (not just misclassified samples)
- Instead of pursuing $\mathbf{a}^t\mathbf{y}_i > 0$, let $\mathbf{a}^t\mathbf{y}_i = b_i$, where b_i is an arbitrarily chosen positive constant
- Replace solutions to a set of **linear inequalities** with a set of **linear equations**

$$\begin{pmatrix} Y_{10} & Y_{11} & \cdots & Y_{1d} \\ Y_{20} & Y_{21} & \cdots & Y_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ Y_{n0} & Y_{n1} & \cdots & Y_{nd} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ \vdots \\ a_d \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ \vdots \\ b_n \end{pmatrix}$$

Minimum Squared Error

- Number of samples is n, number of dimensions is d
- **Y** is the $n \times \widehat{d}$ matrix $(\widehat{d} = d + 1)$, whose *i*-th line is $\mathbf{y}_i^t$
- **B** is the column vector $b = (b_1, \ldots, b_n)^{\ t}$
- Then the set of linear equations can be written as $\mathbf{Ya} = b$
- If **Y** is non-singular, the solution is $\mathbf{a} = \mathbf{Y}^{-1} b$
- Normally,$n > \widehat{d}$, that is, the number of rows in **Y** is greater than the number of columns, the linear equations are overdetermined (超定), and **a** usually has no exact solution

Minimum Squared Error

- Define the error vector

$$e = \mathbf{Ya} - \mathrm{b}$$

- In the case that **a** has no solution, find **a** that minimizes the square of the error vector as the approximate solution of the linear equations

- **Least square error (least error sum of squares)** criterion function

$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathrm{b}\|^2 = \sum_{i=1}^{n} (\mathbf{a}^t \boldsymbol{y}_i - b_i)^2$$

Widrow-Hoff Algorithm

- Use gradient descent method to find the minimum value of

$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathrm{b}\|^2 \qquad\qquad \nabla J_s = 2\mathbf{Y}^t(\mathbf{Ya} - \mathrm{b})$$

- Recursion formula

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\mathbf{Y}^t(\mathbf{Ya}_{\boldsymbol{k}} - \mathrm{b})$$

Widrow-Hoff Algorithm

- Consider the single sample case: Widrow-Hoff algorithm or least mean square (LMS) algorithm

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)(b_k - \mathbf{a}^t(k)\mathbf{y}^k)\mathbf{y}^k$$

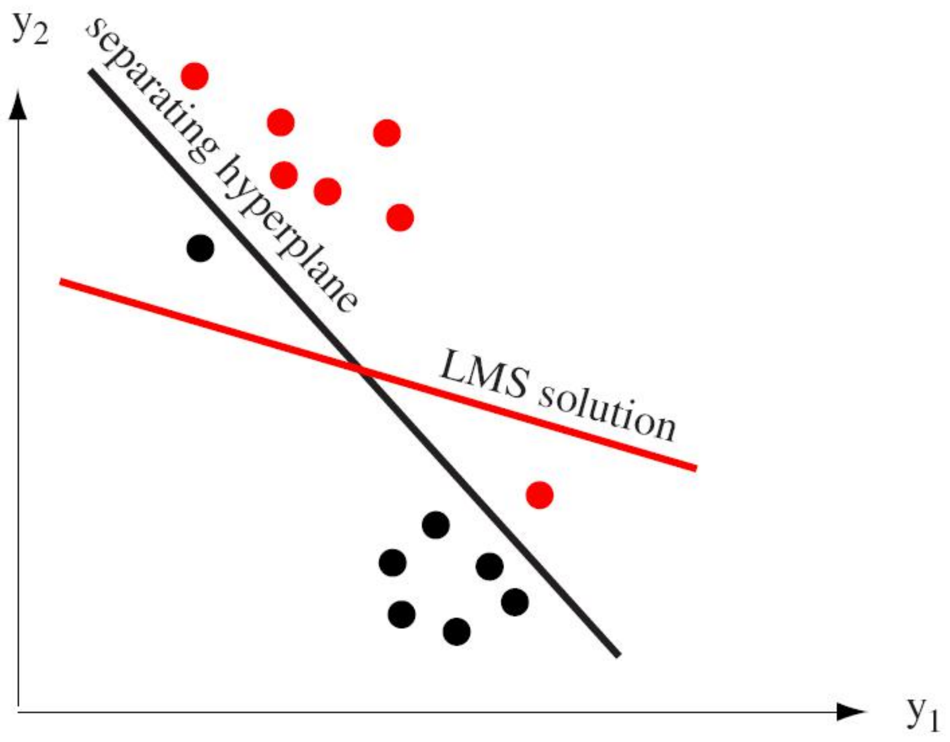
Algorithm description

```
1. begin initialize a, b, threshold $\theta$, $\eta(\cdot)$, $k \leftarrow 0$
2.          do $k \leftarrow k+1$
3.             $i = ((k-1) \bmod n) + 1$
4.             $\mathbf{y}^k \leftarrow \mathbf{y}_i;\ b_k \leftarrow b_i; \mathbf{a}_* \leftarrow \mathbf{a};\ \mathbf{a} \leftarrow \mathbf{a} + \eta(k)(b_k - \mathbf{a}^t\mathbf{y}^k)\mathbf{y}^k$
5.          until $\|\eta(k)(b_k - \mathbf{a}_*^t\mathbf{y}^k)\mathbf{y}^k\| < \theta$
6.       return a
7. end
```

