

# Chapter 5:

## Linear Discriminant Functions

- Introduction
- Linear Discriminant Functions and Decisions Surfaces
- Generalized Linear Discriminant Functions
- Minimum Squared Error Procedures
- Support Vector Machines

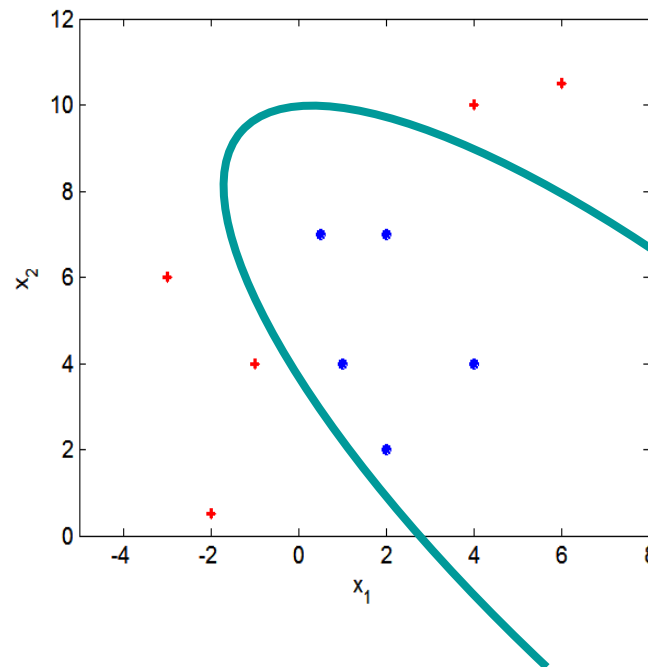


- Introduction
  - In chapter 3, the underlying probability densities were known (or given)
  - The training sample was used to estimate the parameters of these probability densities (ML, MAP estimations)
  - In this chapter, we only know the proper forms for the discriminant functions: similar to non-parametric techniques
  - They may not be optimal, but they are very simple to use
  - They provide us with linear classifiers

- The problem of finding a linear discriminant function will be formulated as a problem of minimizing a criterion function.
  - *sample risk, or training error; the average loss incurred in classifying training the set of training samples.*
- It is difficult to derive the minimum-risk linear discriminant anyway, and for that reason we investigate several related criterion functions that are analytically more tractable.

# Discriminant Approach

- Classification is viewed as “*learning good decision boundaries*” that separate the examples belonging to different classes in a data set.



# Discriminant function estimation

- Specify a parametric form of the decision boundary (e.g., *linear* or *quadratic*) .
- Find the “best” decision boundary of the specified form using a set of training examples.
- This is done by minimizing a criterion function
  - e.g., “training error” (or “sample risk”)

$$J(w) = \frac{1}{n} \sum_{k=1}^n [z_k - g(x_k, w)]^2$$

- Linear discriminant functions and decisions surfaces

**The Two-Category Case**

– **Definition**

A discriminant function that is a linear combination of the components of  $\mathbf{x}$  can be written as

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 \quad (1)$$

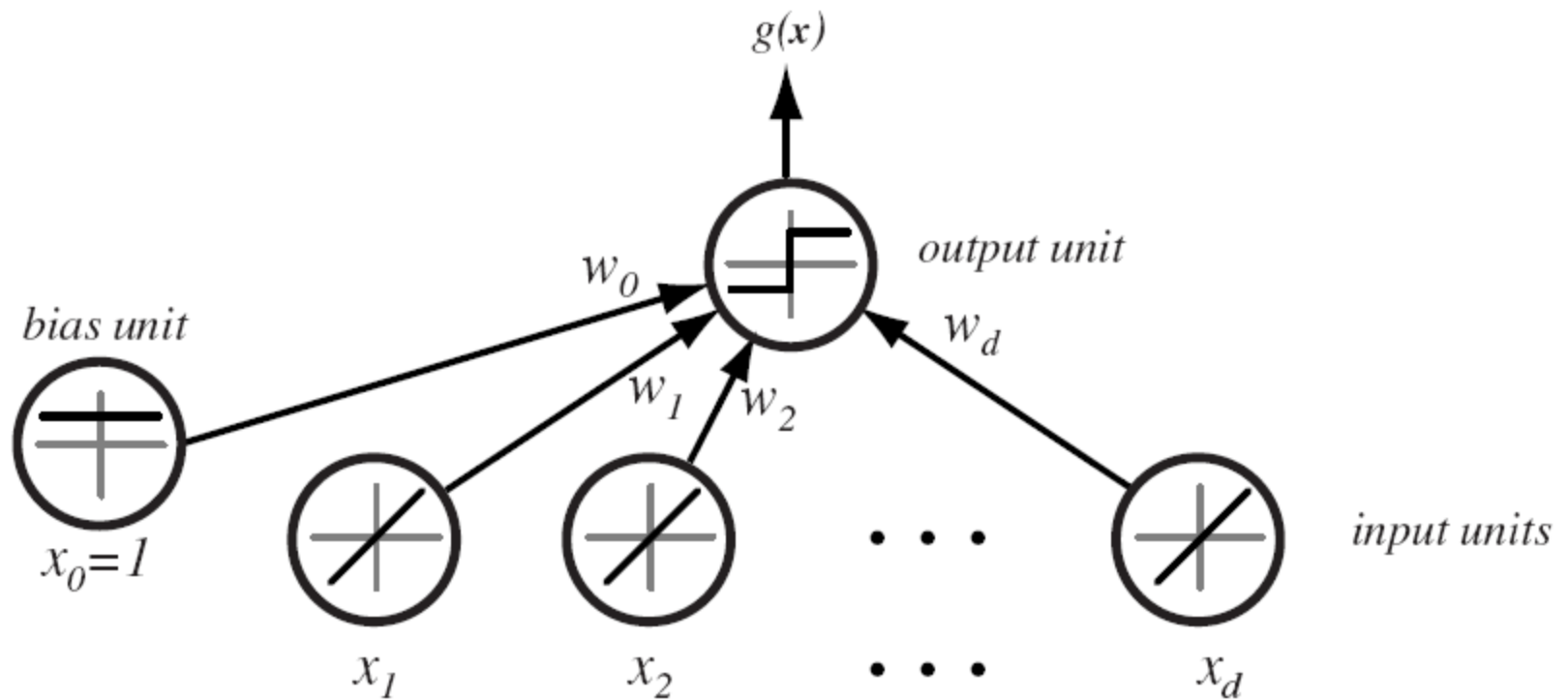
where  $\mathbf{w}$  is the weight vector and  $w_0$  the bias

– **A two-category classifier** with a discriminant function of the form (1) uses the following rule:

Decide  $\omega_1$  if  $g(\mathbf{x}) > 0$  and  $\omega_2$  if  $g(\mathbf{x}) < 0$

$\Leftrightarrow$  Decide  $\omega_1$  if  $\mathbf{w}^t \mathbf{x} > -w_0$  and  $\omega_2$  otherwise

If  $g(\mathbf{x}) = 0 \Rightarrow \mathbf{x}$  is assigned to either class



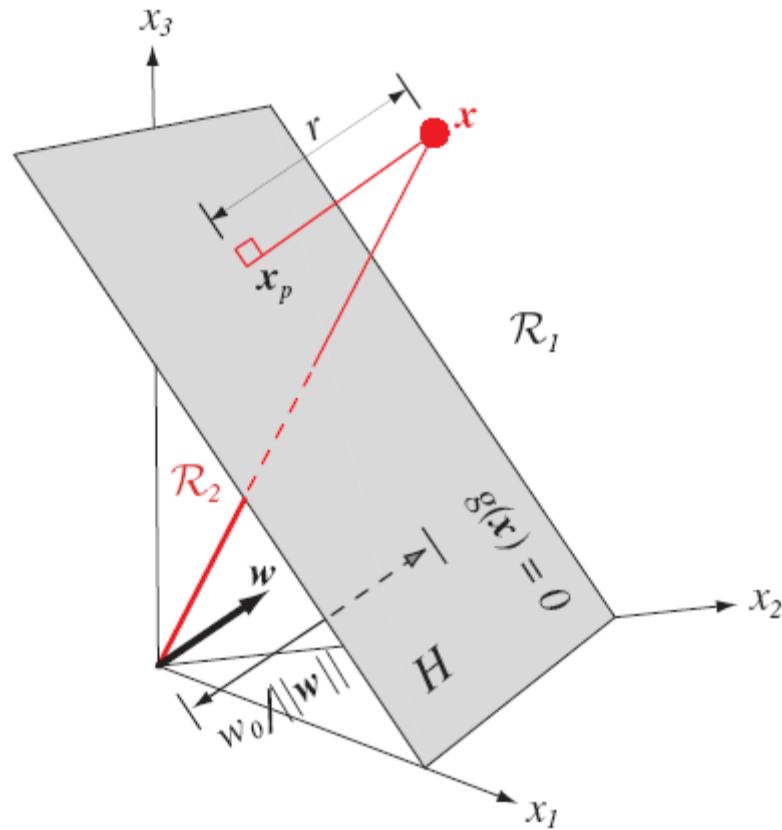
**Figure 5.1:** A simple linear classifier having  $d$  input units, each corresponding to the values of the components of an input vector. Each input feature value  $x_i$  is multiplied by its corresponding weight  $w_i$ ; the output unit sums all these products and emits a  $+1$  if  $\mathbf{w}^t \mathbf{x} + w_0 > 0$  or a  $-1$  otherwise.

- The equation  $g(\mathbf{x}) = 0$  defines the decision surface that separates points assigned to the category  $\omega_1$  from points assigned to the category  $\omega_2$
- When  $g(\mathbf{x})$  is linear, the decision surface is a hyperplane
- Algebraic measure of the distance from  $\mathbf{x}$  to the hyperplane (interesting result!)
- If  $\mathbf{x}_1$  and  $\mathbf{x}_2$  are both on the decision surface, then

$$\mathbf{w}^t \mathbf{x}_1 + w_0 = \mathbf{w}^t \mathbf{x}_2 + w_0 \quad \Leftrightarrow \quad \mathbf{w}^t (\mathbf{x}_1 - \mathbf{x}_2) = 0,$$

$\mathbf{w}$  is normal to any vector lying in the hyperplane.





**Figure 5.2:** The linear decision boundary  $H$ , where  $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$ , separates the feature space into two half-spaces  $R_1$  (where  $g(\mathbf{x}) > 0$ ) and  $R_2$  (where  $g(\mathbf{x}) < 0$ ).

$w_0$  determines the distance of the hyperplane from the origin 9

$$\mathbf{x} = \mathbf{x}_p + \frac{r \cdot \mathbf{w}}{\|\mathbf{w}\|} \quad (\text{since } \mathbf{w} \text{ is colinear with } \mathbf{x} - \mathbf{x}_p \text{ and } \frac{\mathbf{w}}{\|\mathbf{w}\|} = 1)$$

$$\text{since } g(\mathbf{x}_p) = 0 \text{ and } \mathbf{w}^t \cdot \mathbf{w} = \|\mathbf{w}\|^2 \Rightarrow g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 =$$

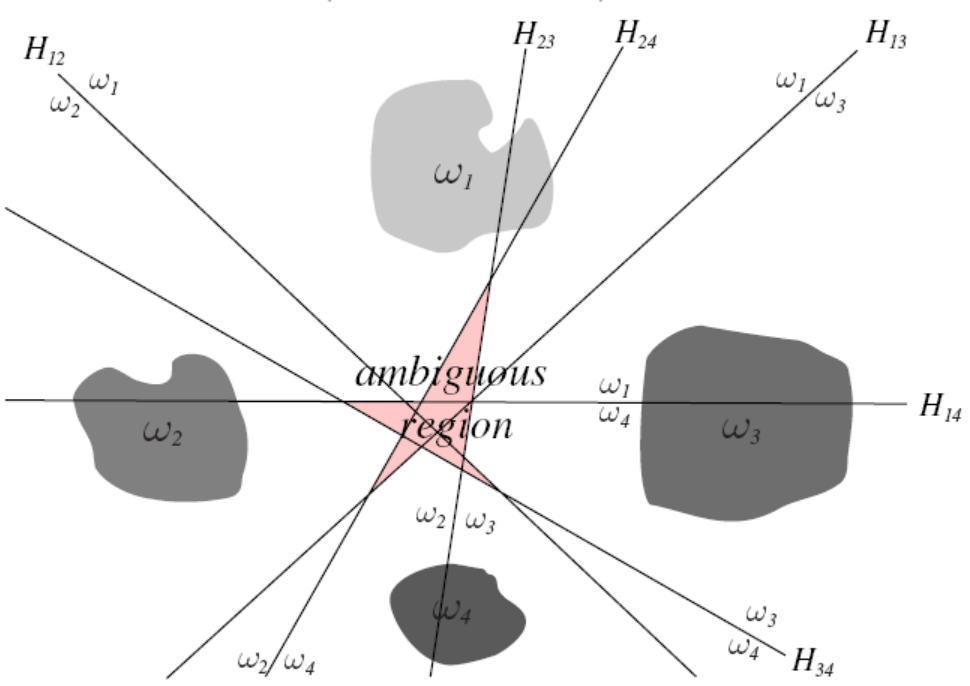
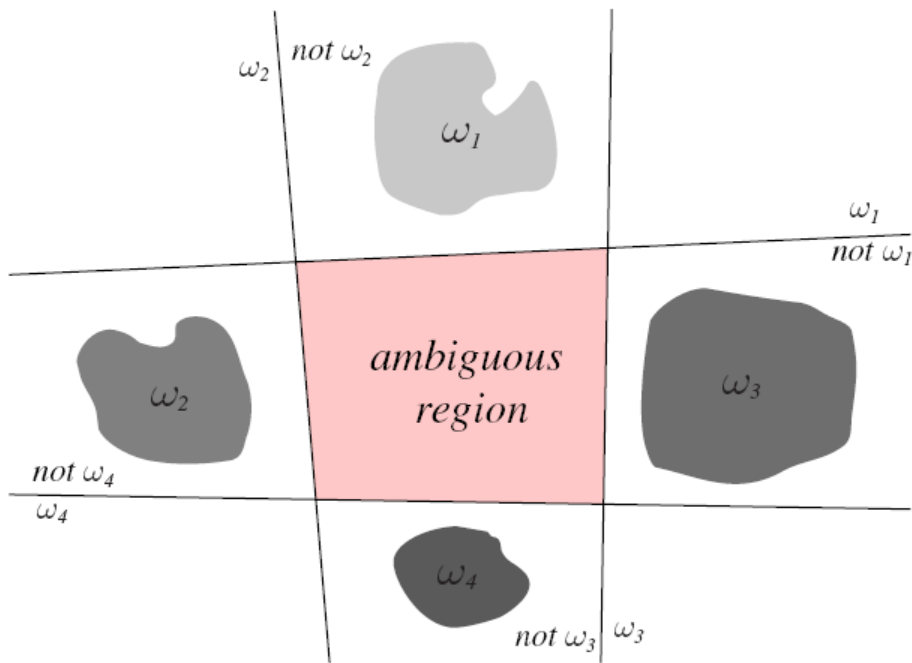
$$= \mathbf{w}^t \left( \mathbf{x}_p + \frac{r \cdot \mathbf{w}}{\|\mathbf{w}\|} \right) + w_0 = \mathbf{w}^t \mathbf{x}_p + w_0 + \frac{r \mathbf{w}^t \cdot \mathbf{w}}{\|\mathbf{w}\|} = r \|\mathbf{w}\|$$

$$\text{therefore } r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|} \rightarrow \text{in particular } d(0, H) = \frac{g(\mathbf{0})}{\|\mathbf{w}\|} = \frac{w_0}{\|\mathbf{w}\|}$$

- In conclusion, a linear discriminant function divides the feature space by a hyperplane decision surface
- The orientation of the surface is determined by the normal vector  $\mathbf{w}$  and the location of the surface is determined by the bias

# The multi-category case

- Reduce the problem to  $c - 1$  two-class; separates points assigned to  $\omega_i$  from those not assigned to  $\omega_i$ .
- Use  $c(c - 1)/2$  linear discriminants, one for every pair of classes.
  - both can lead to regions in which the classification is undefined.
- Defining  $c$  linear discriminant functions



**Figure 5.3:** Linear decision boundaries for a four-class problem. The top figure shows  $\omega_i/\text{not } \omega_i$  dichotomies while the bottom figure shows  $\omega_i/\omega_j$  dichotomies. The pink regions have ambiguous category assignments.

- No  $\Delta H_{12} H_{23} H_{24}$
- $\Delta H_{24} H_{13} H_{34}$
- $\Delta H_{34} H_{12} H_{23}$
- $\square H_{14} H_{24} H_{34} H_{12}$

- **The multi-category case**

- We define  $c$  linear discriminant functions

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

and assign  $\mathbf{x}$  to  $\omega_i$  if  $g_i(\mathbf{x}) > g_j(\mathbf{x}) \forall j \neq i$ ; in case of ties, the classification is undefined

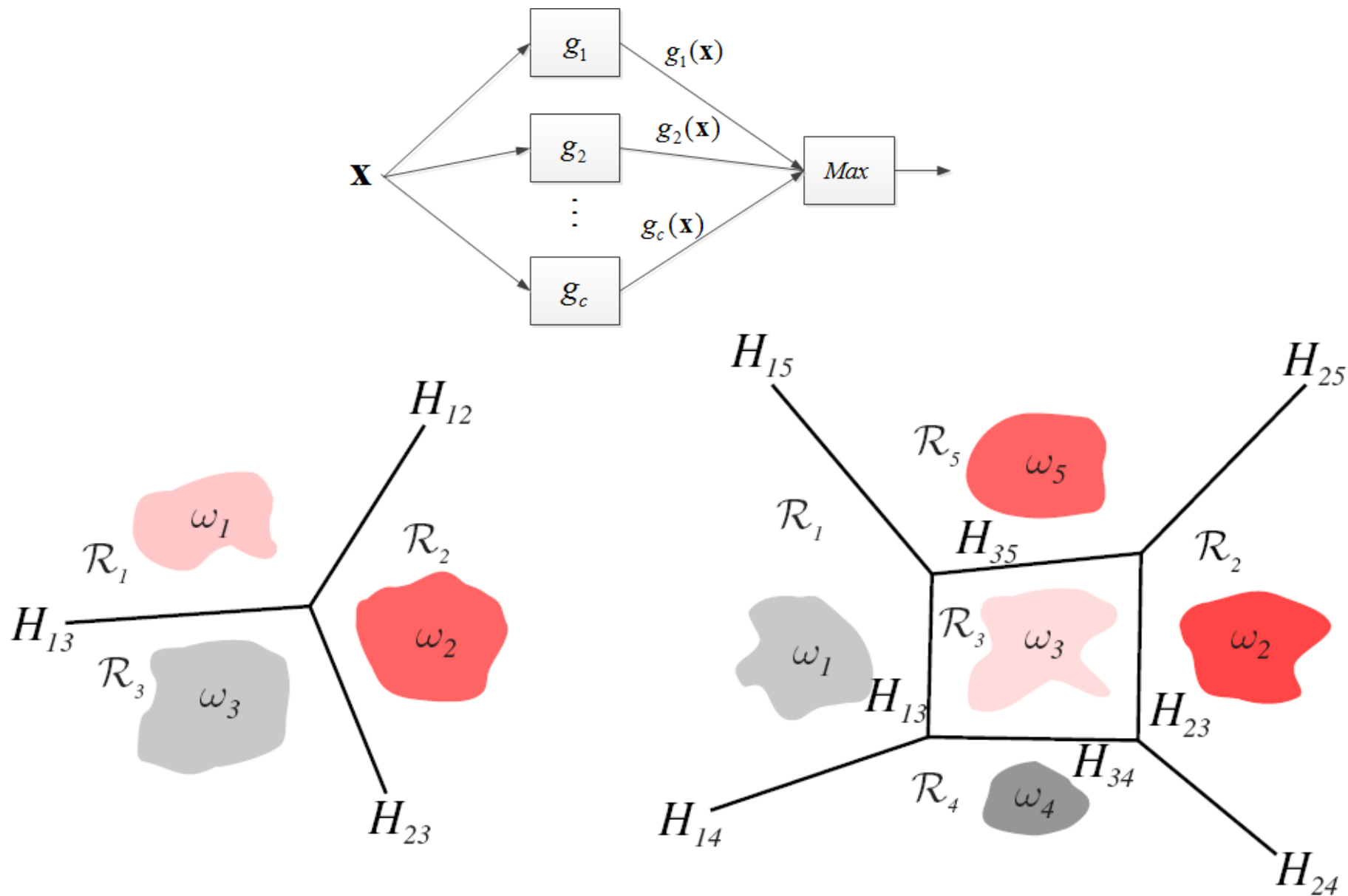
- The classifier is a “**linear machine**”.
- A linear machine divides the feature space into  $c$  decision regions, with  $g_i(\mathbf{x})$  being the largest discriminant if  $\mathbf{x}$  is in the region  $R_i$

- For a two contiguous regions  $R_i$  and  $R_j$ ; the boundary that separates them is a portion of hyperplane  $H_{ij}$  defined by:

$$\begin{aligned} g_i(\mathbf{x}) &= g_j(\mathbf{x}) \\ \Leftrightarrow (\mathbf{w}_i - \mathbf{w}_j)^t \mathbf{x} + (w_{i0} - w_{j0}) &= 0 \end{aligned}$$

$\mathbf{w}_i - \mathbf{w}_j$  is normal to  $H_{ij}$  and

$$d(\mathbf{x}, H_{ij}) = \frac{g_i - g_j}{\|\mathbf{w}_i - \mathbf{w}_j\|}$$



**Figure 5.4:** Decision boundaries produced by a linear machine for a three-class problem and a five-class problem.<sup>15</sup>

- It is easy to show that the decision regions for a linear machine are convex, this restriction limits the flexibility and accuracy of the classifier.
- In particular, for good performance every decision region should be singly connected, and this tends to make the linear machine most suitable for problems for which the conditional densities  $p(\mathbf{x}/\omega_i)$  are unimodal.