LMS
(least-mean-squared)

Widrow-Hoff Algorithm

- Widrow-Hoff algorithm minimizes the sum of squares of the distance between the training point and the hyperplane
- Even in the case of linear separability, the solution of Widrow-Hoff algorithm may not be able to classify all the training samples completely correctly
- However, Widrow-Hoff algorithm can get good approximate solutions in both linearly separable and inseparable cases



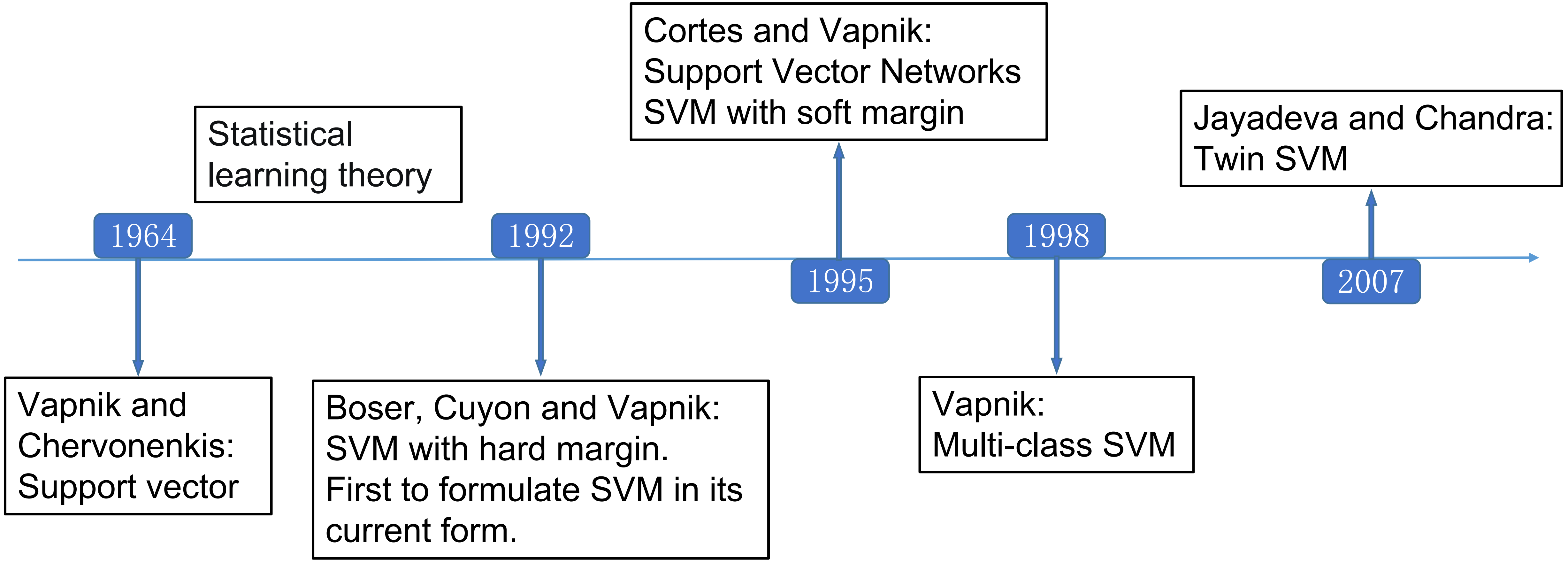
Support Vector Machine

Vladimir N. Vapnik, 1963—

- Support Vector Machine (SVM, 支持向量机) relies on a special preprocessing of the data so as to achieve a good classification
- The preprocessing maps the original data to a higher dimensional space $y_k = \varphi(x_k)$ by a nonlinear mapping $\varphi(\bullet)$ that makes the mapped points y_k linearly separable in this higher dimensional space

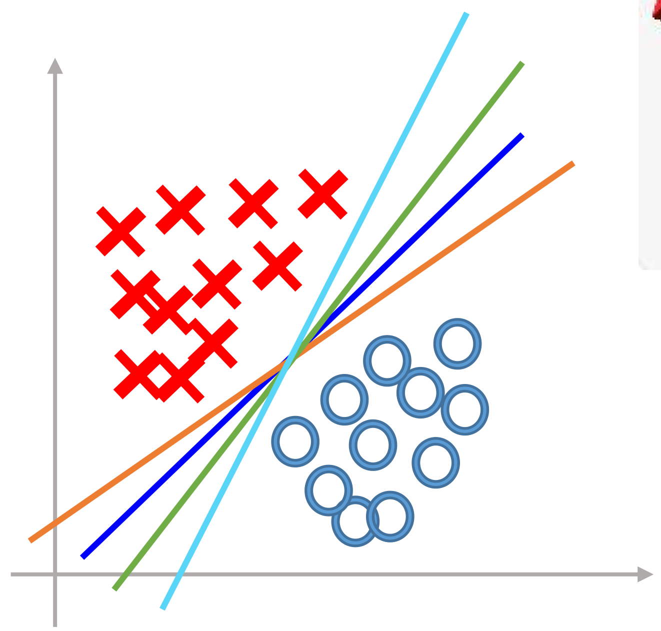
Support Vector Machine

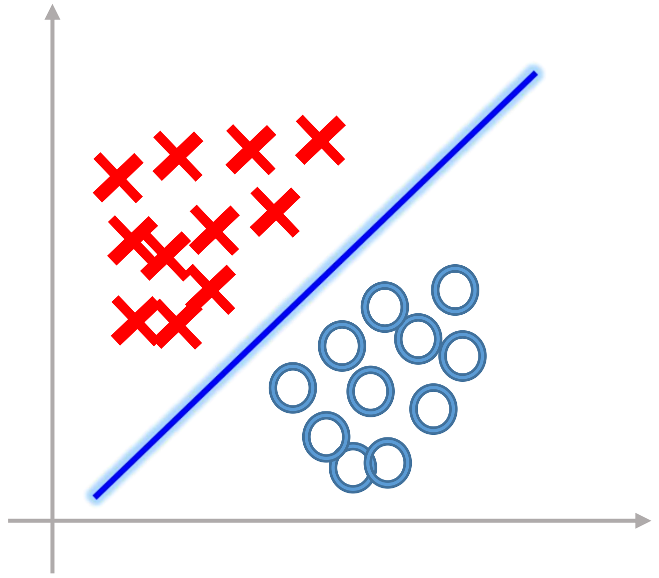
- The development of support vector machine


Statistical learning theory
Cortes and Vapnik: Support Vector Networks SVM with soft margin
Jayadeva and Chandra: Twin SVM
1964
1992
1995
1998
2007
Vapnik and Chervonenkis: Support vector
Boser, Cuyon and Vapnik: SVM with hard margin. First to formulate SVM in its current form.
Vapnik: Multi-class SVM

Support Vector Machine

- Which linear decision surface is the best?

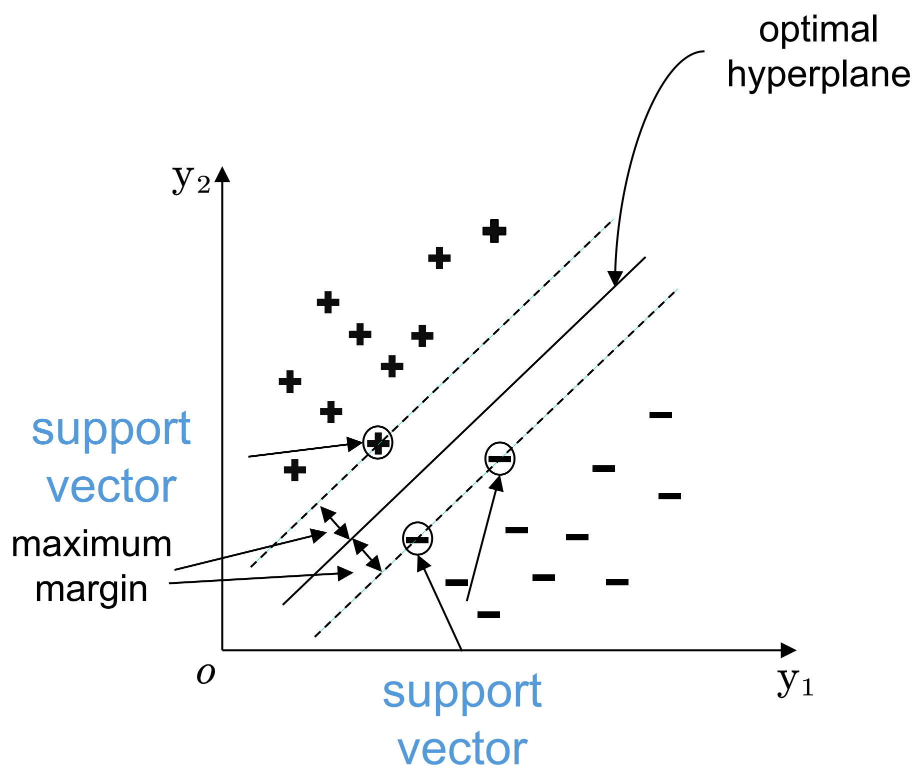




Support Vector Machine

- **Margin** (间隔)
  - The minimum distance from the model to the decision surface is called the classification margin (分类间隔)
  - The larger the classification margin, the better the decision surface