- **Generalized Linear Discriminant Functions**

Decision boundaries which separate between classes may not always be linear

- The complexity of the boundaries may sometimes request the use of highly non-linear surfaces
- A popular approach to generalize the concept of linear decision functions is to consider a generalized decision function as:

$$g(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \dots + w_N f_N(\mathbf{x}) + w_{N+1} \quad (1)$$

where  $f_i(\mathbf{x})$ ,  $1 \leq i \leq N$  are scalar functions of the pattern  $\mathbf{x}$ ,  $\mathbf{x} \in IR^n$

- Introducing  $f_{N+1}(\mathbf{x}) = 1$  we get:

$$g(\mathbf{x}) = \sum_{i=1}^{N+1} w_i f_i(\mathbf{x}) = \mathbf{w}^T \dot{\mathbf{x}}$$

$$\text{where } \mathbf{w} = (w_1, w_2, \dots, w_N, w_{N+1})^T$$

$$\text{and } \dot{\mathbf{x}} = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_N(\mathbf{x}), f_{N+1}(\mathbf{x}))^T$$

- This latter representation of  $g(\mathbf{x})$  implies that any decision function defined by equation (1) can be treated as linear in the  $(N + 1)$  dimensional space ( $N + 1 > n$ )
- $g(\mathbf{x})$  maintains its non-linearity characteristics in  $IR^n$
- Quadratic decision functions for a 2-dimensional feature space

$$g(\mathbf{x}) = w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2 + w_4 x_1 + w_5 x_2 + w_6$$

$$\text{here: } \mathbf{w} = (w_1, w_2, \dots, w_6)^T \text{ and } \dot{\mathbf{x}} = (x_1^2, x_1 x_2, x_2^2, x_1, x_2, 1)^T$$

- For patterns  $\mathbf{x} \in IR^n$ , the most general quadratic decision function is given by:

$$g(\mathbf{x}) = \sum_{i=1}^n w_{ii} x_i^2 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n w_{ij} x_i x_j + \sum_{i=1}^n w_i x_i + w_{n+1} \quad (2)$$

The number of terms at the right-hand side is:

$$l = N + 1 = n + \frac{n(n-1)}{2} + n + 1 = \frac{(n+1)(n+2)}{2}$$

This is the total number of weights which are the free parameters of the problem

- If for example  $n = 3$ , the vector  $\mathbf{x}$  is 10-dimensional
- If for example  $n = 10$ , the vector  $\mathbf{x}$  is 65-dimensional

- In the case of polynomial decision functions of order  $m$ , a typical  $f_i(\mathbf{x})$  is given by:

$$f_i(\mathbf{x}) = x_{i_1}^{e_1} x_{i_2}^{e_2} \dots x_{i_m}^{e_m}$$

where  $1 \leq i_1, i_2, \dots, i_m \leq n$  and  $e_i, 1 \leq i \leq m$  is 0 or 1.

- It is a polynomial with a degree between 0 and  $m$ . To avoid repetitions, we request  $i_1 \leq i_2 \leq \dots \leq i_m$

- $$g^m(\mathbf{x}) = \sum_{i_1=1}^n \sum_{i_2=i_1}^n \dots \sum_{i_m=i_{m-1}}^n w_{i_1 i_2 \dots i_m} x_{i_1} x_{i_2} \dots x_{i_m} + g^{m-1}(\mathbf{x})$$

(where  $g^0(\mathbf{x}) = w_{n+1}$ ) is the most general polynomial decision function of order  $m$

Example 1: Let  $n = 3$  and  $m = 2$  then:

$$\begin{aligned}
 g^2(\mathbf{x}) &= \sum_{i_1=1}^3 \sum_{i_2=i_1}^3 w_{i_1 i_2} x_{i_1} x_{i_2} + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 \\
 &= w_{11} x_1^2 + w_{12} x_1 x_2 + w_{13} x_1 x_3 + w_{22} x_2^2 + w_{23} x_2 x_3 + w_{33} x_3^2 \\
 &\quad + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4
 \end{aligned}$$

Example 2: Let  $n = 2$  and  $m = 3$  then:

$$\begin{aligned}
 g^3(\mathbf{x}) &= \sum_{i_1=1}^2 \sum_{i_2=i_1}^2 \sum_{i_3=i_2}^2 w_{i_1 i_2 i_3} x_{i_1} x_{i_2} x_{i_3} + g^2(\mathbf{x}) \\
 &= w_{111} x_1^3 + w_{112} x_1^2 x_2 + w_{122} x_1 x_2^2 + w_{222} x_2^3 + g^2(\mathbf{x})
 \end{aligned}$$

$$\text{where } g^2(\mathbf{x}) = \sum_{i_1=1}^2 \sum_{i_2=i_1}^2 w_{i_1 i_2} x_{i_1} x_{i_2} + g^1(\mathbf{x})$$

$$= w_{11} x_1^2 + w_{12} x_1 x_2 + w_{22} x_2^2 + w_1 x_1 + w_2 x_2 + w_3$$

- The commonly used **quadratic** decision function can be represented as the general  $n$ -dimensional quadratic surface:

$$g(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} + c$$

where the matrix  $\mathbf{A} = (a_{ij})$ , the vector

$\mathbf{b} = (b_1, b_2, \dots, b_n)^T$  and  $c$ , depends on the weights  $w_{ii}$ ,  $w_{ij}$ ,  $w_i$  of equation (2)

- If  $\mathbf{A}$  is positive definite then the decision function is a hyperellipsoid with axes in the directions of the eigenvectors of  $\mathbf{A}$ 
  - In particular: if  $\mathbf{A} = \mathbf{I}_n$  (Identity), the decision function is simply the  $n$ -dimensional hypersphere
  - If  $\mathbf{A}$  is negative definite, the decision function describes a hyperhyperboloid
  - In conclusion: it is only the matrix  $\mathbf{A}$  which determines the shape and characteristics of the decision function

## Exercise: Consider a 3 dimensional space and cubic polynomial decision functions

1. How many terms are needed to represent a decision function if only cubic and linear functions are assumed?
2. Present the general 4<sup>th</sup> order polynomial decision function for a 2 dimensional pattern space
3. Let  $\mathbb{R}^3$  be the original pattern space and let the decision function associated with the pattern classes  $\omega_1$  and  $\omega_2$  be:

$$g(\mathbf{x}) = 2x_1^2 + x_3^2 + x_2x_3 + 4x_1 - 2x_2 + 1$$

for which  $g(\mathbf{x}) > 0$  if  $\mathbf{x} \in \omega_1$  and  $g(\mathbf{x}) < 0$  if  $\mathbf{x} \in \omega_2$

- a) Rewrite  $g(\mathbf{x})$  as  $g(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} + c$
- b) Determine the class of each of the following pattern vectors:

$$(1,1,1), (1,10,0), (0,1/2,0)$$

- Positive Definite Matrices

1. A square matrix  $\mathbf{A}$  is *positive definite* if  $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$  for all nonzero column vectors  $\mathbf{x}$ .
2. It is *negative definite* if  $\mathbf{x}^T \mathbf{A} \mathbf{x} < 0$  for all nonzero  $\mathbf{x}$ .
3. It is *positive semi-definite* if  $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$ .
4. And *negative semi-definite* if  $\mathbf{x}^T \mathbf{A} \mathbf{x} \leq 0$  for all  $\mathbf{x}$ .

These definitions are hard to check directly and you might as well forget them for all practical purposes.



More useful in practice are the following properties, which hold when the matrix  $\mathbf{A}$  is symmetric and which are easier to check.

The  *$i$ -th principal minor* of  $\mathbf{A}$  is the matrix  $\mathbf{A}_i$  formed by the first  $i$  rows and columns of  $\mathbf{A}$ . So, the first principal minor of  $\mathbf{A}$  is the matrix  $\mathbf{A}_1 = (a_{11})$ , the second principal minor is the matrix:

$$\mathbf{A}_2 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \text{ and so on.}$$

- The matrix  $\mathbf{A}$  is positive definite if all its principal minors  $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n$  have strictly positive determinants
- If these determinants are non-zero and alternate in signs, starting with  $\det(\mathbf{A}_1) < 0$ , then the matrix  $\mathbf{A}$  is negative definite
- If the determinants are all non-negative, then the matrix is positive semi-definite
- If the determinant alternate in signs, starting with  $\det(\mathbf{A}_1) \leq 0$ , then the matrix is negative semi-definite

To fix ideas, consider a  $2 \times 2$  symmetric matrix:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

- It is positive definite if:
  - a)  $\det(\mathbf{A}_1) = a_{11} > 0$
  - b)  $\det(\mathbf{A}_2) = a_{11}a_{22} - a_{12}a_{12} > 0$
- It is negative definite if:
  - a)  $\det(\mathbf{A}_1) = a_{11} < 0$
  - b)  $\det(\mathbf{A}_2) = a_{11}a_{22} - a_{12}a_{12} > 0$
- It is positive semi-definite if:
  - a)  $\det(\mathbf{A}_1) = a_{11} \geq 0$
  - b)  $\det(\mathbf{A}_2) = a_{11}a_{22} - a_{12}a_{12} \geq 0$
- And it is negative semi-definite if:
  - a)  $\det(\mathbf{A}_1) = a_{11} \leq 0$
  - b)  $\det(\mathbf{A}_2) = a_{11}a_{22} - a_{12}a_{12} \geq 0.$

## Exercise 2:

Let  $\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix}$

1. Compute the decision boundary assigned to the matrix  $\mathbf{A}$  ( $g(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} + c$ ) in the case where  $\mathbf{b}^T = (1, 2)$  and  $c = -3$
2. Solve  $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$  and find the shape and the characteristics of the decision boundary separating two classes  $\omega_1$  and  $\omega_2$
3. Classify the following points:
  - $\mathbf{x}^T = (0, -1)$
  - $\mathbf{x}^T = (1, 1)$

## Solution of Exercise 2:

1. 
$$\begin{aligned} g(\mathbf{x}) &= (x_1, x_2) \begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + (x_1, x_2) \begin{pmatrix} 1 \\ 2 \end{pmatrix} - 3 \\ &= (2x_1 + x_2, x_1 + 4x_2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + x_1 + 2x_2 - 3 \\ &= 2x_1^2 + x_1x_2 + x_1x_2 + 4x_2^2 + x_1 + 2x_2 - 3 \\ &= 2x_1^2 + 4x_2^2 + 2x_1x_2 + x_1 + 2x_2 - 3 \end{aligned}$$

2. For  $\lambda_1 = 3 + \sqrt{2}$  using  $\begin{pmatrix} 2 - \lambda & 1 \\ 1 & 4 - \lambda \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$ , we obtain :

$$\begin{cases} (-1 - \sqrt{2})x_1 + x_2 = 0 \\ x_1 + (1 - \sqrt{2})x_2 = 0 \end{cases} \Leftrightarrow (-1 - \sqrt{2})x_1 + x_2 = 0$$

This latter equation is a straight line colinear to the vector:

$$\vec{\mathbf{v}}_1 = (1, 1 + \sqrt{2})^T$$

For  $\lambda_2 = 3 - \sqrt{2}$  using  $\begin{pmatrix} 2 - \lambda & 1 \\ 1 & 4 - \lambda \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$ , we obtain :

$$\begin{cases} (\sqrt{2} - 1)x_1 + x_2 = 0 \\ x_1 + (1 + \sqrt{2})x_2 = 0 \end{cases} \Leftrightarrow (\sqrt{2} - 1)x_1 + x_2 = 0$$

This latter equation is a straight line colinear to the vector:

$$\vec{\mathbf{v}}_2 = (1, 1 - \sqrt{2})^T$$

The ellipsis decision boundary has two axes, which are respectively colinear to the vectors  $\mathbf{v}_1$  and  $\mathbf{v}_2$

3.  $\mathbf{x} = (0, -1)^T \Rightarrow g(0, -1) = -1 < 0 \Rightarrow \mathbf{x} \in \omega_2$

$$\mathbf{x} = (1, 1)^T \Rightarrow g(1, 1) = 8 > 0 \Rightarrow \mathbf{x} \in \omega_1$$

# Generalized linear discriminant

$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\mathbf{x})$$

or

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

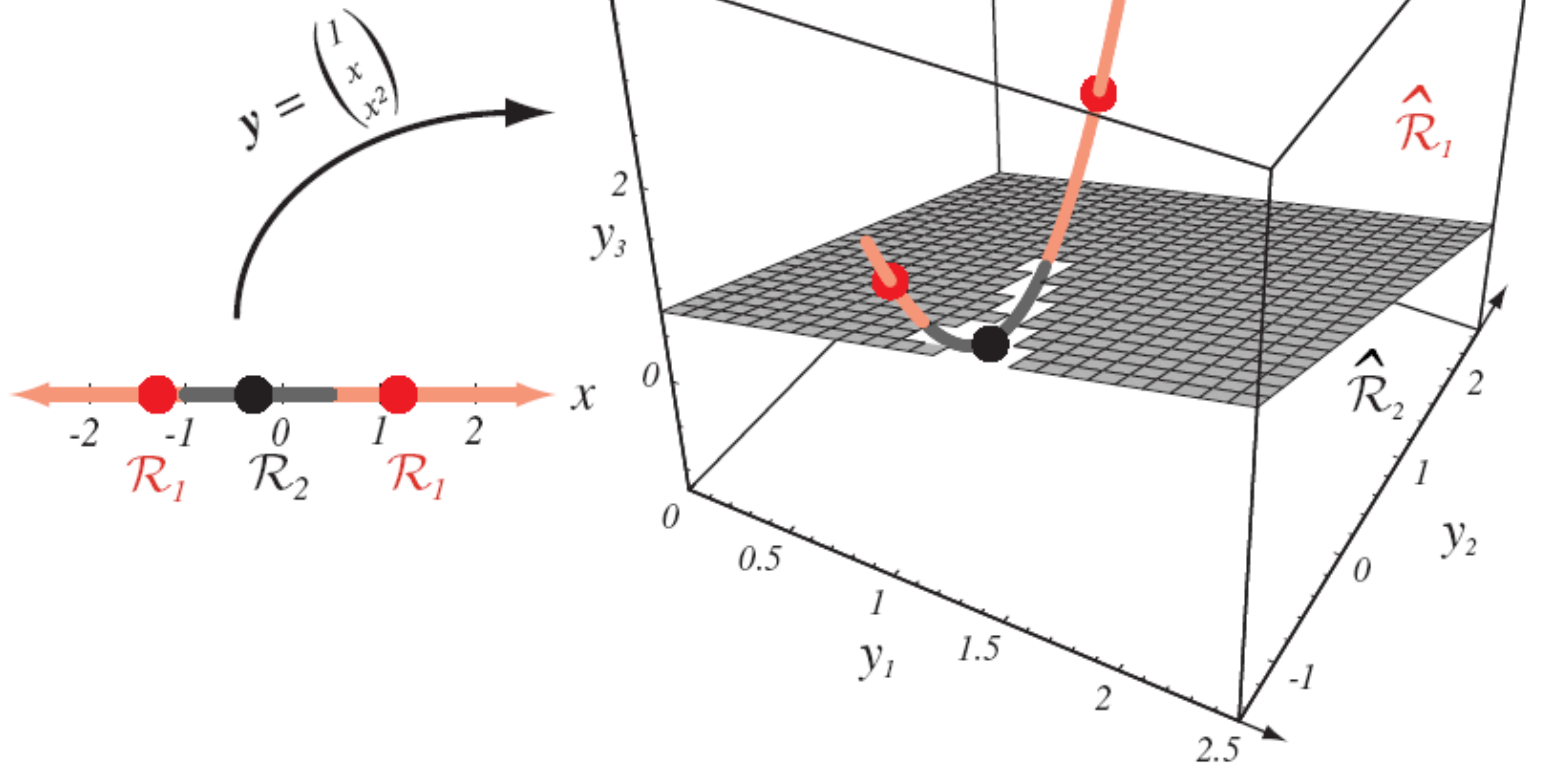
Let the quadratic discriminant function be  $g(x) = a_1 + a_2x + a_3x^2$  so that the three-dimensional vector  $\mathbf{y}$  is given by

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

See next Fig

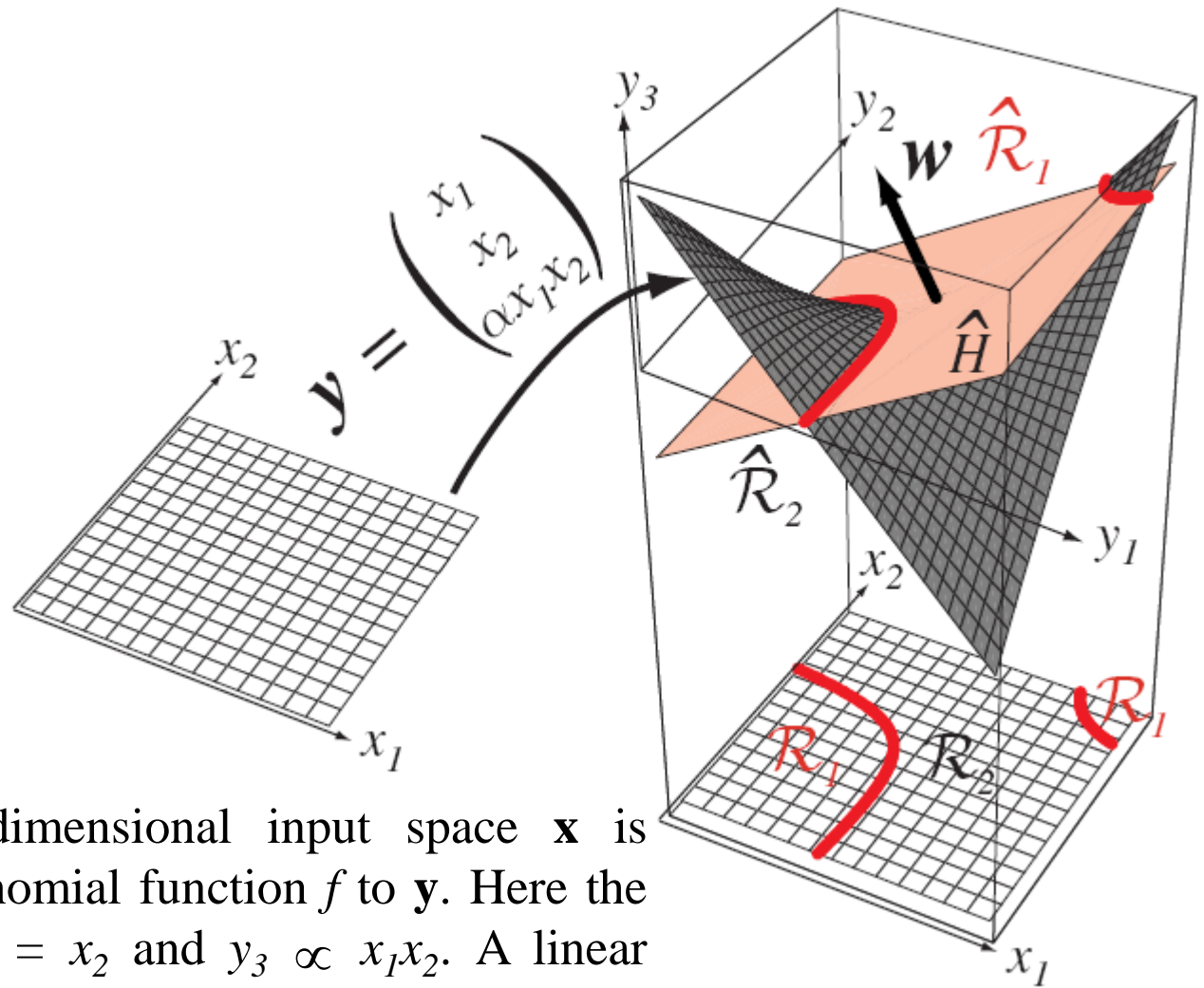
Although the decision regions in  $\mathbf{y}$ -space are convex, this is by no means the case in  $x$ -space. More generally speaking, even with relatively simple functions  $y_i(x)$ , decision surfaces induced in an  $x$ -space can be fairly complex (Fig. 5.6).