- The sample point closest to the decision surface is called the **support vector** (支持向量)



Support Vector Machine

- The weight vector a and the mapping point y in the high-dimensional space are both augmented, then the discriminant function is

$$g(\boldsymbol{y}) = \mathbf{a}^t \boldsymbol{y}$$

- Category mark
  - $z_k = +1$ indicates that $\boldsymbol{x}_k$ belongs to ω_1
  - $z_k = -1$ indicates that $\boldsymbol{x}_k$ belongs to ω_2
- Suppose the edge margin is 1, then there is

$$z_k g(\boldsymbol{y_k}) \geq 1 \qquad k = 1, \ldots, n$$

Support Vector Machine

- Distance from sample point to decision surface

$$\frac{|g(\mathbf{y}_k)|}{||\mathbf{a}||} = \frac{z_k g(\mathbf{y}_k)}{||\mathbf{a}||} \geq \frac{1}{||\mathbf{a}||}$$

classification margin

- Optimization objectives

maximizes $\frac{1}{||\mathbf{a}||}$, that is, minimize $||\mathbf{a}||$

s.t. $z_k g(\boldsymbol{y_k}) \geq 1$

Support Vector Machine

- The training of SVM

Kuhn-Tucker construction method

Minimize $L(\mathbf{a},\alpha)=\frac{1}{2}\|\mathbf{a}^2\|-\sum_{k=1}^{n}\alpha_k[z_k\mathbf{a}^t\boldsymbol{y}_k-1]$

s.t. $z_k g(\boldsymbol{y_k})\geq 1$

$\alpha_k[z_k\mathbf{a}^t\boldsymbol{y}_k-1]=0$

$\alpha_k\geq 0,\quad k=1,\dots,n$

Maximize $L(\alpha)=\sum_{k=1}^{n}\alpha_k-\frac{1}{2}\sum_{k,j}^{n}\alpha_k\alpha_j z_k z_j\boldsymbol{y}_j^t\boldsymbol{y}_k$

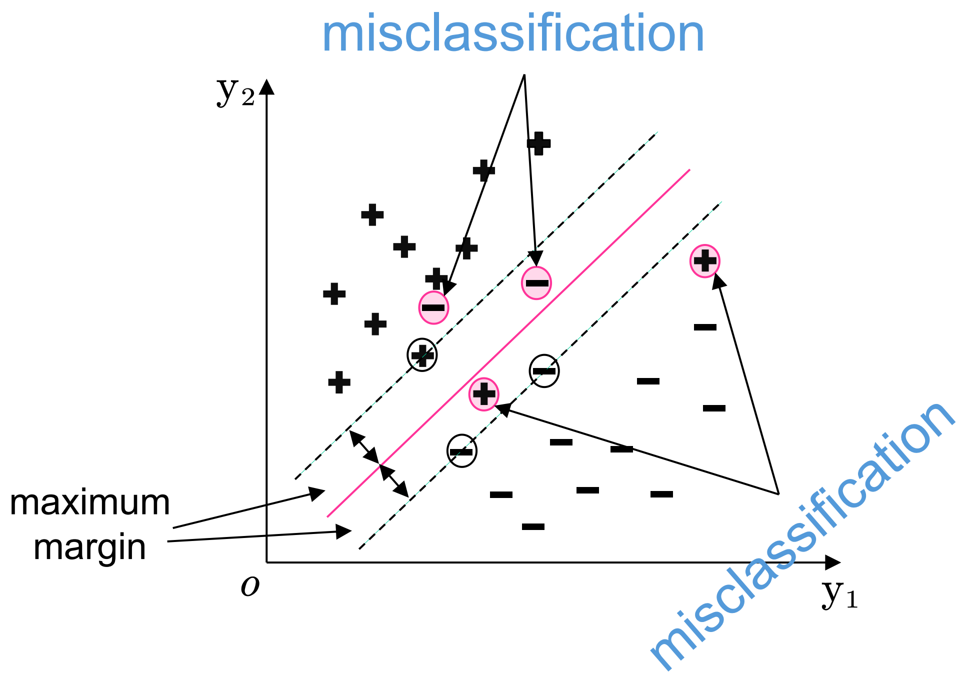
s.t. $\sum_{k=1}^{n}\alpha_k z_k=0 \qquad \alpha_k\geq 0,\ k=1,\dots,n$

quadratic programming problem

- **conjugate gradient method**
- **interior point method**
- **active set**
- **……**

Support Vector Machine

■ Support Vector Machine with Soft Margin



Hard Margin:

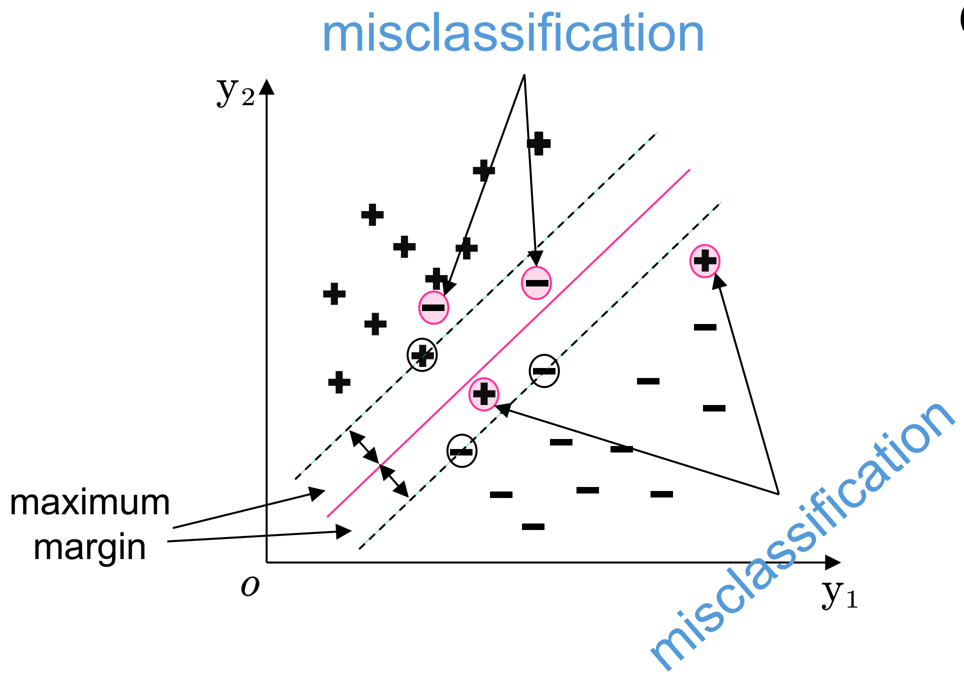
- The optimal solution may not exist
- It more likely tends to be overfitting

Soft Margin:

- The existence of optimal solution is guaranteed
- The generalization performance of model is improved
- The anti-interference ability is enhanced

Support Vector Machine

■ Support Vector Machine with Soft Margin



Optimization Objective with Hard Margin

$$\min_{a} \frac{1}{2}\|a\|^2$$
$$s.t.\ z_k a_k^T y_k \geq 1,\ k = 1,2,\dots,n.$$

$$\Big\Downarrow$$

slack variable $\xi > 0$

Empirical Loss

Optimization Objective with Soft Margin

$$\min_{a} \frac{1}{2}\|a\|^2 + \boxed{C\sum_{k=1}^{n}\xi_i}$$
$$s.t.\ z_k a_k^T y_k \geq 1 - \xi_i,$$
$$\xi_k \geq 0,\ k = 1,2,\dots,n.$$

Support Vector Machine

- The training of SVM with soft margin

Karush-Kuhn-Tucher construction method

Minimize $L(\mathrm{a}, \alpha, \xi, \mu) = \frac{1}{2}\|\mathrm{a}\|^2 + C\sum_{k=1}^{n}\xi_k - \sum_{k=1}^{n}\alpha_k\left[z_k \mathrm{a}_k^T y_k + \xi_k - 1\right] - \sum_{k=1}^{n}\mu_k\xi_k$