Error in figure:  
Plane should pass  
through origin



**Figure 5.5:** The mapping  $\mathbf{y} = (1, x, x^2)^t$  takes a line and transforms it to a parabola in three dimensions. A plane splits the resulting  $\mathbf{y}$  space into regions corresponding to two categories, and this in turn gives a non-simply connected decision region in the one-dimensional  $x$  space.





**Figure 5.6:** The two-dimensional input space  $\mathbf{x}$  is mapped through a polynomial function  $f$  to  $\mathbf{y}$ . Here the mapping is  $y_1 = x_1$ ,  $y_2 = x_2$  and  $y_3 \propto x_1x_2$ . A linear discriminant in this transformed space is a hyperplane, which cuts the surface. Points to the positive side of the hyperplane  $\hat{H}$  correspond to category  $\omega_1$ , and those beneath it  $\omega_2$ . Here, in terms of the  $\mathbf{x}$  space,  $\mathcal{R}_1$  is a not simply connected.

In the particular case of the linear discriminant function

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^d w_i x_i = \sum_{i=0}^d w_i x_i$$

where we set  $x_0 = 1$

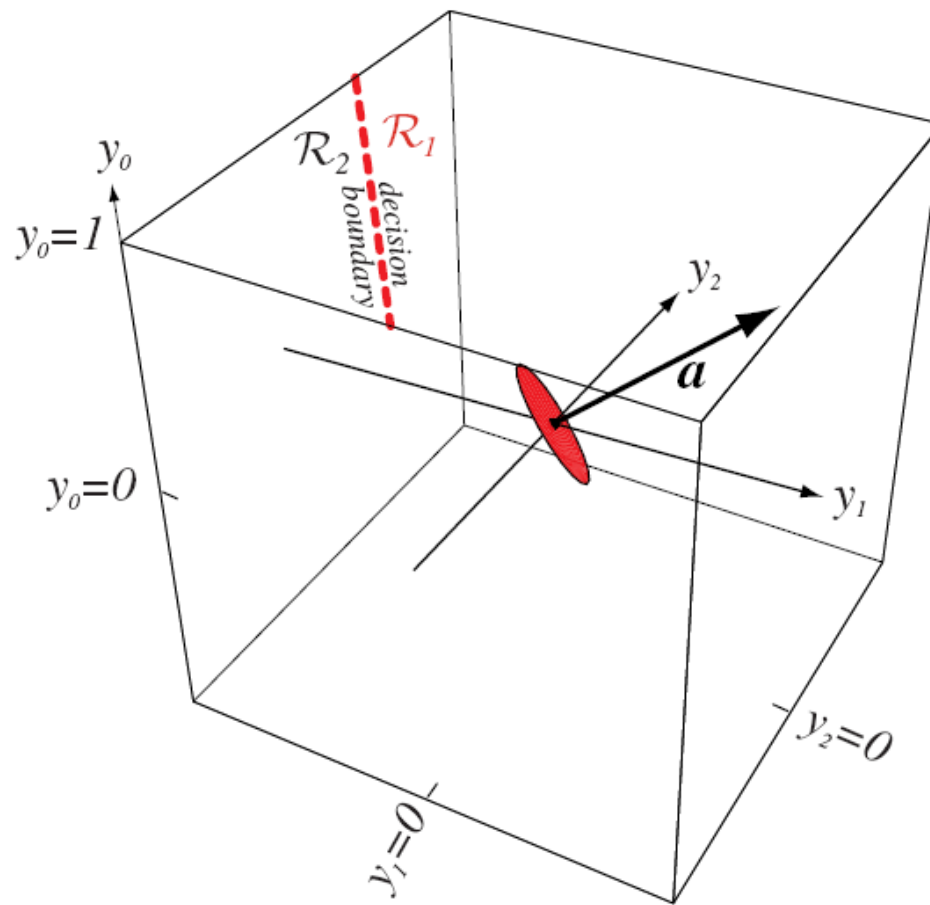
$$\mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix},$$

and  $\mathbf{y}$  is sometimes called an *augmented feature vector*.  
Likewise, an augmented weight vector can be written as:

$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

The addition of a constant component to  $\mathbf{x}$  preserves all distance relationships among samples. The resulting  $\mathbf{y}$  vectors all lie in a  $d$ -dimensional subspace, which is the  $\mathbf{x}$ -space itself. The hyperplane decision surface  $\hat{H}$  defined by  $\mathbf{a}^t \mathbf{y} = 0$  passes through the origin in  $\mathbf{y}$ -space, even though the corresponding hyperplane  $\mathbf{H}$  can be in any position in  $\mathbf{x}$ -space.

The distance from  $\mathbf{y}$  to  $\hat{H}$  is given by  $|\mathbf{a}^t \mathbf{y}| / \|\mathbf{a}\|$ , or  $|g(\mathbf{x})| / \|\mathbf{a}\|$ . Since  $\|\mathbf{a}\| > \|\mathbf{w}\|$ , this distance is less than, or at most equal to the distance from  $\mathbf{x}$  to  $\mathbf{H}$ .

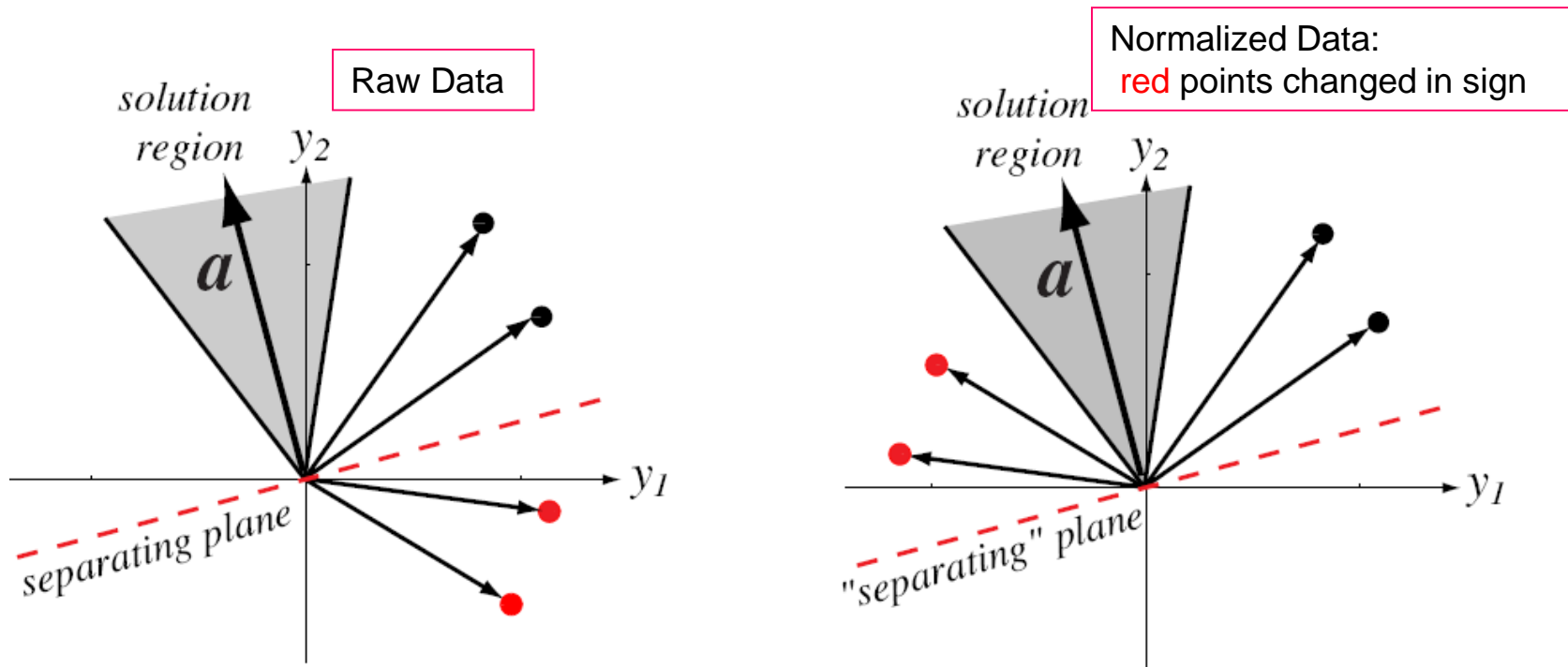


**Figure 5.7:** A three-dimensional augmented feature space  $\mathbf{y}$  and augmented weight vector  $\mathbf{a}$  (at the origin). The set of points for which  $\mathbf{a}^t \mathbf{y} = 0$  is a plane (or more generally, a hyperplane) perpendicular to  $\mathbf{a}$  and passing through the origin of  $\mathbf{y}$  space, as indicated by the red disk. Such a plane need not pass through the origin of the two-dimensional  $\mathbf{x}$ -space at the top, of course, as shown by the dashed line. Thus there exists an augmented weight vector  $\mathbf{a}$  that will lead to any straight decision line in  $\mathbf{x}$ -space.

# The Two-Category Linearly-Separable Case

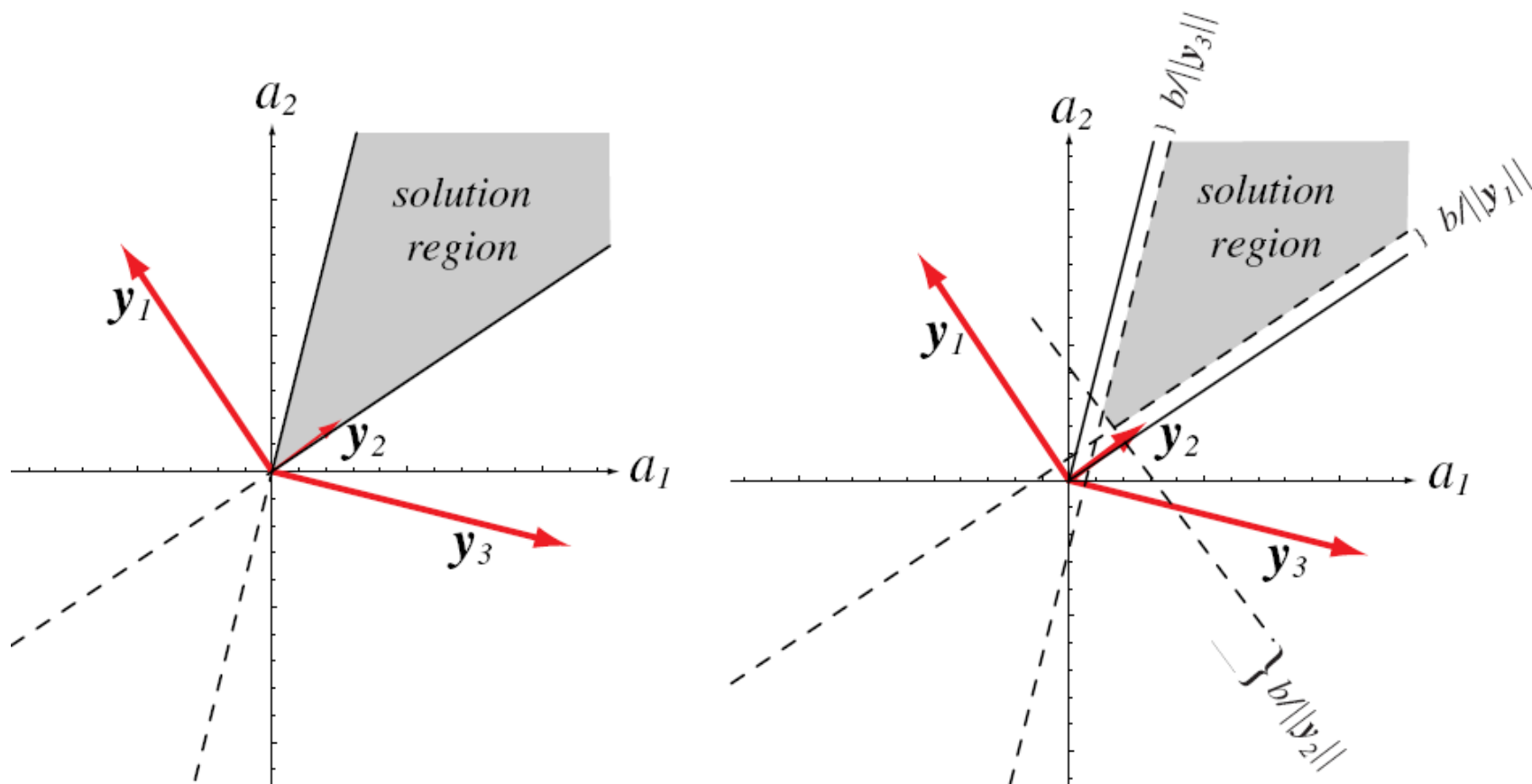
- Suppose now that we have a set of  $n$  samples  $\mathbf{y}_1, \dots, \mathbf{y}_n$ , some labeled  $\omega_1$  and some labeled  $\omega_2$ . We want to use these samples to determine the weights  $\mathbf{a}$  in a linear discriminant function  $g(\mathbf{x}) = \mathbf{a}^t \mathbf{y}$ .
- If there exists a solution (weight vector) for which the probability of error is very low, the samples are said to be linearly separable.
- A sample  $\mathbf{y}_i$  is classified correctly if  $\mathbf{a}^t \mathbf{y}_i > 0$  and  $\mathbf{y}_i$  is labelled  $\omega_1$  or if  $\mathbf{a}^t \mathbf{y}_i < 0$  separable and  $\mathbf{y}_i$  is labeled  $\omega_2$ .

- This suggests a “normalization”, the replacement of all samples labelled  $\omega_2$  by their negatives. With this “normalization” we can forget the labels and look for a weight vector  $\mathbf{a}$  such that  $\mathbf{a}^t \mathbf{y}_i > 0$  for all of the samples. Such a weight vector is called a *separating vector* or more generally a *solution vector*.
- The equation  $\mathbf{a}^t \mathbf{y}_i = 0$  defines a hyperplane through the origin of weight space having  $\mathbf{y}_i$  as a normal vector.
- The solution vector — if it exists — must be on the positive side of every hyperplane.



**Figure 5.8:** Four training samples (black for  $\omega_1$ , *red* for  $\omega_2$ ) and the *solution region* in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been “normalized” — i.e., changed in sign. Now the solution vector leads to a plane that places all “normalized” points on the same side.

- If we seek the minimum-length weight vector satisfying  $\mathbf{a}^t \mathbf{y}_i \geq b$  for all  $i$ , then  $b$  which is a positive constant called *the margin*.





# Gradient Descent Procedures

- To find solution to set of inequalities  $\mathbf{a}^t \mathbf{y}_i > 0$ , define a **criteria function  $J(\mathbf{a})$**  that is minimized if  $\mathbf{a}$  is a solution vector... We will define such a function later.
- Basic gradient descent is very simple. We start with some arbitrarily chosen weight vector  $\mathbf{a}(1)$  and compute the gradient vector  $\nabla J(\mathbf{a}(1))$ . The next value  $\mathbf{a}(2)$  is obtained by moving some distance from  $\mathbf{a}(1)$  in the direction of steepest descent, i.e., along the negative of the gradient.

$$\mathbf{a}(k + 1) = \mathbf{a}(k) - \eta(k) \nabla J(\mathbf{a}(k)),$$

where  $\eta$  is a positive scale factor or **learning rate** that sets the step size.

## Algorithm 1 (Basic gradient descent)

```
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
```

We now consider a principled method for setting the learning rate. Suppose that the criterion function can be well approximated by the second-order expansion around a value  $\mathbf{a}(k)$  as

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

# Taylor Series Expansion

$$\begin{aligned} F(x) = & F(x^*) + \frac{d}{dx}F(x) \Big|_{x=x^*} (x - x^*) \\ & + \frac{1}{2} \frac{d^2}{dx^2} F(x) \Big|_{x=x^*} (x - x^*)^2 + \dots \\ & + \frac{1}{n!} \frac{d^n}{dx^n} F(x) \Big|_{x=x^*} (x - x^*)^n + \dots \end{aligned}$$

# Vector Case

$$F(\mathbf{x}) = F(x_1, x_2, \dots, x_n)$$

$$\begin{aligned} F(\mathbf{x}) = & F(\mathbf{x}^*) + \frac{\partial}{\partial x_1} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_1 - x_1^*) + \frac{\partial}{\partial x_2} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_2 - x_2^*) \\ & + \dots + \frac{\partial}{\partial x_n} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_n - x_n^*) + \frac{1}{2} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_1 - x_1^*)^2 \\ & + \frac{1}{2} \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_1 - x_1^*)(x_2 - x_2^*) + \dots \end{aligned}$$

# Matrix Form

$$F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*) \\ + \frac{1}{2} (\mathbf{x} - \mathbf{x}^*)^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*) + \dots$$

Gradient

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} F(\mathbf{x}) \end{bmatrix}$$

Hessian

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_1 \partial x_n} F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_2 \partial x_n} F(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_n^2} F(\mathbf{x}) \end{bmatrix}$$

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

- where  $\mathbf{H}$  is the Hessian matrix of second partial derivatives  $\partial^2 J / \partial a_i \partial a_j$  evaluated at  $\mathbf{a}(k)$ .
- Substituting  $\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k) \nabla J(\mathbf{a}(k))$ ,

*Taylor series expansion*

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

–  $J(\mathbf{a}(k+1))$  can be minimized by the choice (derivative vs.  $\eta(k)$ )

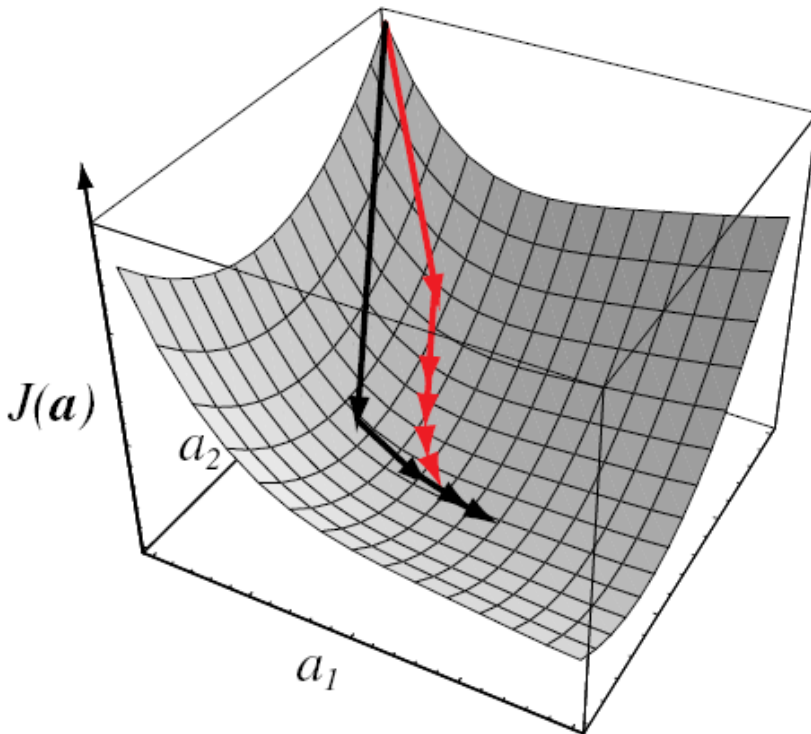
$$\eta(k) = \frac{\|\nabla J\|^2}{\nabla J^t \mathbf{H} \nabla J},$$

*optimum learning rate*

- An alternative approach, take the gradient of the second-order approximation and set it equal to zero: It is *Newton's algorithm*. (ref: [ANN Ch 9 lecture notes](#))

## Algorithm 2 (Newton descent)

```
1 begin initialize 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 a
6 end
```



**Figure 5.10:** The sequence of weight vectors given by a simple gradient descent method (red) and by Newton's (second order) algorithm (black). Newton's method typically leads to greater improvement per step, even when using optimal learning rates for both methods. However the added computational burden of inverting the Hessian matrix used in Newton's method is not always justified, and simple descent may suffice.

# Minimizing the Perceptron Criterion Function

- Criterion function for solving the linear inequalities  $\mathbf{a}^t \mathbf{y}_i > 0$ . The most obvious choice is to let  $J(\mathbf{a}; \mathbf{y}_1, \dots, \mathbf{y}_n)$  be the number of samples misclassified by  $\mathbf{a}$ . However, because this function is piecewise constant, it is obviously a poor candidate for a gradient search. A better choice is the *Perceptron criterion function*

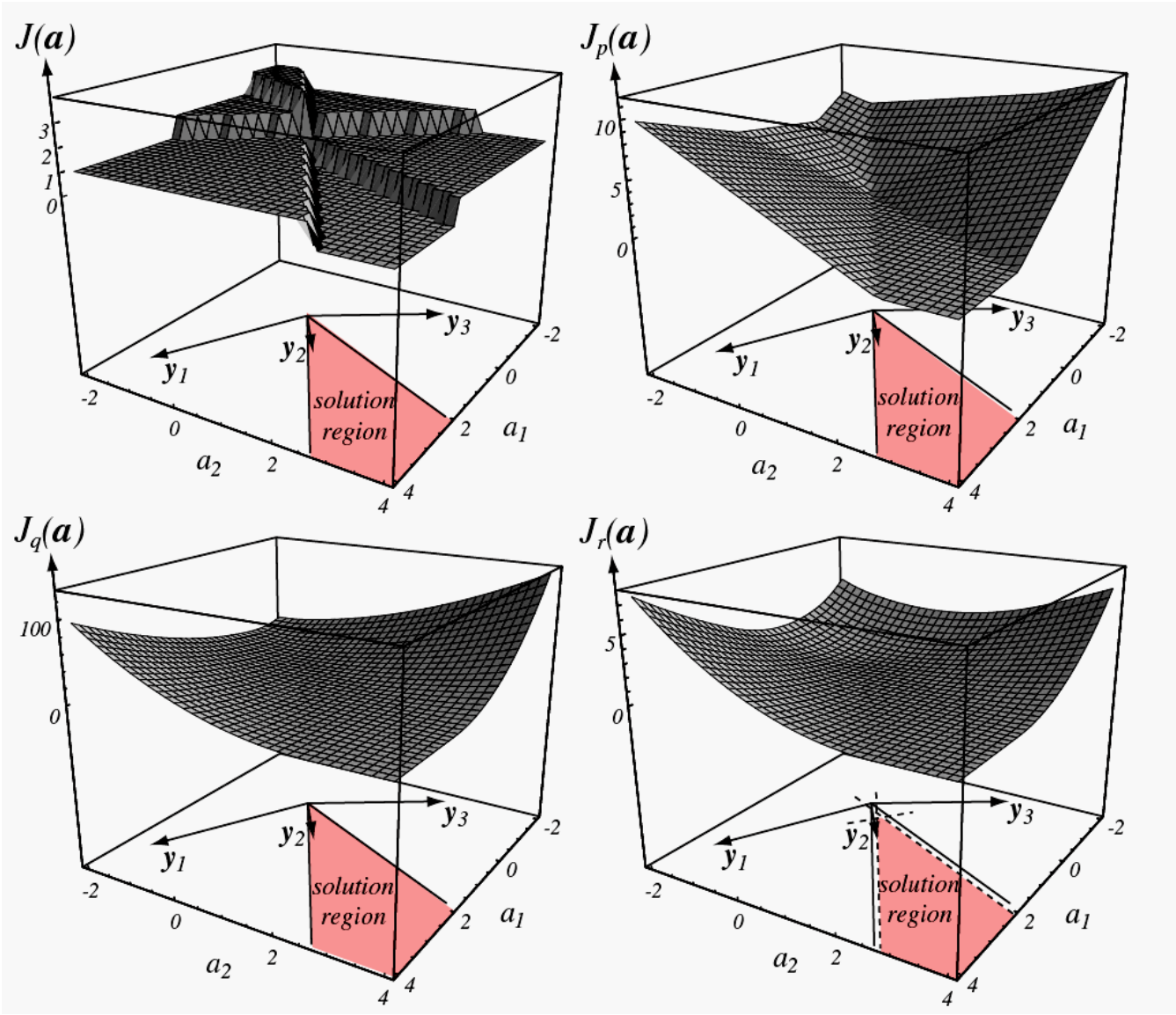
$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^t \mathbf{y}),$$

where  $\mathcal{Y}(\mathbf{a})$  is the set of samples misclassified by  $\mathbf{a}$ .

$J_p(\mathbf{a})$  is proportional to the sum of the distances from the misclassified samples to the decision boundary.



Perceptron criterion: Piecewise linear, acceptable for gradient descent



No of misclassified samples: Piecewise constant, unacceptable

Squared error: Useful when patterns are not linearly separable

Squared Error with margin

**FIGURE 5.11.** Four learning criteria as a function of weights in a linear classifier. At the upper left is the total number of patterns misclassified, which is piecewise constant and hence unacceptable for gradient descent procedures. At the upper right is the **Perceptron criterion**, which is piecewise linear and acceptable for gradient descent. The lower left is **squared error**, which has nice analytic properties and is useful even when the patterns are not linearly separable. The lower right is the **square error with margin**. A designer may adjust the margin  $b$  in order to force the solution vector to lie toward the middle of the  $b=0$  solution region in hopes of improving generalization of the resulting classifier.


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

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

### Algorithm 3 (Batch Perceptron)

```
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
```

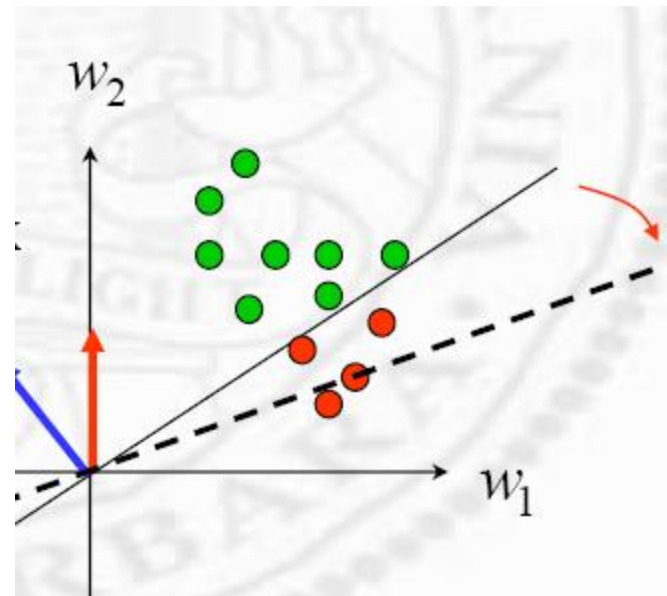
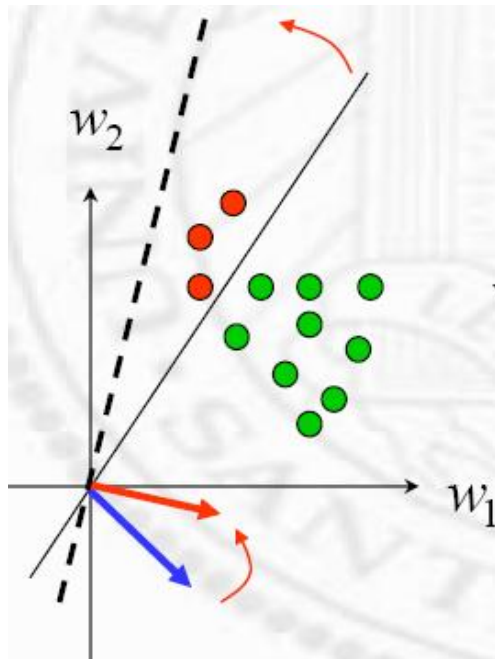
consider all  
examples  
misclassified

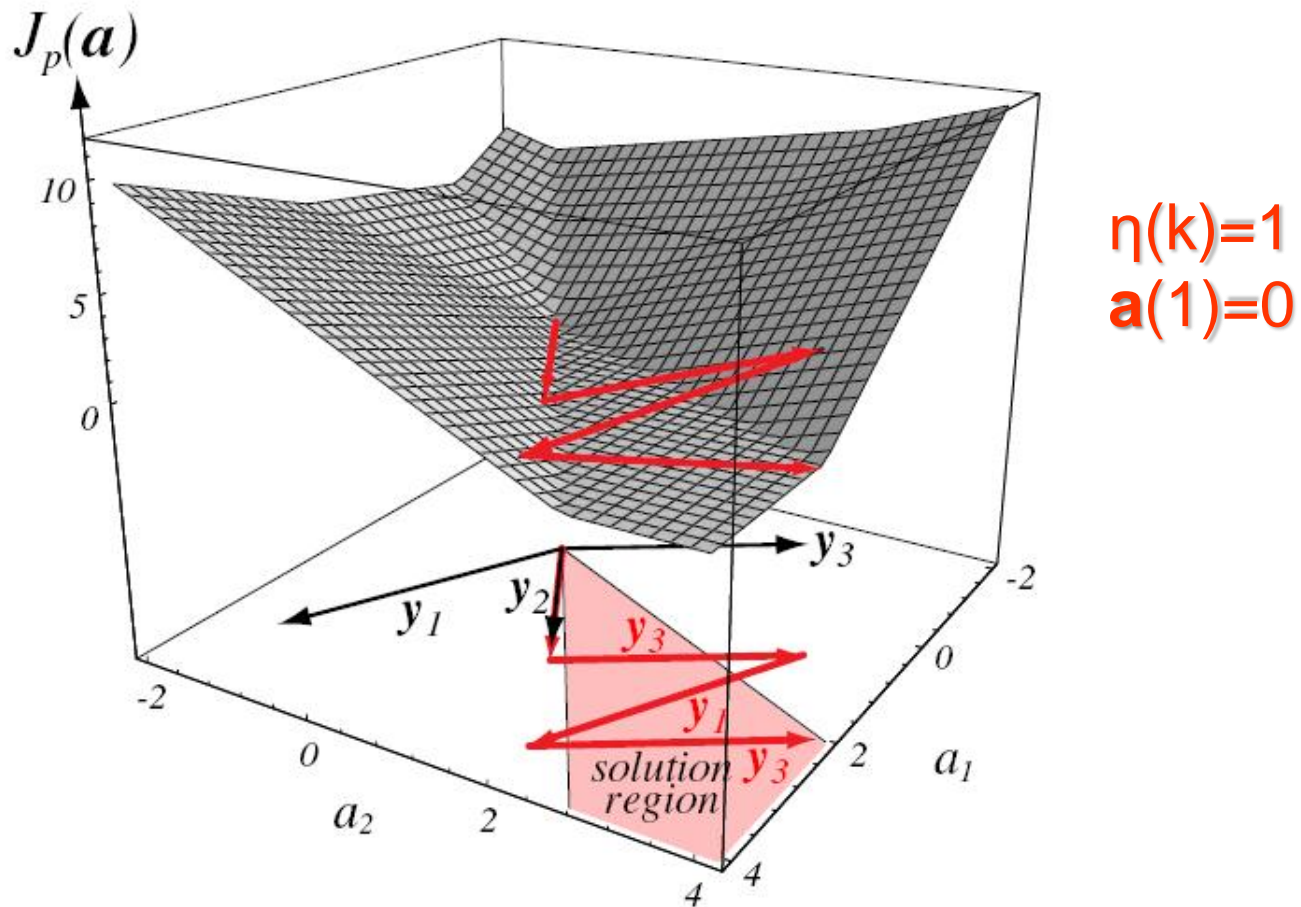


We use the term “**batch**” training to refer to the fact that (in general) a large group of samples is used when computing each weight update.

# Perceptron rule (cont'd)

- Move the hyperplane so that training samples are on its positive side.





**Figure 5.12:** *The weight vector begins at 0, and the algorithm sequentially adds to it vectors equal to the “normalized” misclassified patterns themselves. In the example shown, this sequence is,  $\mathbf{y}_1 + \mathbf{y}_2 + \mathbf{y}_3$ ,  $\mathbf{y}_3$ ,  $\mathbf{y}_1$ ,  $\mathbf{y}_3$ , at which time the vector lies in the solution region and iteration terminates. Note that the second update (by  $\mathbf{y}_3$ ) takes the candidate vector farther from the solution region than after the first update (In an alternate, batch method, all the misclassified points are added at each iteration step leading to a smoother trajectory in weight space.)*

# Convergence of Single-Sample Correction (simpler than batch)

$\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   
 $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_1, \mathbf{y}_2, \dots$

$\mathbf{y}^k$  is one of the  $n$  samples  $\mathbf{y}_1, \dots, \mathbf{y}_n$ , and where each  $\mathbf{y}^k$  is misclassified.

$$\begin{aligned} \mathbf{a}(1) & \text{ arbitrary} \\ \mathbf{a}(k+1) &= \mathbf{a}(k) + \mathbf{y}^k \quad k \geq 1 \end{aligned}$$

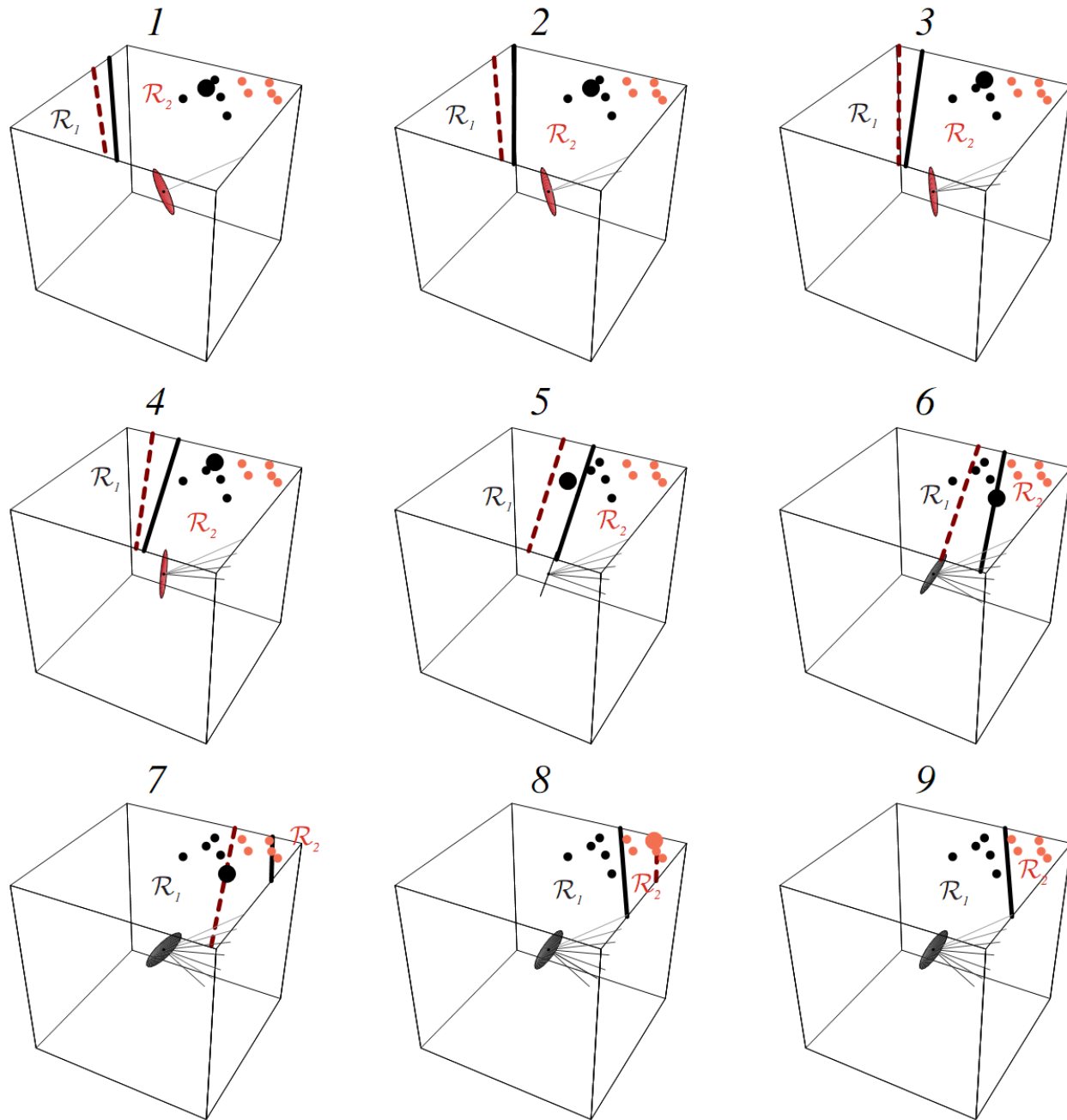
## ■ Algorithm 4. (Fixed-Increment Single-Sample Perceptron)

```
1 begin initialize  $\mathbf{a}, k \leftarrow 0$ 
2   do  $k \leftarrow (k + 1) \bmod n$ 
3     if  $\mathbf{y}^k$  is misclassified by  $\mathbf{a}$  then  $\mathbf{a} \leftarrow \mathbf{a} + \mathbf{y}^k$ 
4     until all patterns properly classified
5   return  $\mathbf{a}$ 
6 end
```

$\eta(k)=1$

consider one example at a time





**Theorem 5.1** (Perceptron Convergence) If training samples are linearly separable then the sequence of weight vectors given by Algorithm 4 will terminate at a solution vector.

Proof? p230

## Some Direct Generalizations

Correction whenever  $\mathbf{a}^t(k)\mathbf{y}^k$  fails to exceed a margin  $b$

$$\text{Update: } \left. \begin{array}{l} \mathbf{a}(1) \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{y}^k \end{array} \right\} \begin{array}{l} \text{arbitrary} \\ k \geq 1, \end{array}$$

Now  $\mathbf{a}^t(k)\mathbf{y}^k \leq b$  for all  $k$ .



■ **Algorithm 5. (Variable-Increment Perceptron with Margin)**

```
1 begin initialize a, threshold  $\theta$ , margin  $b$ ,  $\eta(\cdot)$ ,  $k \leftarrow 0$   
2           do  $k \leftarrow (k + 1) \bmod n$   
3           if  $\mathbf{a}^t \mathbf{y}^k \leq b$  then  $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \mathbf{y}^k$   
4           until  $\mathbf{a}^t \mathbf{y}^k > b$  for all  $k$   
5 return a  
6 end
```

It can be shown that if the samples are linearly separable and if

$$\eta(k) \geq 0,$$

$$\lim_{m \rightarrow \infty} \sum_{k=1}^m \eta(k) = \infty \quad \& \quad \lim_{m \rightarrow \infty} \frac{\sum_{k=1}^m \eta^2(k)}{\left( \sum_{k=1}^m \eta(k) \right)^2} = 0,$$

then  $\mathbf{a}(k)$  converges to a solution vector  $\mathbf{a}$  satisfying  $\mathbf{a}^t \mathbf{y}_i > b$  for all  $i$

Another variant of interest is our original gradient descent algorithm for  $J_p$ ,

$\mathbf{a}(1)$  arbitrary

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

the set of training samples misclassified by  $\mathbf{a}(k)$ .

**Algorithm 6 (Batch variable increment Perceptron)**

```

1 begin initialize  $\mathbf{a}, \eta(\cdot), k = 0$ 
2   do  $k \leftarrow k + 1$ 
3      $\mathcal{Y}_k = \{\}$ 
4      $j = 0$ 
5     do  $j \leftarrow j + 1$ 
6       if  $\mathbf{y}_j$  is misclassified then Append  $\mathbf{y}_j$  to  $\mathcal{Y}_k$ 
7     until  $j = n$ 
8      $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$ 
9   until  $\mathcal{Y}_k = \{\}$ 
10 return  $\mathbf{a}$ 
11 end

```

From a practical viewpoint, we want to make wise choice of  $b$  and  $\eta(k)$

Skip

## Some Direct Generalizations: Winnow Algorithm

A close descendant of the Perceptron algorithm is the **Winnow algorithm**, which has applicability to separable training data. The key difference is that while the weight vector returned by the Perceptron algorithm has components  $a_i$  ( $i = 0, \dots, d$ ), in Winnow they are scaled according to  $2\sinh[a_i]$ . In one version, *the balanced Winnow algorithm*, there are separate “positive” and “negative” weight vectors,  $\mathbf{a}^+$  and  $\mathbf{a}^-$ , each associated with one of the two categories to be learned.

## Algorithm 7 (Balanced Winnow)

```
1 begin initialize  $\mathbf{a}^+, \mathbf{a}^-, \eta(\cdot), k \leftarrow 0, \alpha > 1$   
2         if  $\text{sign}[\mathbf{a}^{+t} \mathbf{y}_k - \mathbf{a}^{-t} \mathbf{y}_k] \neq z_k$  (pattern misclassified)  
3         then if  $z_k = +1$  then  $a_i^+ \leftarrow \alpha^{+y_i} a_i^+; a_i^- \leftarrow \alpha^{-y_i} a_i^-$  for all  $i$   
4         if  $z_k = -1$  then  $a_i^+ \leftarrow \alpha^{-y_i} a_i^+; a_i^- \leftarrow \alpha^{+y_i} a_i^-$  for all  $i$   
5         return  $\mathbf{a}^+, \mathbf{a}^-$   
6 end
```

### Advantages:

Convergence is faster than in a Perceptron because of proper setting of learning rate.