s.t. $\alpha_k \geq 0,\ \ \mu_k \geq 0$

$\alpha_k\left[z_k \mathrm{a}_k^T y_k + \xi_k - 1\right] = 0$

$z_k g(y_k) + \xi_k \geq 1$

$\xi_k \geq 0,\ \mu_k\xi_k = 0$

$$\Downarrow$$

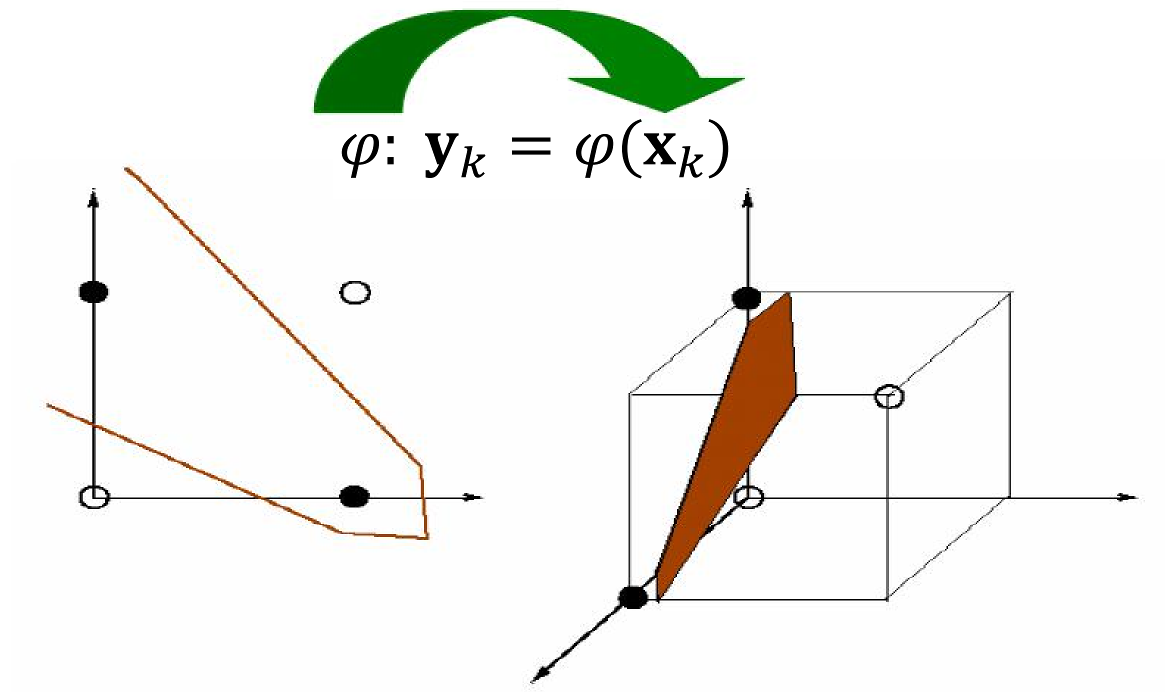
Maximize $L(\alpha) = \sum_{k=1}^{n}\alpha_k - \frac{1}{2}\sum_{\mathrm{k}=1}^{n}\sum_{j=1}^{n}\alpha_k\alpha_j z_k z_j y_j^t y_k,$

s.t. $\sum_{k=1}^{n}\alpha_k z_k = 0\,,\ \ 0 \leq \alpha_k \leq C,\ k = 1, \cdots, n$