Each constituent value does not overshoot its final value.

Benefit is pronounced when there are a large number of irrelevant or redundant features learnt.

# Relaxation Procedures

The criterion function  $J_p$  is by no means the only function we can construct that is minimized when  $\mathbf{a}$  is a solution vector. A close but distinct relative is

$$J_q(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (\mathbf{a}^t \mathbf{y})^2,$$

the set of training samples misclassified by  $\mathbf{a}$ .

Like  $J_p$ ,  $J_q$  focuses attention on the misclassified samples.

Its chief difference is that its gradient is continuous, whereas the gradient of  $J_p$  is not. Thus,  $J_q$  presents a smoother surface to search (Fig. 5.11).

Unfortunately,  $J_q$  is so smooth near the boundary of the solution region that the sequence of weight vectors can converge to a point on the boundary. It is particularly embarrassing to spend some time following the gradient merely to reach the boundary point  $\mathbf{a} = 0$ .

Another problem with  $J_q$  is that its value can be dominated by the longest sample vectors. Both of these problems are avoided by the criterion function

$$J_r(\mathbf{a}) = \frac{1}{2} \sum_{\mathbf{y} \in \mathcal{Y}} \frac{(\mathbf{a}^t \mathbf{y} - b)^2}{\|\mathbf{y}\|^2}, \quad \nabla J_r = \sum_{\mathbf{y} \in \mathcal{Y}} \frac{\mathbf{a}^t \mathbf{y} - b}{\|\mathbf{y}\|^2} \mathbf{y},$$

the set of training samples for which  $\mathbf{a}^t \mathbf{y} \leq b$ .

## Algorithm 8 (Batch relaxation with margin)

```
1 begin initialize  $\mathbf{a}, \eta(\cdot), k = 0$   
2       do  $k \leftarrow k + 1$   
3          $\mathcal{Y}_k = \{\}$   
4          $j = 0$   
5         do  $j \leftarrow j + 1$   
6           if  $\mathbf{y}_j$  is misclassified then Append  $\mathbf{y}_j$  to  $\mathcal{Y}_k$   
7         until  $j = n$   
8          $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}} \frac{b - \mathbf{a}^t \mathbf{y}}{\|\mathbf{y}\|^2} \mathbf{y}$   
9       until  $\mathcal{Y}_k = \{\}$   
10 return  $\mathbf{a}$   
11 end
```

The single-sample  
correction rule

$$\left. \begin{array}{l} \mathbf{a}(1) \quad \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \eta \frac{b - \mathbf{a}^t(k) \mathbf{y}^k}{\|\mathbf{y}^k\|^2} \mathbf{y}^k, \end{array} \right\} \quad (35)$$

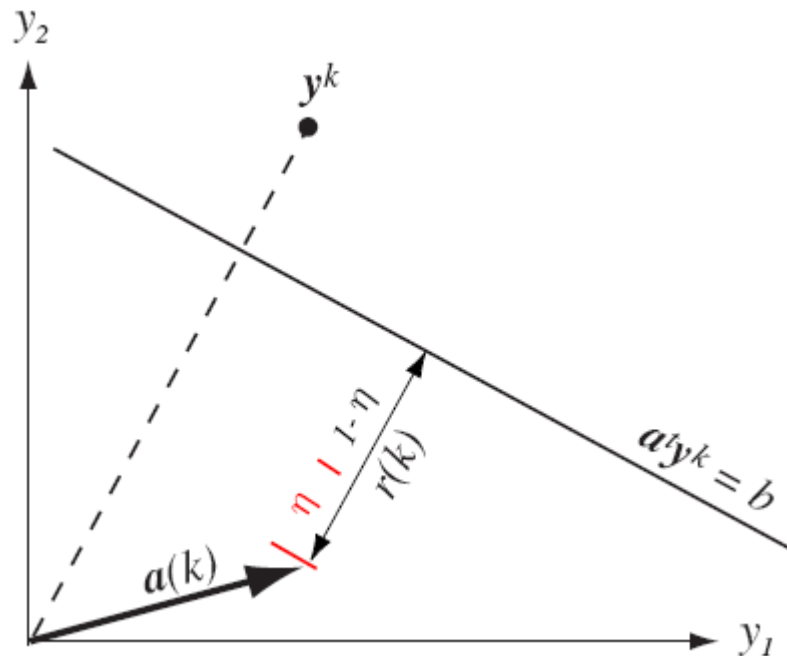
## Algorithm 9 (Single-sample relaxation with margin)

```
1 begin initialize  $\mathbf{a}$ ,  $\eta(\cdot)$ ,  $k = 0$ 
2       do  $k \leftarrow k + 1$ 
3           if  $\mathbf{y}_k$  is misclassified then  $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \frac{b - \mathbf{a}^t(k) \mathbf{y}^k}{\|\mathbf{y}^k\|^2} \mathbf{y}^k$ 
4           until all patterns properly classified
5       return  $\mathbf{a}$            or  $\mathbf{a}^t \mathbf{y}^k > b$  for all  $\mathbf{y}^k$ 
6 end
```

$r(k) = \frac{b - \mathbf{a}^t(k) \mathbf{y}^k}{\|\mathbf{y}^k\|}$  is the distance from  $\mathbf{a}(k)$  to the hyperplane  $\mathbf{a}^t \mathbf{y}^k = b$

If  $\eta = 1$ ,  $\mathbf{a}(k)$  is moved exactly to the hyperplane, so that the “**tension**” created by the inequality  $\mathbf{a}^t(k) \mathbf{y}^k \leq b$  is “**relaxed**” (Fig. 5.14).





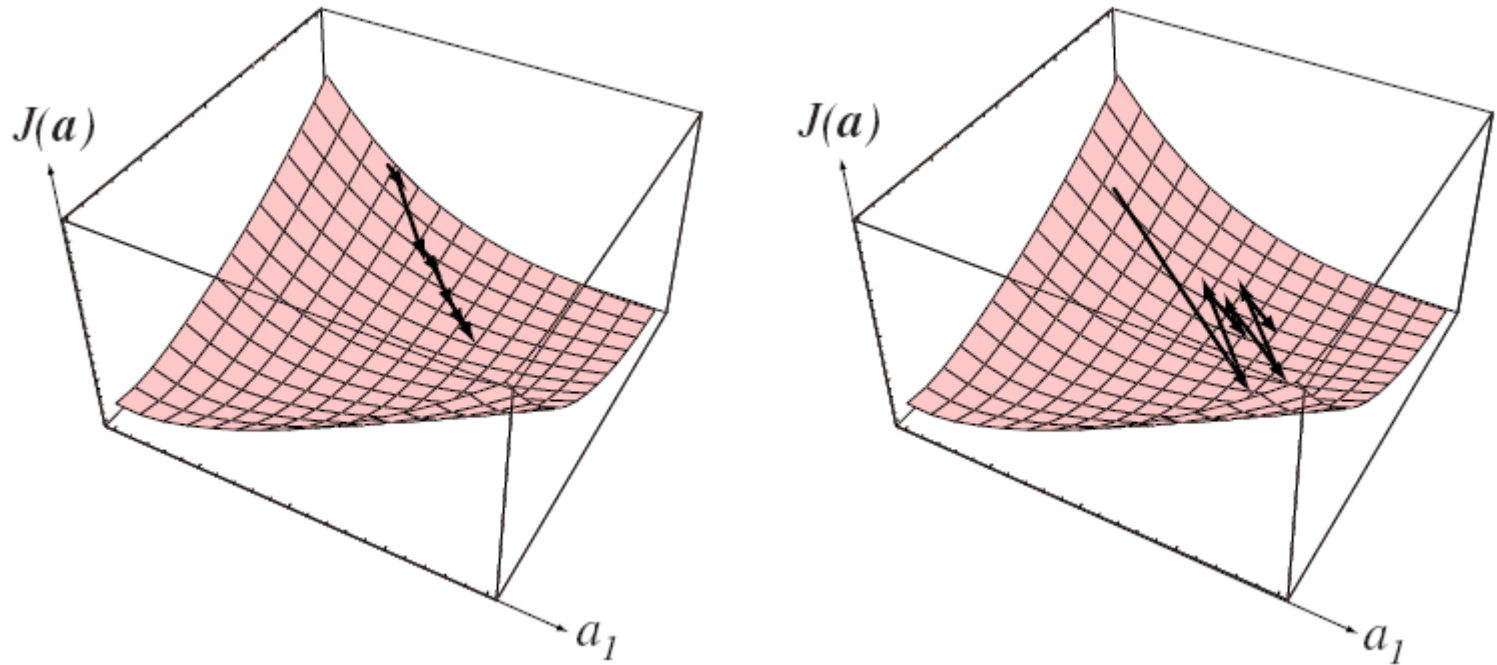
**Figure 5.14:** In each step of a basic relaxation algorithm, the weight vector is moved a proportion  $\eta$  of the way towards the hyperplane defined by  $\mathbf{a}^t \mathbf{y}^k = b$ .

---

From (35)  $\mathbf{a}^t(k+1)\mathbf{y}^k - b = (1 - \eta)(\mathbf{a}^t(k)\mathbf{y}^k - b)$ .

If  $\eta < 1$ , then  $\mathbf{a}^t(k+1)\mathbf{y}^k < b$  underrelaxation

If  $1 < \eta < 2$ , then  $\mathbf{a}^t(k+1)\mathbf{y}^k > b$  overrelaxation



**Figure 5.15:** At the left, under relaxation ( $\eta < 1$ ) leads to needlessly slow descent, or even failure to converge. Over-relaxation ( $1 < \eta < 2$ , shown in the right) describes overshooting; nevertheless convergence will ultimately be achieved.

# Nonseparable Behavior

- The Perceptron and relaxation procedures give us a number of simple methods for finding a separating vector when the samples are linearly separable. All of these methods are called *error-correcting procedures*, because they call for a modification error correcting procedure of the weight vector when and only when an error is encountered.
- When design sets are large, points are most likely not linearly separable!

- How does error correction procedures behave when points are not linearly separable?
  - Corrections will never cease in an error correction procedure
  - Infinite sequence of weight vectors
- Weight vectors are bounded
  - Empirical rule based on weight vector fluctuating near a terminal value
  - Averaging weight vectors can reduce risk of obtaining a bad solution

# Minimum Squared Error Procedures

- The criterion functions we have considered thus far have focused their attention on the misclassified samples. We shall now consider a criterion function that involves *all* of the samples.
- Where previously we have sought a weight vector  $\mathbf{a}$  making all of the inner products  $\mathbf{a}^t \mathbf{y}^i$  positive, now we shall try to make  $\mathbf{a}^t \mathbf{y}^i = b_i$ , where the  $b_i$  are some arbitrarily specified positive constants.
- Let  $\mathbf{Y}$  be the  $n$ -by- $\hat{d}$  matrix ( $\hat{d} = d + 1$ ) whose  $i$ th row is the vector  $\mathbf{y}_i^t$ , and let  $\mathbf{b}$  be the column vector  $\mathbf{b} = (b_1, \dots, b_n)^t$ .

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

or  $\mathbf{Y}\mathbf{a} = \mathbf{b}.$

If  $\mathbf{Y}$  were nonsingular, we could write  $\mathbf{a} = \mathbf{Y}^{-1}\mathbf{b}$ , however,  $\mathbf{Y}$  is rectangular. If we define the error vector  $\mathbf{e}$  by  $\mathbf{e} = \mathbf{Y}\mathbf{a} - \mathbf{b}$ , then one approach is to try to minimize the squared length of the error vector.

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

Weight vector  
to be determined

# MSE Solution based on Pseudoinverse

$$\nabla J_s = \sum_{i=1}^n 2(\mathbf{a}^t \mathbf{y}_i - b_i) \mathbf{y}_i = 2\mathbf{Y}^t (\mathbf{Y}\mathbf{a} - \mathbf{b})$$

Equating to zero  $\mathbf{Y}^t \mathbf{Y}\mathbf{a} = \mathbf{Y}^t \mathbf{b}$ ,

$$\begin{aligned} \mathbf{a} &= (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t \mathbf{b} \\ &= \mathbf{Y}^\dagger \mathbf{b}, \end{aligned} \quad \mathbf{Y}^\dagger \text{ is called the pseudoinverse of } \mathbf{Y}.$$

$$\mathbf{Y}^\dagger \equiv (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t \quad \text{note: } n > d+1$$

Note also that  $\mathbf{Y}^\dagger \mathbf{Y} = \mathbf{I}$ , but  $\mathbf{Y}\mathbf{Y}^\dagger \neq \mathbf{I}$  in general. However, a minimum-squared-error (MSE) solution always exists. In particular, if  $\mathbf{Y}^\dagger$  is defined more generally by

$$\mathbf{Y}^\dagger \equiv \lim_{\epsilon \rightarrow 0} (\mathbf{Y}^t \mathbf{Y} + \epsilon \mathbf{I})^{-1} \mathbf{Y}^t$$

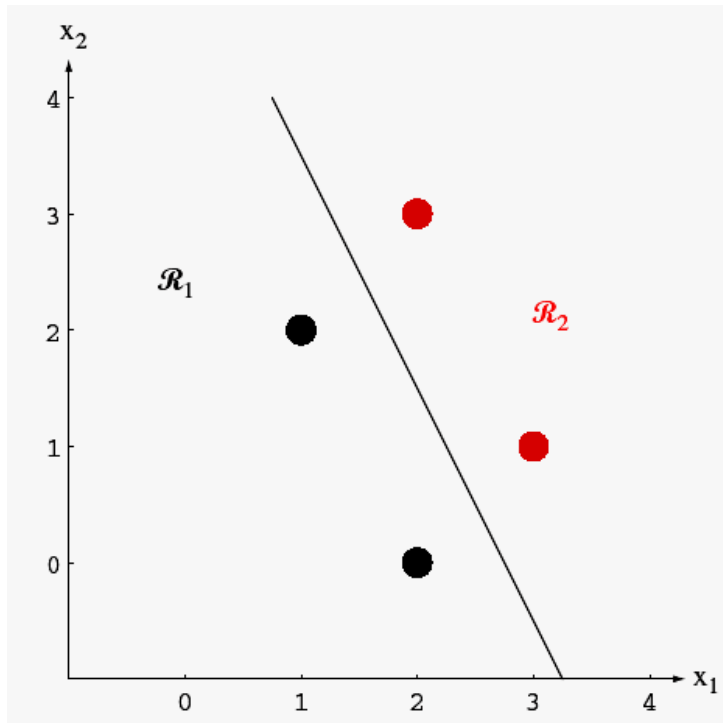
Closed form solution

The MSE solution depends on the margin vector  $\mathbf{b}$ .

# Example of a linear classifier by matrix pseudoinverse

Two classes;  $\omega_1: (1, 2)^t$  and  $(2, 0)^t$ , and  $\omega_2: (3, 1)^t$  and  $(2, 3)^t$

t



$$\mathbf{b} = (1, 1, 1, 1)^t.$$

$$\mathbf{Y} = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 0 \\ -1 & -3 & -1 \\ -1 & -2 & -3 \end{pmatrix}$$

$$\begin{aligned} \mathbf{Y}^\dagger &\equiv \lim_{\epsilon \rightarrow 0} (\mathbf{Y}^t \mathbf{Y} + \epsilon \mathbf{I})^{-1} \mathbf{Y}^t \\ &= \begin{pmatrix} 5/4 & 13/12 & 3/4 & 7/12 \\ -1/2 & -1/6 & -1/2 & -1/6 \\ 0 & -1/3 & 0 & -1/3 \end{pmatrix} \end{aligned}$$

$$\text{our solution is } \mathbf{a} = \mathbf{Y}^t \mathbf{b} = \begin{bmatrix} 11/3 \\ -4/3 \\ -2/3 \end{bmatrix} \quad 72$$



# Properties of MSE Procedure

- Related to Fisher's Linear Discriminant
- Asymptotic approximation to Bayes discriminant function
- Can be formulated as a gradient descent procedure

# MSE Relationship to Fisher's Linear Discriminant

- Show that with proper choice of the vector  $\mathbf{b}$  the MSE discriminant function  $\mathbf{a}^T \mathbf{y}$  is directly related to Fisher's linear discriminant
- Assume first  $n_1$  samples are labelled  $\omega_1$  and second  $n_2$  samples are labelled  $\omega_2$

$$Y = \begin{bmatrix} \mathbf{1}_1 & X_1 \\ -\mathbf{1}_2 & -X_2 \end{bmatrix} \text{ where } \mathbf{1}_i \text{ is a column vector } n_i \text{ ones}$$

and  $X_i$  is a matrix whose rows are the samples labeled  $\omega_i$

Partition  $\mathbf{a}$  and  $\mathbf{b}$  correspondingly

$$\mathbf{a} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} \text{ and with } \mathbf{b} = \begin{bmatrix} \frac{n}{n_1} \mathbf{1}_1 \\ \frac{n}{n_2} \mathbf{1}_2 \end{bmatrix}$$

This special choice of  $\mathbf{b}$  links the MSE solution to Fisher's Linear Discriminant

# MSE and Fisher's Linear Discriminant

- Define sample means  $m_i$  and pooled sample scatter

matrix  $\mathbf{S}_W$

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in D_i} \mathbf{x} \quad i = 1, 2$$

$$\mathbf{S}_W = \sum_{i=1}^2 \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t$$

and plug into MSE formulation yields

$$\mathbf{w} = \alpha n \mathbf{S}_W^{-1} (\mathbf{m}_1 - \mathbf{m}_2)$$

where  $\alpha$  is a scalar which is identical to the solution to the Fisher's linear discriminant except for a scale factor


- Decision rule: Decide  $\omega_1$  if  $\mathbf{w}^t(\mathbf{x} - \mathbf{m}) > 0$ ; otherwise decide  $\omega_2$


Proof:

$$\mathbf{Y}^\dagger \equiv (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t \quad \mathbf{Y}^t \mathbf{Y} \mathbf{a} = \mathbf{Y}^t \mathbf{b}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{1}_1 & \mathbf{X}_1 \\ -\mathbf{1}_2 & -\mathbf{X}_2 \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \frac{n}{n_1} \mathbf{1}_1 \\ \frac{n}{n_2} \mathbf{1}_2 \end{bmatrix}$$


$$\begin{bmatrix} \mathbf{1}_1^t & -\mathbf{1}_2^t \\ \mathbf{X}_1^t & -\mathbf{X}_2^t \end{bmatrix} \begin{bmatrix} \mathbf{1}_1 & \mathbf{X}_1 \\ -\mathbf{1}_2 & -\mathbf{X}_2 \end{bmatrix} \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{1}_1^t & -\mathbf{1}_2^t \\ \mathbf{X}_1^t & -\mathbf{X}_2^t \end{bmatrix} \begin{bmatrix} \frac{n}{n_1} \mathbf{1}_1 \\ \frac{n}{n_2} \mathbf{1}_2 \end{bmatrix}$$


$$\begin{bmatrix} n & (n_1 \mathbf{m}_1 + n_2 \mathbf{m}_2)^t \\ (n_1 \mathbf{m}_1 + n_2 \mathbf{m}_2) & \mathbf{S}_W + n_1 \mathbf{m}_1 \mathbf{m}_1^t + n_2 \mathbf{m}_2 \mathbf{m}_2^t \end{bmatrix} \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} 0 \\ n(\mathbf{m}_1 - \mathbf{m}_2) \end{bmatrix}$$


$$\begin{cases} w_0 = -\mathbf{m}^t \mathbf{w}, \\ \left[ \frac{1}{n} \mathbf{S}_W + \frac{n_1 n_2}{n^2} (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t \right] \mathbf{w} = \mathbf{m}_1 - \mathbf{m}_2. \end{cases} \quad 76$$

Since the vector  $(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t \mathbf{w}$  is in the direction of  $\mathbf{m}_1 - \mathbf{m}_2$  for any value of  $\mathbf{w}$ , we can write

$$\frac{n_1 n_2}{n^2} (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t \mathbf{w} = (1 - \alpha)(\mathbf{m}_1 - \mathbf{m}_2),$$

where  $\alpha$  is some scalar

$$\longrightarrow \mathbf{w} = \alpha n \mathbf{S}_W^{-1} (\mathbf{m}_1 - \mathbf{m}_2),$$

# MSE Relationship to Bayes

- If  $\mathbf{b}=\mathbf{1}_n$  MSE approaches a minimum squared error approximation to Bayes discriminant function

$$g_0(\mathbf{x}) = P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x})$$

in the limit as number of samples approaches infinity.

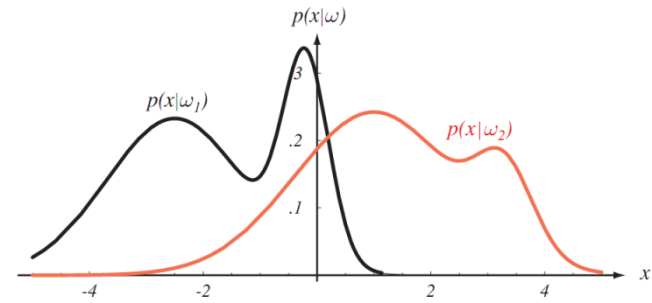
$$g(\mathbf{x}) = \mathbf{a}^t \mathbf{y}$$

# MSE Approximation to Bayes

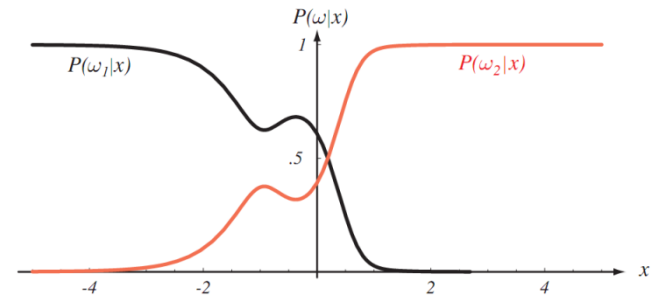
- If  $\mathbf{b}=\mathbf{1}_n$  MSE solution approaches a minimum mean squared approximation to Bayes discriminant function

However MSE does not necessarily minimize probability of error

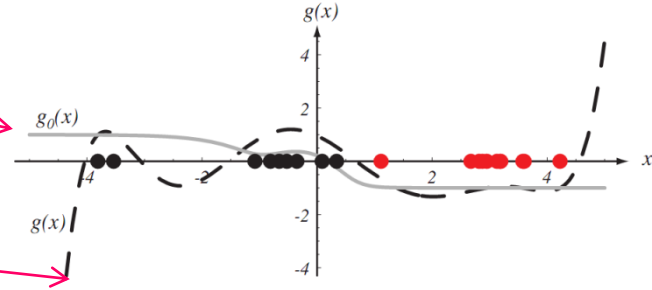
Class conditional densities



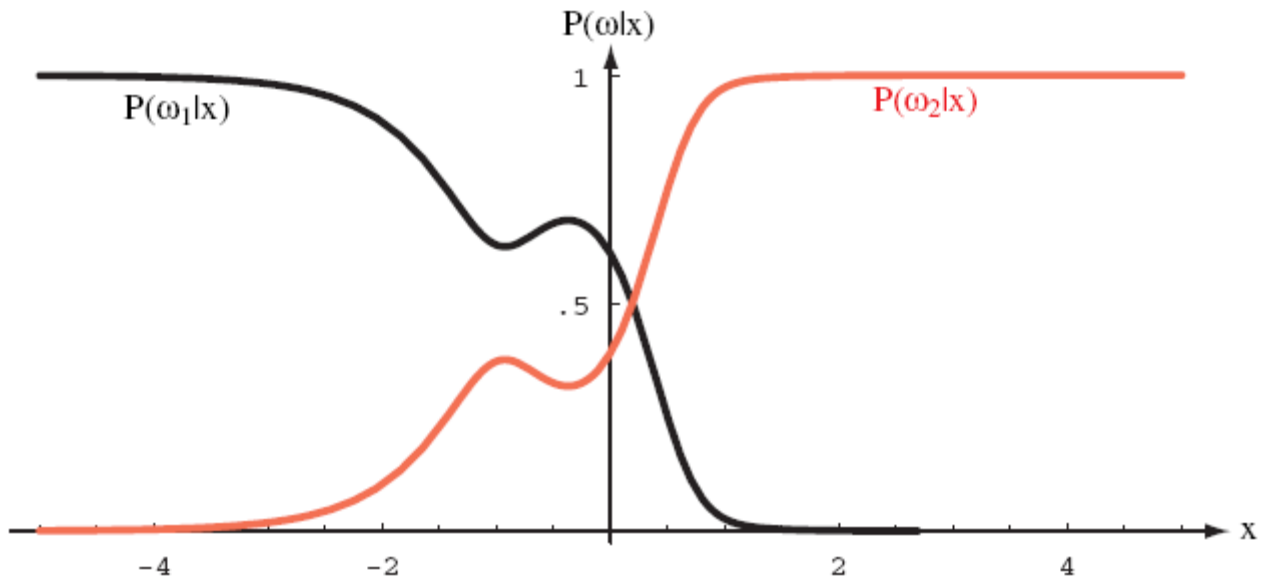
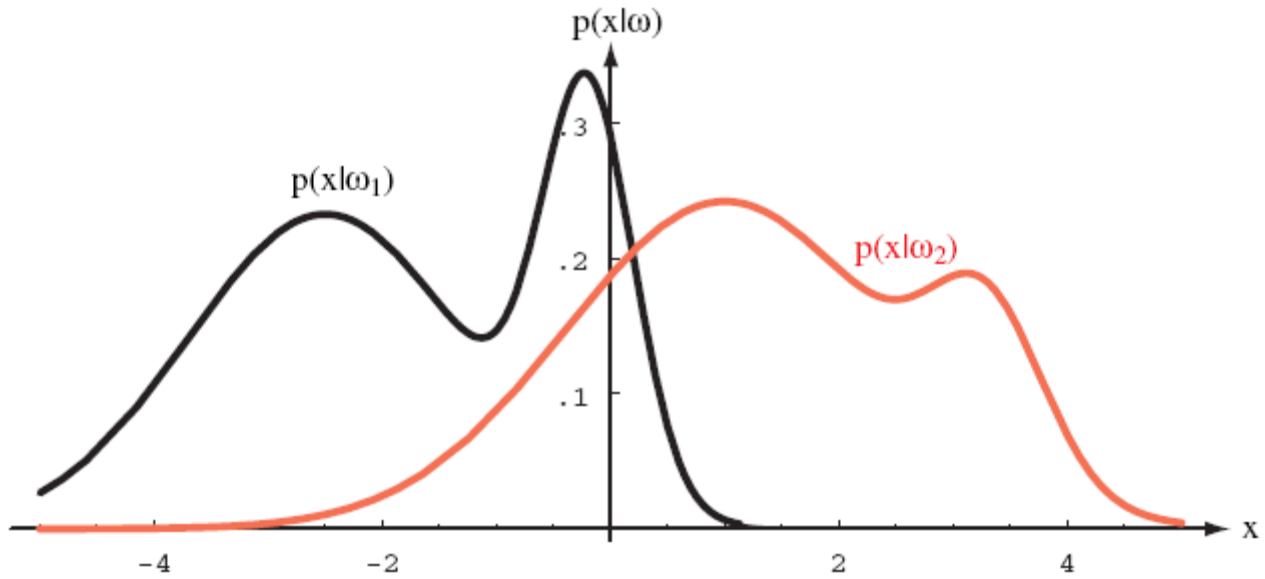
Posteriors



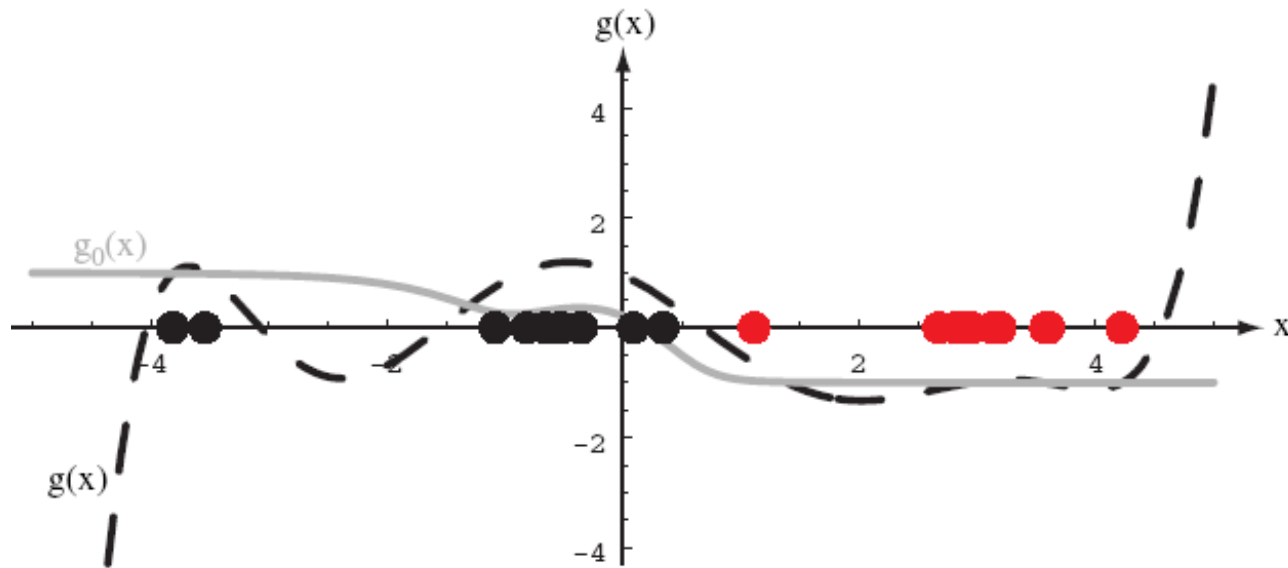
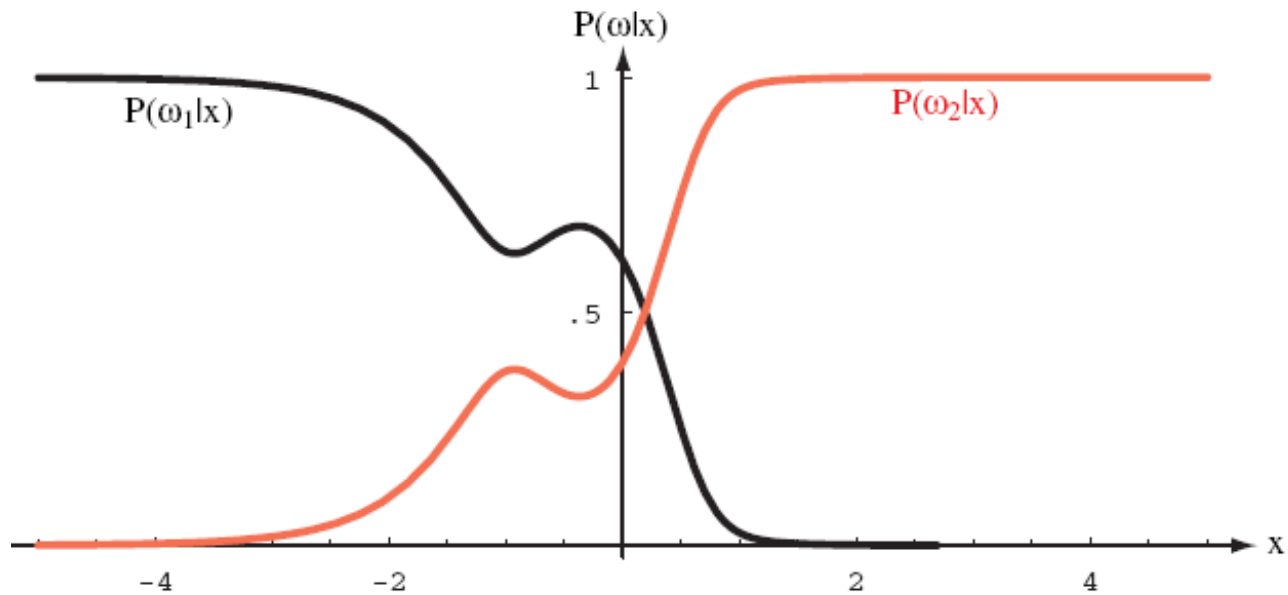
Bayes discriminant function



MSE solution (best approximation in region of data points)







$$g_0(x) = P(\omega_1|x) - P(\omega_2|x)$$

# MSE Solution using Gradient Descent

- Criterion function  $J_s(\mathbf{a}) = \|\mathbf{Y}\mathbf{a}-\mathbf{b}\|^2$  could be minimized by gradient descent
- Advantage over pseudo-inverse:
  - Problem when  $\mathbf{Y}^t\mathbf{Y}$  is *singular*
  - Avoids need for working with large matrices
  - Computation involved is a feedback scheme that copes with round off or truncation

# The Widrow-Hoff Procedure

Since  $\nabla J_s = 2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b})$ , the obvious update rule is

$$\left. \begin{array}{l} \mathbf{a}(1) \quad \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\mathbf{Y}^t(\mathbf{Y}\mathbf{a}_k - \mathbf{b}). \end{array} \right\}$$

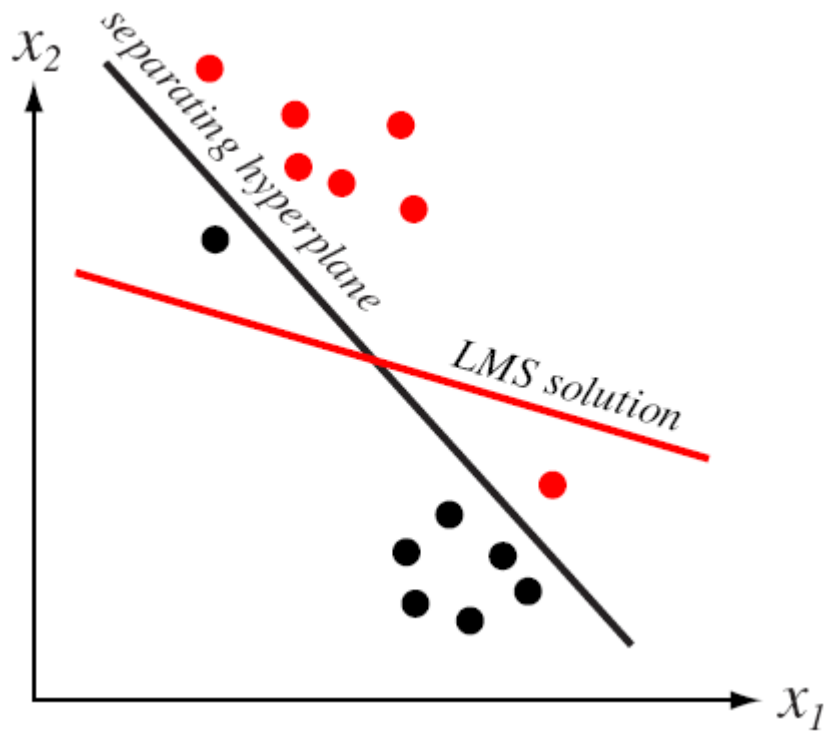
Can be reduced for storage requirement to the rule where samples are considered sequentially:

$$\left. \begin{array}{l} \mathbf{a}(1) \quad \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)(b_k - \mathbf{a}(k)^t \mathbf{y}^k) \mathbf{y}^k, \end{array} \right\}$$

# the *Widrow-Hoff* or LMS rule (Least-Mean-Squared)

## Algorithm 10 (LMS)

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



**Figure 5.17:** The LMS algorithm need not converge to a separating hyperplane, even if one exists. Since the LMS solution minimizes the sum of the squares of the distances of the training points to the hyperplane, for this example the plane is rotated clockwise compared to a separating hyperplane.

# The Ho-Kashyap Procedures

- The Perceptron and relaxation procedures find separating vectors if the samples are linearly separable, but do not converge on nonseparable problems.
- If the margin vector  $\mathbf{b}$  is chosen arbitrarily, all we can say is that the MSE procedures minimize  $\|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$ . Now if the training samples happen to be linearly separable, then there exists an  $\hat{\mathbf{a}}$  and a  $\hat{\mathbf{b}}$  such that

$$\mathbf{Y}\hat{\mathbf{a}} = \hat{\mathbf{b}} > \mathbf{0}$$

where by  $\hat{\mathbf{b}} > \mathbf{0}$ , we mean that every component of  $\hat{\mathbf{b}}$  is positive.

We shall now see how the MSE procedure can be modified to obtain both a separating vector  $\mathbf{a}$  and a margin vector  $\mathbf{b}$ .

Both  $\mathbf{a}$  and  $\mathbf{b}$  in the criterion function  $J_s(\mathbf{a}, \mathbf{b}) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$  are allowed to vary (subject to the constraint  $\mathbf{b} > \mathbf{0}$ ), then the minimum value of  $J_s$  is zero, and the  $\mathbf{a}$  that achieves that minimum is a separating vector.

The gradient of  $J_s$  with respect to  $\mathbf{a}$  is given by

$$\nabla_{\mathbf{a}} J_s = 2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b}),$$

and the gradient of  $J_s$  with respect to  $\mathbf{b}$  is given by

$$\nabla_{\mathbf{b}} J_s = -2(\mathbf{Y}\mathbf{a} - \mathbf{b}).$$

For any value of  $\mathbf{b}$ , we can always take  $\mathbf{a} = \mathbf{Y}^\dagger \mathbf{b}$ , thereby obtaining  $\nabla_{\mathbf{a}} J_s = \mathbf{0}$  and minimizing  $J_s$  with respect to  $\mathbf{a}$  in one step.

We must avoid a descent procedure that converges to  $\mathbf{b} = \mathbf{0}$ . We start with  $\mathbf{b} > \mathbf{0}$  and to refuse to reduce any of its components. We can do this and still try to follow the negative gradient if we first set all positive components of  $\nabla_{\mathbf{b}} J_s$  to zero.

Thus, if we let  $|\mathbf{v}|$  denote the vector whose components are the magnitudes of the corresponding components of  $\mathbf{v}$ , we are led to consider an update rule for the margin of the form

$$\mathbf{b}(k + 1) = \mathbf{b}(k) - (\eta/2)[\nabla_{\mathbf{b}} J_s - |\nabla_{\mathbf{b}} J_s|].$$

Ho-Kashyap rule for minimizing  $J_s(\mathbf{a}, \mathbf{b})$ :

$\mathbf{b}(1) > \mathbf{0}$  but otherwise arbitrary

$$\mathbf{b}(k + 1) = \mathbf{b}(k) + 2\eta(k)\mathbf{e}^+(k),$$

Where  $\mathbf{e}(k)$  is the error vector  $\mathbf{e}(k) = \mathbf{Y}\mathbf{a}(k) - \mathbf{b}(k)$ ,

$\mathbf{e}^+(k)$  is the positive part of the error vector

$$\mathbf{e}^+(k) = (1/2)(\mathbf{e}(k) + |\mathbf{e}(k)|) \text{ and } \mathbf{a}(k) = \mathbf{Y}^\dagger \mathbf{b}(k), k = 1, 2, \dots$$



## Algorithm 11 (Ho-Kashyap)

```
1 begin initialize  $\mathbf{a}, \mathbf{b}, \eta(\cdot) < 1$ , criteria  $b_{min}, k_{max}$ 
2       do  $k \leftarrow k + 1$ 
3          $\mathbf{e} \leftarrow \mathbf{Y}\mathbf{a} - \mathbf{b}$ 
4          $\mathbf{e}^+ \leftarrow 1/2(\mathbf{e} + \text{Abs}[\mathbf{e}])$ 
5          $\mathbf{b} \leftarrow \mathbf{b} + 2\eta(k)\mathbf{e}^+$ 
6          $\mathbf{a} \leftarrow \mathbf{Y}^\dagger \mathbf{b}$ 
7         if  $\text{Abs}[\mathbf{e}] \leq b_{min}$  then return  $\mathbf{a}, \mathbf{b}$  and exit
8       until  $k = k_{max}$ 
9       Print NO SOLUTION FOUND
10 end
```

The Ho-Kashyap algorithm provides us with a separating vector in the separable case, and with evidence of nonseparability in the nonseparable case. However, there is no bound on the number of steps needed to disclose nonseparability.