quadratic programming problem

Support Vector Machine

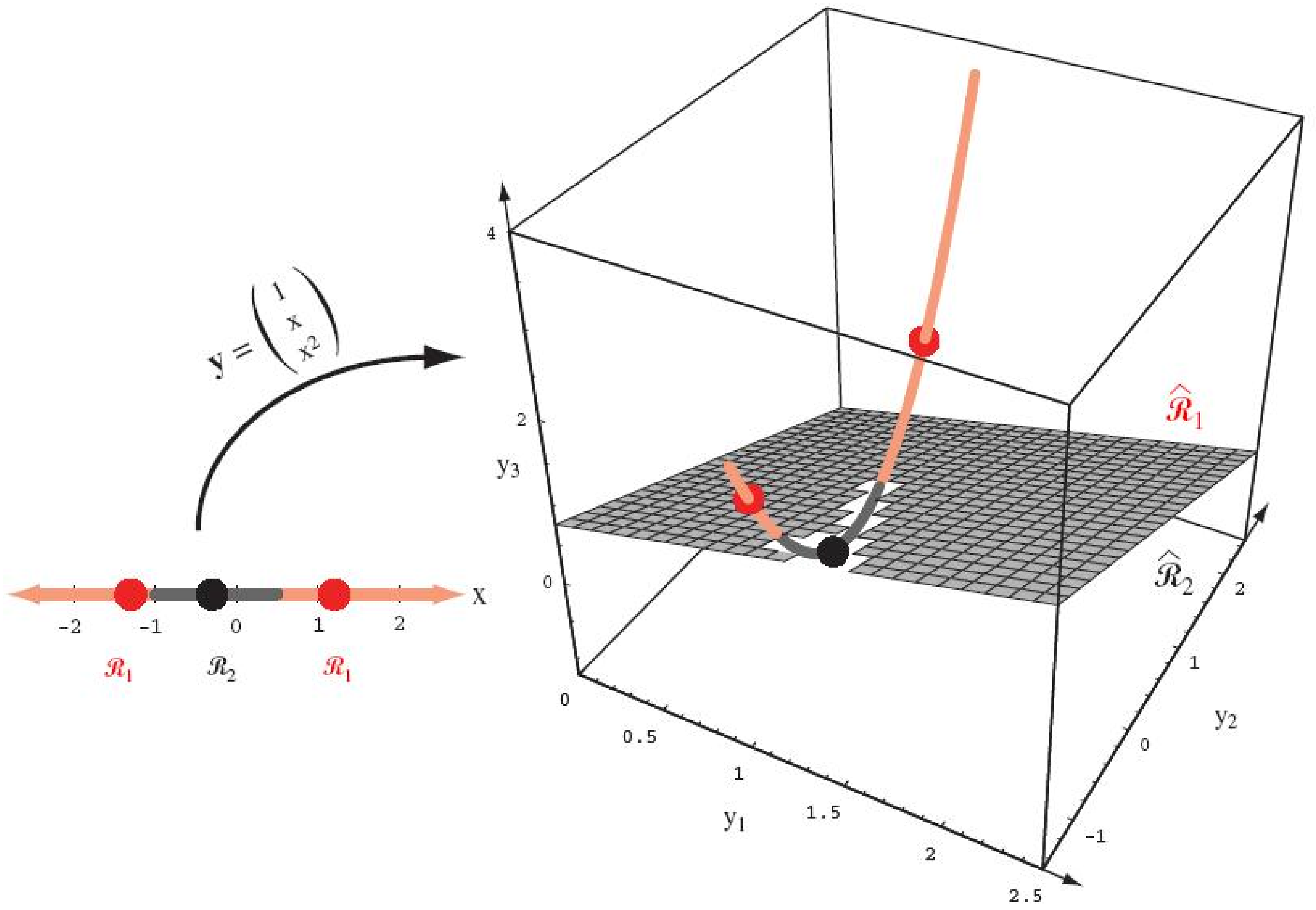
- Non-linear mapping



Basic idea: when the original training sample is linearly inseparable, a certain nonlinear transformation can be used to map the sample points in the original data space to a high-dimensional space, so that the mapping points in this high-dimensional space tend to linearly separable

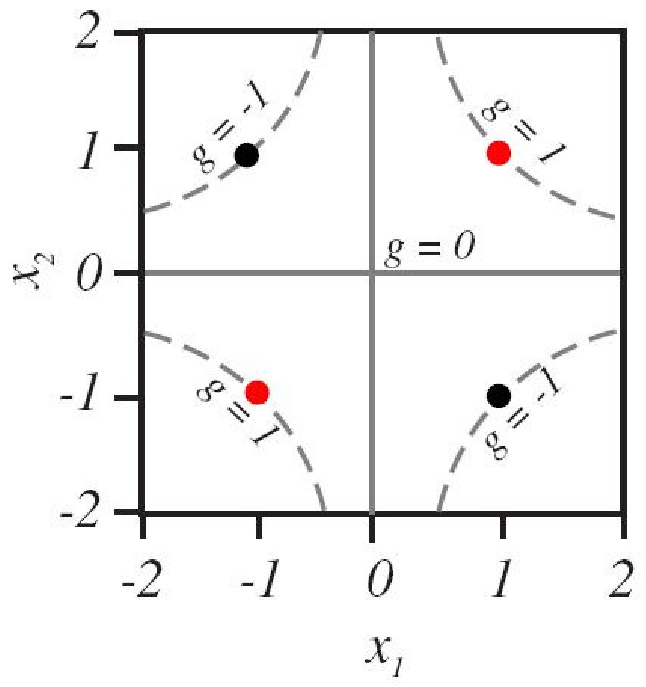
Support Vector Machine

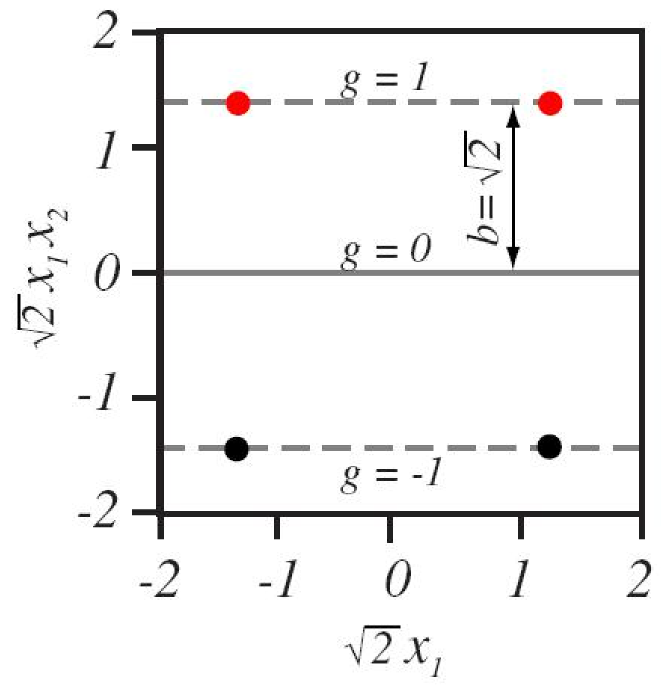
- Non-linear mapping



Support Vector Machine

- Non-linear mapping
  - Non-linear mapping φ reflects the prior knowledge of the designer
  - Without prior knowledge, commonly used nonlinear mapping functions include: **polynomial function**, **Gaussian function**, etc.





Support Vector Machine

- **Kernel trick** (核技巧)
  - Linear classifiers only rely on **inner product computations** $\mathbf{x}_i^t \mathbf{x}_j$
  - If all sample points map to a higher dimensional (or even infinite dimensional) space φ: $\mathbf{y}_k = \varphi(\mathbf{x}_k)$, the inner product $\varphi(\mathbf{x}_i)^t \varphi(\mathbf{x}_j)$ in this higher dimensional space is often difficult or impossible to compute at all
  - Kernel function refers to the function in the original data space corresponding to the inner product calculation in the transformed space

$$k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^t \varphi(\mathbf{x}_j)$$

This shows that the inner product computation in the complex higher dimensional space can be done indirectly through the kernel function in the lower dimensional space

If a problem involves only inner product computation, you can specify no specific mapping function, but only the kernel function corresponding to that mapping function

Support Vector Machine

- **Kernel trick** (核技巧)
  - What kind of function can be a kernel?
    - **Mercer theorem**
      Any positive semi-definite symmetric function can be used as a kernel
      - **positive semi-definite**
        The matrix A composed of $a_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is a positive semi-definite matrix, that is, for any non-zero real vector z, there is $\mathbf{z}^t \mathbf{A}\mathbf{z} \geq 0$
      - **symmetric**

$$k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i)$$

Support Vector Machine

- Kernel trick (核技巧)
  - Commonly used kernel functions
    - Polynomial kernel

$$k(\mathbf{x},\mathbf{y}) = (\mathbf{x}^t\mathbf{y} + 1)^d$$

    - Gaussian kernel

$$k(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$

    - Sigmoid kernel

$$k(\mathbf{x},\mathbf{y}) = \tanh\left(k\mathbf{x}^t\mathbf{y} - \delta\right)$$

Generalize to Multicategory Problem

- The algorithm mentioned above assumes the two-category premises

- Generalize to multicategory problems

  - If $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n$ is linearly separable, then there is a set of weight vectors $\widehat{\mathbf{a}_1}, \ldots\widehat{\mathbf{a}_c}$, when $\boldsymbol{y}_k \in \mathcal{Y}_i$, for all $j \neq i$, there is $\hat{\mathbf{a}}_i^t \boldsymbol{y}_k > \hat{\mathbf{a}}_j^t \boldsymbol{y}_k$

  - When $\widehat{\mathbf{a}_1}, \ldots\widehat{\mathbf{a}_c}$ are determined, for test sample **y**, if for all $j \neq i$,there is $\hat{\mathbf{a}}_i^t \mathbf{y} > \hat{\mathbf{a}}_j^t \mathbf{y}$. Then **y** belongs to category i

Generalize to Multicategory Problem

- Kesler construction method
  - Suppose $\boldsymbol{y} \in \mathcal{Y}_1$, then

$$\hat{\mathbf{a}}_1^t \mathbf{y} - \hat{\mathbf{a}}_j^t \mathbf{y} > 0, \;\; j = 2, \ldots, c$$

  - Equivalently， $c\hat{d}$-dimensional weight vector

$$\hat{\mathbf{a}} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_c \end{bmatrix}$$

Can correctly classify the c-1 $c\hat{d}$ -dimensional sample set (as shown on the following page)

Generalize to Multicategory Problem

- Kesler construction method

$$\eta_{12} = \begin{bmatrix} \mathbf{y} \\ -\mathbf{y} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \eta_{12} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \\ -\mathbf{y} \\ \vdots \\ 0 \end{bmatrix}, \cdots, \eta_{1c} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ -\mathbf{y} \end{bmatrix}$$

Each η_{1j} corresponds to normalizing the samples belonging to ω_1 and ω_j, that is, negative samples belonging to ω_j. In this way, $\hat{\boldsymbol{\alpha}}^t \boldsymbol{\eta}_{1j} > 0$ can be guaranteed and **multi-category problem** can be successfully converted into **two-category problem**

Generalize to Multicategory Problem

- Kesler construction method
  - When $\boldsymbol{y} \in \mathcal{Y}_i$, c-1 $c\widehat{d}$ dimensional training samples can be similarly constructed, where the i-th sub-vector is **y**, the j-th sub-vector is **-y**, and the rest are 0
  - If there is $\widehat{\boldsymbol{\alpha}}^t \boldsymbol{\eta}_{ij} > 0$ for all $j \neq i$, then the components in $\widehat{\boldsymbol{\alpha}}$ constitute c weight vectors of the linear machine that can correctly classify the multi-category sample set