# Some Related Procedures

If we write  $\mathbf{Y}^\dagger = (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t$  and make use of the fact that  $\mathbf{Y}^t\mathbf{e}(k) = 0$ , we can modify the Ho-Kashyap rule as follows

$$\left. \begin{aligned} \mathbf{b}(1) &> 0 \quad \text{but otherwise arbitrary} \\ \mathbf{a}(1) &= \mathbf{Y}^\dagger\mathbf{b}(1) \\ \mathbf{b}(k+1) &= \mathbf{b}(k) + \eta(\mathbf{e}(k) + |\mathbf{e}(k)|) \\ \mathbf{a}(k+1) &= \mathbf{a}(k) + \eta\mathbf{Y}^\dagger|\mathbf{e}(k)|, \end{aligned} \right\}$$

$$\mathbf{e}(k) = \mathbf{Y}\mathbf{a}(k) - \mathbf{b}(k).$$

This then gives the algorithm for fixed learning rate:

## Algorithm 12 (Modified Ho-Kashyap)

```
1 begin initialize  $\mathbf{a}, \mathbf{b}, \eta < 1$ , criterion  $b_{min}, k_{max}$ 
2       do  $k \leftarrow k + 1$ 
3          $\mathbf{e} \leftarrow \mathbf{Y}\mathbf{a} - \mathbf{b}$ 
4          $\mathbf{e}^+ \leftarrow 1/2(\mathbf{e} + \text{Abs}[\mathbf{e}])$ 
5          $\mathbf{b} \leftarrow \mathbf{b} + 2\eta(k)(\mathbf{e} + \text{Abs}[\mathbf{e}])$ 
6          $\mathbf{a} \leftarrow \mathbf{Y}^\dagger \mathbf{b}$ 
7         if  $\text{Abs}[\mathbf{e}] \leq b_{min}$  then return  $\mathbf{a}, \mathbf{b}$  and exit
8       until  $k = k_{max}$ 
9       print NO SOLUTION FOUND
10 end
```

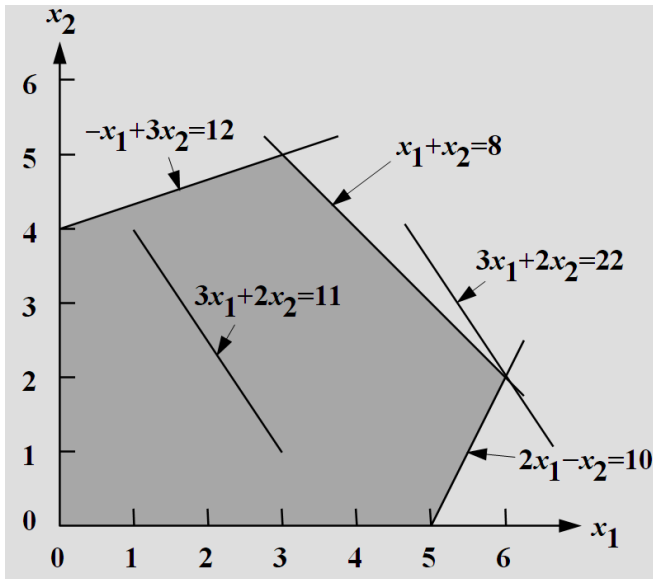
This algorithm differs from the Perceptron and relaxation algorithms for solving linear inequalities in at least three ways:

- (1) it varies both the weight vector  $\mathbf{a}$  and the margin vector  $\mathbf{b}$ ,
- (2) it provides evidence of nonseparability, but
- (3) it requires the computation of the pseudoinverse of  $\mathbf{Y}$ .

# Linear Programming Algorithms\*

- The Perceptron, relaxation and Ho-Kashyap procedures are basically gradient descent procedures for solving simultaneous linear inequalities.
- Linear programming techniques are procedures for maximizing or minimizing linear functions subject to linear equality or inequality constraints.
- Find a vector  $\mathbf{u} = (u_1, \dots, u_m)^t$  that minimizes the linear (scalar) objective function  $z = \boldsymbol{\alpha}^t \mathbf{u}$  subject to the constraint  $\mathbf{A}\mathbf{u} \geq \boldsymbol{\beta}$ , where  $\boldsymbol{\alpha}$  is an  $m$ -by-1 cost vector,  $\boldsymbol{\beta}$  is an  $l$ -by-1 vector, and  $\mathbf{A}$  is an  $l$ -by- $m$  matrix.

- The simplex algorithm is the classical iterative procedure for solving this problem. For technical reasons, it requires the imposition of one more constraint, viz.,  $\mathbf{u} \geq 0$ .



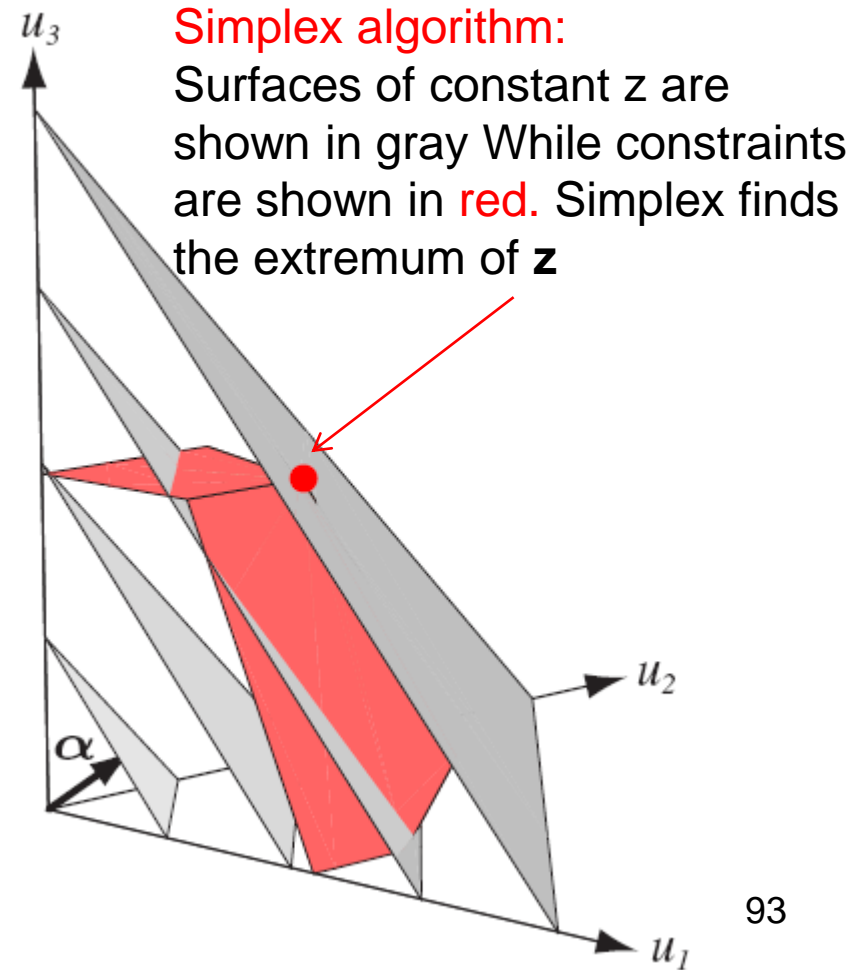
$$\text{maximize } 3x_1 + 2x_2$$

$$\text{subject to } -x_1 + 3x_2 \leq 12$$

$$x_1 + x_2 \leq 8$$

$$2x_1 - x_2 \leq 10$$

$$x_1, x_2 \geq 0.$$



- If we think of  $\mathbf{u}$  as being the weight vector  $\mathbf{a}$ , this constraint is unacceptable.
- We write  $\mathbf{a} \equiv \mathbf{a}^+ - \mathbf{a}^-$  where
 
$$\mathbf{a}^+ \equiv 0.5(|\mathbf{a}| + \mathbf{a}), \quad \mathbf{a}^- \equiv 0.5(|\mathbf{a}| - \mathbf{a})$$
- Suppose that we have a set of  $n$  samples  $\mathbf{y}_1, \dots, \mathbf{y}_n$  and we want a weight vector  $\mathbf{a}$  that satisfies  $\mathbf{a}^t \mathbf{y}_i \geq b_i > 0$  for all  $i$ .
- One approach is to introduce what is called an artificial variable  $\tau \geq 0$  by writing  $\mathbf{a}^t \mathbf{y}_i \geq \tau > b_i$ .
- **Minimize  $\tau$  over all values of  $\tau$  and  $\mathbf{a}$  that satisfy the conditions  $\mathbf{a}^t \mathbf{y}_i \geq b_i$  and  $\tau \geq 0$ .**

- If the answer is zero, the samples are linearly separable, and we have a solution. If the answer is positive, there is no separating vector, but we have proof that the samples are non-separable.
- Formally, our problem is to find a vector  $\mathbf{u}$  that minimizes the objective function  $z = \boldsymbol{\alpha}^t \mathbf{u}$  subject to the constraints  $\mathbf{A}\mathbf{u} \geq \boldsymbol{\beta}$  and  $\mathbf{u} \geq 0$ , where

$$\mathbf{A} = \begin{bmatrix} \mathbf{y}_1^t & -\mathbf{y}_1^t & 1 \\ \mathbf{y}_2^t & -\mathbf{y}_2^t & 1 \\ \vdots & \vdots & \vdots \\ \mathbf{y}_n^t & -\mathbf{y}_n^t & 1 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \mathbf{a}^+ \\ \mathbf{a}^- \\ \tau \end{bmatrix}, \quad \boldsymbol{\alpha} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

# Support Vector Machines

- We have seen how to train linear machines with margins. *Support Vector Machines* (SVMs) are motivated by many of the same considerations, but rely on preprocessing the data to represent patterns in a high dimension — typically much higher than the original feature space.
- With an appropriate nonlinear mapping  $\Phi()$  to a sufficiently high dimension, data from two categories can always be separated by a hyperplane.



- Here we assume each pattern  $\mathbf{x}_k$  has been transformed to  $\mathbf{y}_k = \Phi(\mathbf{x}_k)$ ; we return to the choice of  $\Phi()$  below.
- For each of the  $n$  patterns,  $k = 1, 2, \dots, n$ , we let  $z_k = \pm 1$ , according to whether pattern  $k$  is in  $\omega_1$  or  $\omega_2$ . A linear discriminant in an augmented  $\mathbf{y}$  space is

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

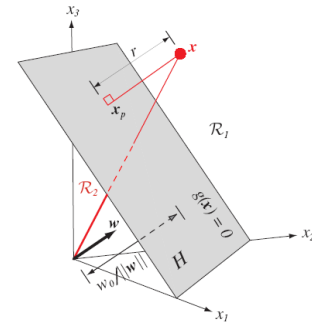
where both the weight vector and the transformed pattern vector are augmented (by  $a_0 = w_0$  and  $y_0 = 1$ , respectively). Thus a separating hyperplane insures

$$z_k \mathbf{g}(\mathbf{y}_k) \geq 1, \quad k = 1, \dots, n \quad (105)$$

- The goal in training a Support Vector Machine is to find the separating hyperplane with the largest margin;

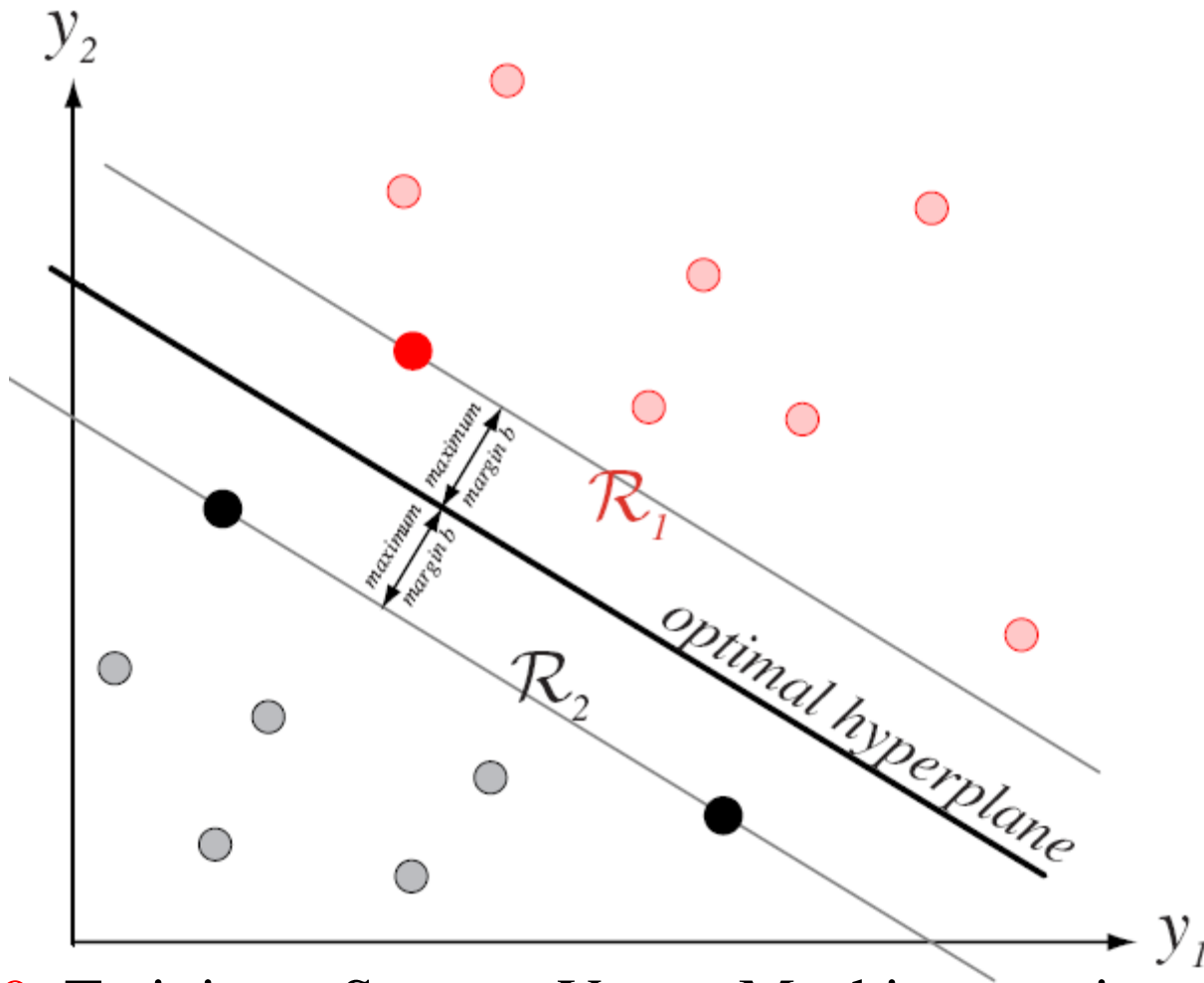
As illustrated in Fig. 5.2 the distance from any hyperplane to a (transformed) pattern  $\mathbf{y}$  is  $|g(\mathbf{y})|/\|\mathbf{a}\|$ , and assuming that a positive margin  $\mathbf{b}$  exists, Eq. 105 implies

$$\frac{z_k g(\mathbf{y}_k)}{\|\mathbf{a}\|} \geq b \quad k = 1, \dots, n;$$



the goal is to find the weight vector  $\mathbf{a}$  that maximizes  $\mathbf{b}$ .

The **support vectors** are the (transformed) training patterns for which Eq. 105 represents an equality — that is, the **support vectors are (equally) close** to the hyperplane.



**Figure 5.19:** Training a Support Vector Machine consists of finding the optimal hyperplane, i.e., the one with the maximum distance from the nearest training patterns. The support vectors are those (nearest) patterns, a distance  $b$  from the hyperplane. The three support vectors are shown in solid dots.

The support vectors are the training samples that define the optimal separating hyperplane and are the most difficult patterns to classify. Informally speaking, they are the patterns most informative for the classification task.

## **SVM training**

The first step is, of course, to choose the nonlinear  $\Phi$  - functions that map the input to a higher dimensional space. Often this choice will be informed by the designer's knowledge of the problem domain. In the absence of such information, one might choose to use polynomials, Gaussians or yet other basis functions.

We begin by recasting the problem of minimizing the magnitude of the weight vector constrained by the separation into an unconstrained problem by the method of Lagrange undetermined multipliers.

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

and seek to minimize  $L(\cdot)$  with respect to the weight vector  $\mathbf{a}$ , and maximize it with respect to the undetermined multipliers  $\alpha_k \geq 0$ .

It can be shown using the so-called Kuhn- Tucker construction (also associated with Karush 1939) that this optimization can be reformulated as **maximizing**:

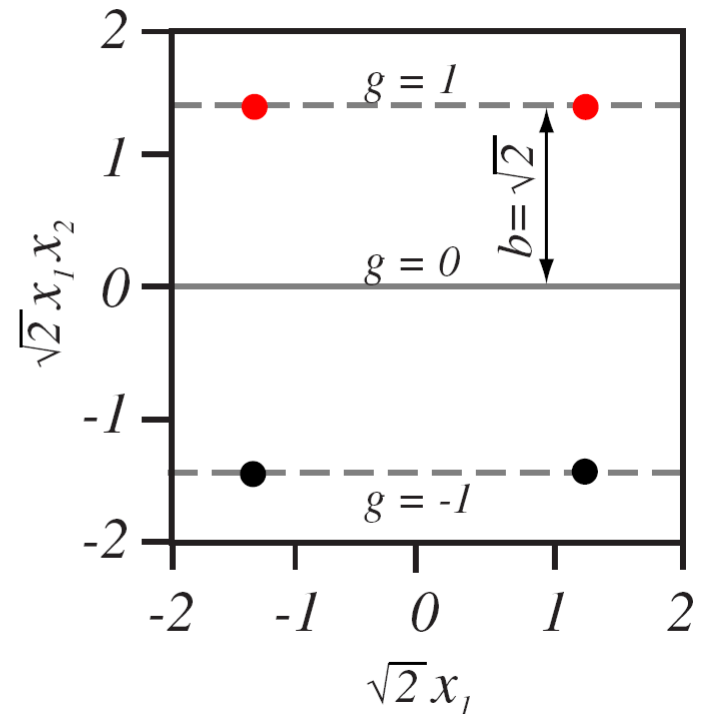
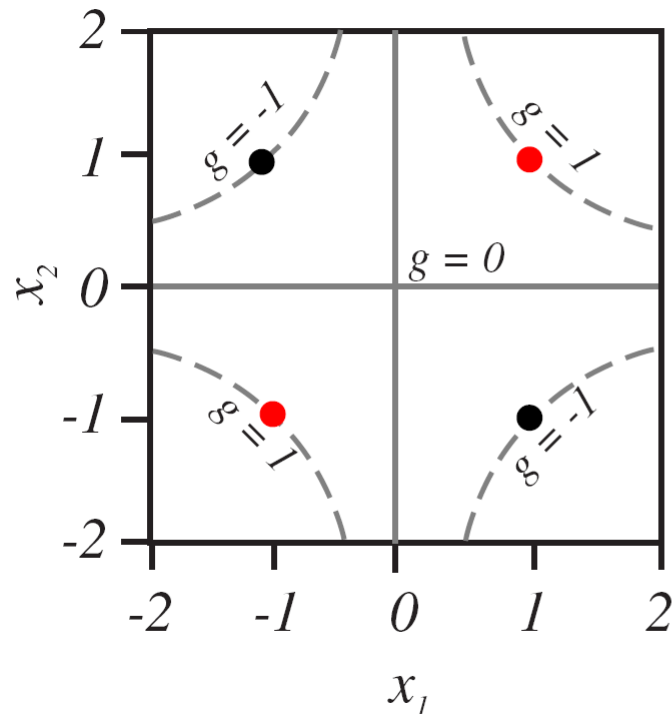
$$L(\boldsymbol{\alpha}) = \sum_{k=1}^n \alpha_k - \frac{1}{2} \sum_{k,j}^n \alpha_k \alpha_j z_k z_j \mathbf{y}_j^t \mathbf{y}_k,$$

subject to the constraints

$$\sum_{k=1}^n z_k \alpha_k = 0 \quad \alpha_k \geq 0, k = 1, \dots, n,$$

An important benefit of the Support Vector Machine approach is that the complexity of the resulting classifier is characterized by the number of support vectors — independent of the dimensionality of the transformed space. Thus SVMs tend to be less prone to problems of over-fitting than some other methods.

## Example 2: SVM for the XOR problem



While many  $\Phi$ -functions could be used, here we use the simplest expansion up to second order:  $1$ ,  $\sqrt{2}x_1$ ,  $\sqrt{2}x_2$ ,  $\sqrt{2}x_1x_2$ ,  $x_1^2$  and  $x_2^2$ , where the  $\sqrt{2}$  is convenient for normalization.

We seek to maximize Eq.

$$\sum_{k=1}^4 \alpha_k - \frac{1}{2} \sum_{k,j}^n \alpha_k \alpha_j z_k z_j \mathbf{y}_j^t \mathbf{y}_k$$

subject to the constraints

$$\begin{aligned} \alpha_1 - \alpha_2 + \alpha_3 - \alpha_4 &= 0 \\ 0 \leq \alpha_k & \quad k = 1, 2, 3, 4. \end{aligned}$$

While we could use iterative gradient descent, we can use analytic techniques instead. The solution is  $\alpha_k = 1/8$ , for  $k = 1, 2, 3, 4$ , and from the last term in Eq. 108 this implies that all four training patterns are support vectors — an unusual case due to the highly symmetric nature of the XOR problem.



The final discriminant function is

$$g(\mathbf{x}) = g(x_1, x_2) = x_1 x_2,$$

and the decision hyperplane is defined by  $g = 0$ , which properly classifies all training patterns.

The margin is easily computed from the solution  $\|a\|$  and is found to be  $b = 1/\|a\| = \sqrt{2}$ .

# Linear SVM: the separable case

- Linear discriminant

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$

Decide  $\omega_1$  if  $g(\mathbf{x}) > 0$  and  $\omega_2$  if  $g(\mathbf{x}) < 0$

- Class labels

$$z_k = \begin{cases} +1 & \text{if } \mathbf{x}_k \in \omega_1 \\ -1 & \text{if } \mathbf{x}_k \in \omega_2 \end{cases}$$

- Normalized version

$$z_k g(\mathbf{x}_k) > 0 \quad \text{or} \quad z_k (\mathbf{w}^t \mathbf{x}_k + w_0) > 0, \quad \text{for } k = 1, 2, \dots, n$$

# Linear SVM: the separable case

- The distance of a point  $\mathbf{x}_k$  from the separating hyperplane should satisfy the constraint:

$$\frac{z_k g(\mathbf{x}_k)}{\|\mathbf{w}\|} \geq b, \quad b > 0$$

- To ensure uniqueness, impose:

$$b\|\mathbf{w}\|=1$$

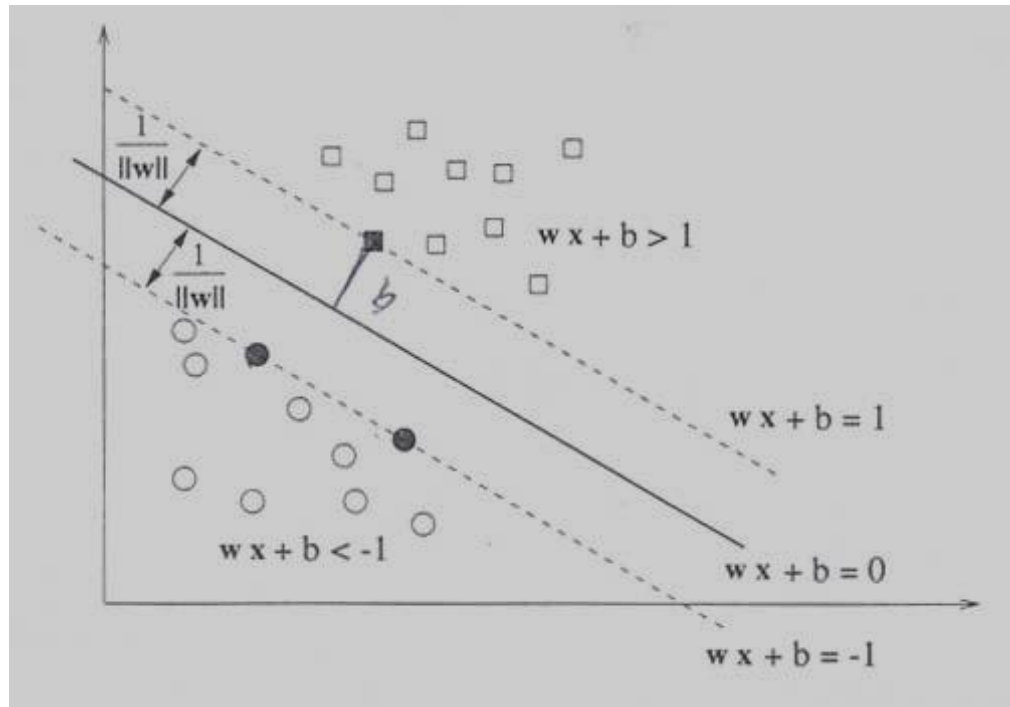
- The above constraint becomes:

$$z_k g(\mathbf{x}_k) \geq 1 \quad \text{where } b = \frac{1}{\|\mathbf{w}\|}$$

# Linear SVM: the separable case

**Maximize Margin** :  $2b = \frac{2}{\|\mathbf{w}\|} \Rightarrow$  **Minimize**  $\frac{1}{2} \|\mathbf{w}\|$

Subject to  $z_k (\mathbf{w}^t \mathbf{x}_k + w_0) > 1$ , for  $k = 1, 2, \dots, n$



Quadratic  
Programming  
Problem !

# Linear SVM: the separable case

- Use Lagrange optimization:

$$L(\mathbf{w}, w_0, \lambda) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{k=1}^n \lambda_k [z_k (\mathbf{w}^t \mathbf{x}_k + w_0) - 1], \quad \lambda_k \geq 0$$

- Easier to solve the “dual” problem:

**Problem 2:** Maximize  $\sum_{k=1}^n \lambda_k - \frac{1}{2} \sum_{k,j} \lambda_k \lambda_j z_k z_j \mathbf{x}_j^t \mathbf{x}_k$

subject to  $\sum_{k=1}^n z_k \lambda_k = 0, \quad \lambda_k \geq 0, \quad k = 1, 2, \dots, n$

# Linear SVM: the separable case

- The solution is given by:

The Karush-Kuhn-Tucker (KKT) conditions

$$\mathbf{w} = \sum_{k=1}^n z_k \lambda_k \mathbf{x}_k$$

Only support vectors  
contribute to the solution!!

$$w_0 = z_k - \mathbf{w}^t \mathbf{x}_k$$

$$g(\mathbf{x}) = \sum_{k=1}^n z_k \lambda_k (\mathbf{x}^t \cdot \mathbf{x}_k) + w_0 = \sum_{k=1}^n z_k \lambda_k (\mathbf{x} \cdot \mathbf{x}_k) + w_0$$

- It can be shown that if  $\mathbf{x}_k$  is a **not** support vector, then  $\lambda_k=0$ .

# Linear SVM: the non-separable case

- Allow misclassifications (i.e., soft margin classifier) by introducing error variables  $\psi_k$  :

$$z_k (\mathbf{w}^t \mathbf{x}_k + w_0) \geq 1 - \psi_k, \quad k = 1, 2, \dots, n$$

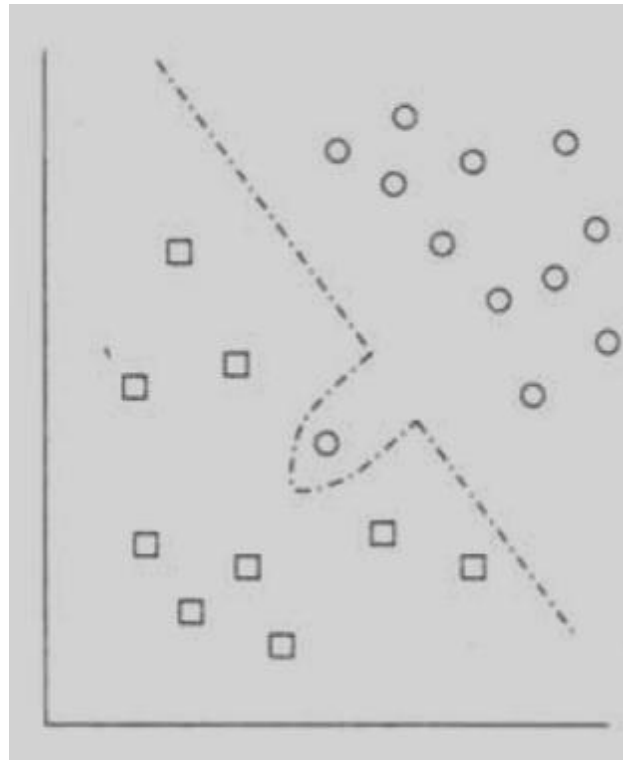
**Problem 3:** Minimize  $\frac{1}{2} \|\mathbf{w}\|^2 + c \sum_{k=1}^n \psi_k$

subject to  $z_k (\mathbf{w}^t \mathbf{x}_k + w_0) \geq 1 - \psi_k, \quad k = 1, 2, \dots, n$

- The result is a hyperplane that minimizes the sum of errors  $\psi_k$  while maximizing the margin for the correctly classified data.

# Linear SVM: the non-separable case

- The constant  $c$  controls the tradeoff between margin and misclassification errors (aims to prevent outliers from affecting the optimal hyperplane).





# Linear SVM: the non-separable case

- We can reformulate "Problem 3" as maximizing the following problem (*dual problem*):

$$\mathbf{Problem\ 4:}\ \text{Maximize}\ \sum_{k=1}^n \lambda_k - \frac{1}{2} \sum_{k,j} \lambda_k \lambda_j z_k z_j \mathbf{x}_j^t \mathbf{x}_k$$

$$\text{subject to}\ \sum_{k=1}^n z_k \lambda_k = 0 \text{ and } 0 \leq \lambda_k \leq c, k = 1, 2, \dots, n$$

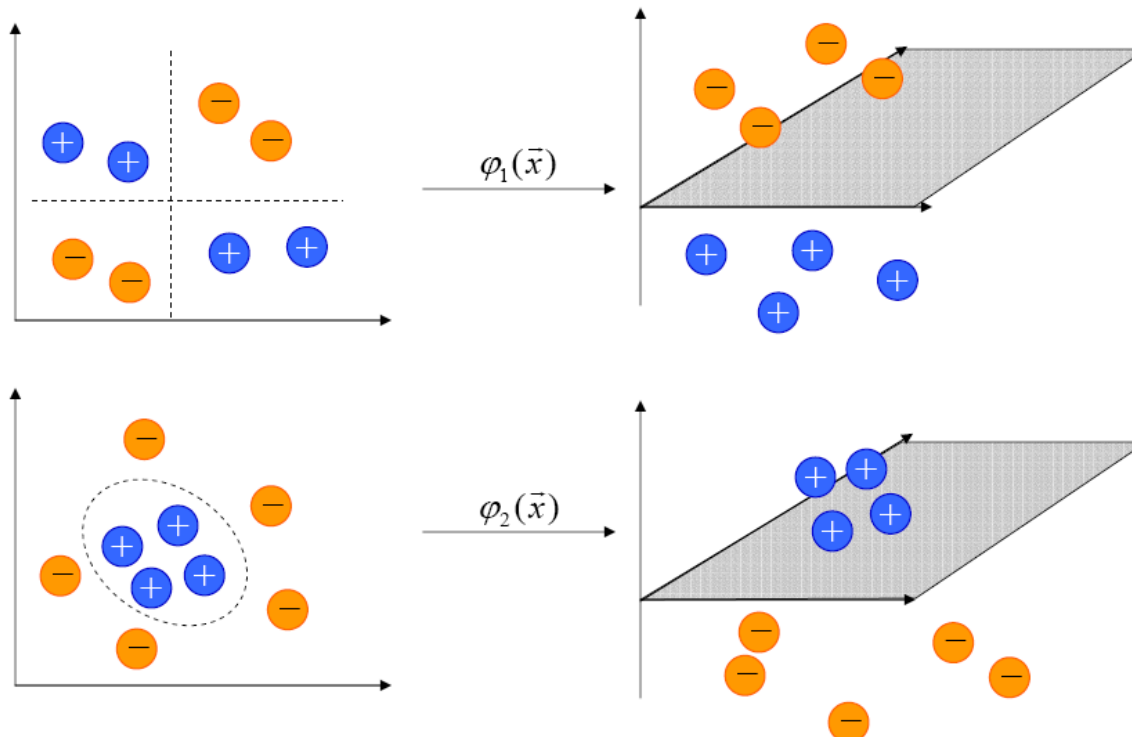
where the use of error variables  $\psi_k$  constraint the range of the Lagrange coefficients from 0 to  $c$ .

# Nonlinear SVM

Extending the above concepts to the non-linear case relies on preprocessing the data to represent them in a much higher dimensionality space.

$$\mathbf{x}_k \rightarrow \Phi(\mathbf{x}_k)$$

Using an appropriate nonlinear mapping  $\Phi(\cdot)$  to a sufficiently high dimensional space, data from two classes can always be separated by a hyperplane.



Linearly Separable in Higher Dimension

# Nonlinear SVM (cont'd)

- The decision function for the optimal hyperplane is given by

$$g(\mathbf{x}) = \sum_{k=1}^n z_k \lambda_k (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_k)) + w_0$$

- The decision rule is the same as before:

*decide  $\omega_1$  if  $g(x) > 0$  and  $\omega_2$  if  $g(x) < 0$*

- The disadvantage of this approach is that the mapping  $x_k \rightarrow \Phi(x_k)$  might be very computationally intensive to compute.
-

# The kernel trick

- Compute dot products using a kernel function

$$K(\mathbf{x}, \mathbf{x}_k) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_k)$$

- Advantages of using a kernel
  - No need to know  $\Phi()$  !!
  - The discriminant is given by:

$$g(\mathbf{x}) = \sum_{k=1}^n z_k \lambda_k K(\mathbf{x}, \mathbf{x}_k) + w_0$$

# The kernel trick (cont'd)

*polynomial kernel:*  $K(x,y)=(x \cdot y)^d$

- The kernel trick implies that the computation remains feasible even if the feature space has very high dimensionality.

\* It can be shown for the case of polynomial kernels that the data is mapped to a space of dimension  $h = \binom{p+d-1}{d}$  where  $p$  is the original dimensionality.

\* Suppose  $p=256$  and  $d = 4$ , then  $h=183,181,376$  !!

\* A dot product in the high dimensional space would require  $O(h)$  computations while the kernel requires only  $O(p)$  computations.

# Choice of kernel is not unique!

*Example:* consider  $x \in R^2$ ,  $\Phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \in R^3$ , and  $K(x, y) = (x \cdot y)^2$

$$(x \cdot y)^2 = (x_1y_1 + x_2y_2)^2$$

$$\Phi(x) \cdot \Phi(y) = x_1^2y_1^2 + 2x_1y_1x_2y_2 + x_2^2y_2^2 = (x_1y_1 + x_2y_2)^2$$

- Note that neither the mapping  $\Phi()$  nor the high dimensional space are unique.

$$\Phi(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} (x_1^2 - x_2^2) \\ 2x_1x_2 \\ (x_1^2 + x_2^2) \end{pmatrix} \in R^3 \quad \text{or} \quad \Phi(x) = \begin{pmatrix} x_1^2 \\ x_1x_2 \\ x_1x_2 \\ x_2^2 \end{pmatrix} \in R^4$$

# Suitable kernel functions

- Kernel functions which can be expressed as a dot product in some space satisfy the *Mercer's* condition (see Burges' paper).
- The *Mercer's* condition does not tell us how to construct  $\Phi()$  or even what the high dimensional space is.
- By using different kernel functions, SVM implement a variety of learning machines, some of which coincide with classical architectures (see below).

$$\textit{polynomial: } K(x, x_k) = (x \cdot x_k)^d$$

$$\textit{sigmoidal: } K(x, x_k) = \tanh(v_k(x \cdot x_k) + c_k)$$

(corresponds to a two-layer sigmoidal neural network)

$$\textit{Gaussian: } K(x, x_k) = \exp\left(\frac{-\|x - x_k\|^2}{2\sigma_k^2}\right)$$

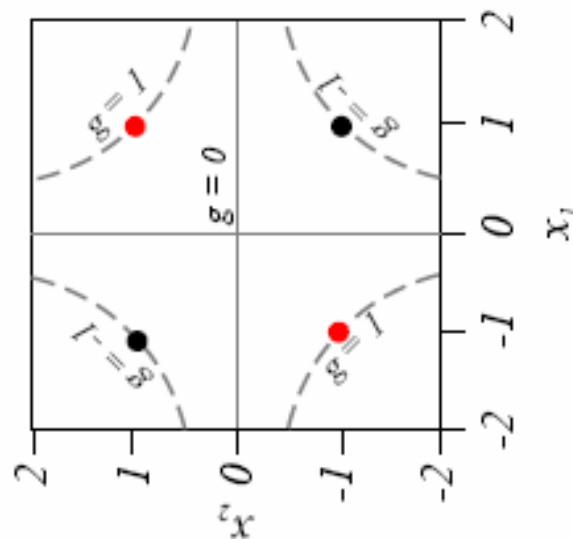
(corresponds to a radial basis function (RBF) neural network)

# Example

- Consider the XOR problem which is non-linearly separable:

(1,1) and (-1,-1) belong to  $\omega_1$

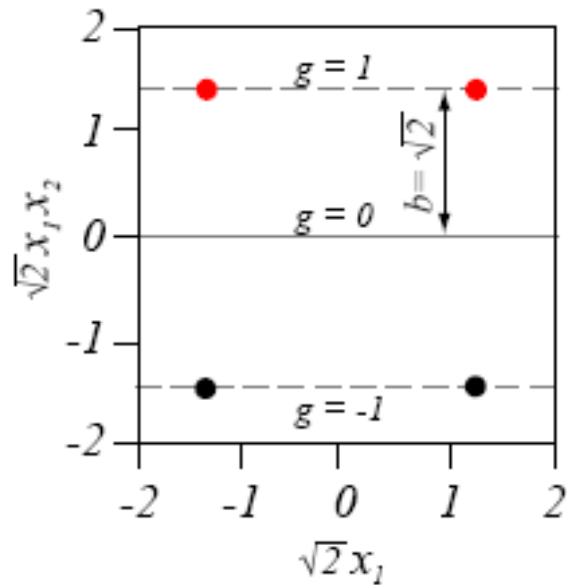
(1,-1) and (-1,1) belong to  $\omega_2$





# Example (cont'd)

- Consider the following mapping (many other mappings could be used too):



$$y = \Phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_2 \\ x_2^2 \\ 1 \end{pmatrix}$$

# Example (cont'd)

- The above transformation maps  $x_k$  to a 6-dimensional space:

$$\begin{aligned} y_1 = \Phi(x_1) &= \begin{pmatrix} 1 \\ \sqrt{2} \\ \sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \end{pmatrix} & y_3 = \Phi(x_3) &= \begin{pmatrix} 1 \\ -\sqrt{2} \\ \sqrt{2} \\ -\sqrt{2} \\ 1 \\ 1 \end{pmatrix} \\ y_2 = \Phi(x_2) &= \begin{pmatrix} 1 \\ \sqrt{2} \\ -\sqrt{2} \\ -\sqrt{2} \\ 1 \\ 1 \end{pmatrix} & y_4 = \Phi(x_4) &= \begin{pmatrix} 1 \\ -\sqrt{2} \\ -\sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

# Example (cont'd)

- We seek to maximize:

$$\sum_{k=1}^4 \lambda_k - \frac{1}{2} \sum_{k,j} \lambda_k \lambda_j z_k z_j \Phi(x_j^t) \Phi(x_k)$$

$$\text{subject to } \sum_{k=1}^4 z_k \lambda_k = 0, \lambda_k \geq 0, k = 1, 2, \dots, 4$$

- The solution turns out to be:

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \frac{1}{8}$$

- Since all  $\lambda_k \neq 0$ , all  $x_k$  are support vectors !

# Example (cont'd)

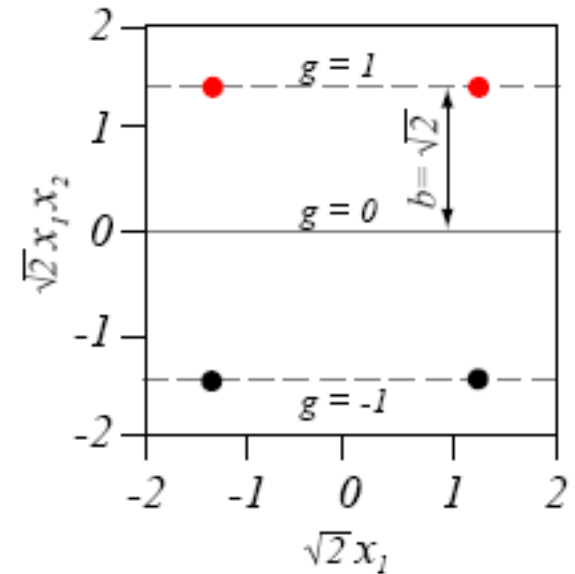
- We can now compute  $w$ :

$$w = \sum_{k=1}^4 z_k \lambda_k \Phi(x_k) = \frac{1}{8} \begin{pmatrix} 1 \\ \sqrt{2} \\ \sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \end{pmatrix} - \frac{1}{8} \begin{pmatrix} 1 \\ \sqrt{2} \\ -\sqrt{2} \\ -\sqrt{2} \\ 1 \\ 1 \end{pmatrix} + \frac{1}{8} \begin{pmatrix} 1 \\ -\sqrt{2} \\ \sqrt{2} \\ -\sqrt{2} \\ 1 \\ 1 \end{pmatrix} - \frac{1}{8} \begin{pmatrix} 1 \\ -\sqrt{2} \\ -\sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 \\ 0 \\ \sqrt{2} \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- The solution for  $w_0$  can be determined using any support vector, e.g.,  $x_1$ :

$$w^t \Phi(x_1) + w_0 = z_1 \quad \text{or} \quad \underline{w_0 = z_1 - w^t x_1 = 0}$$

# Example (cont'd)



- The margin  $b$  is computed as follows:

$$b = \frac{1}{\|w\|} = \sqrt{2}$$

- The decision function is the following:

$$g(x) = w^t \Phi(x) + w_0 = x_1 x_2$$

where we decide  $\omega_1$  if  $g(x) > 0$  and  $\omega_2$  if  $g(x) < 0$

# Comments on SVMs

- Global optimization method, no local optima (i.e., based on exact optimization, not approximate methods).
- The performance of SVMs depends on the choice of the kernel and its parameters.
  - The best choice of kernel for a given problem is still a research problem.
- Its complexity depends on the number of support vectors, **not** on the dimensionality of the transformed space.
- Appear to avoid overfitting in high dimensional spaces and generalize well using a small training set.
- The optimal design of multi-class SVM classifiers is a research topic.