Linear Discriminant Functions

Empirical Risk Minimization

- Every classifier / regressor does what is called as - `empirical risk minimization'

- Learning pertains to coming up with an architecture that can minimize a risk / loss function defined on the training /empirical data.

Classifier taxonomy

- Generative classifiers
- Discriminative classifiers

- Types of generative classifier

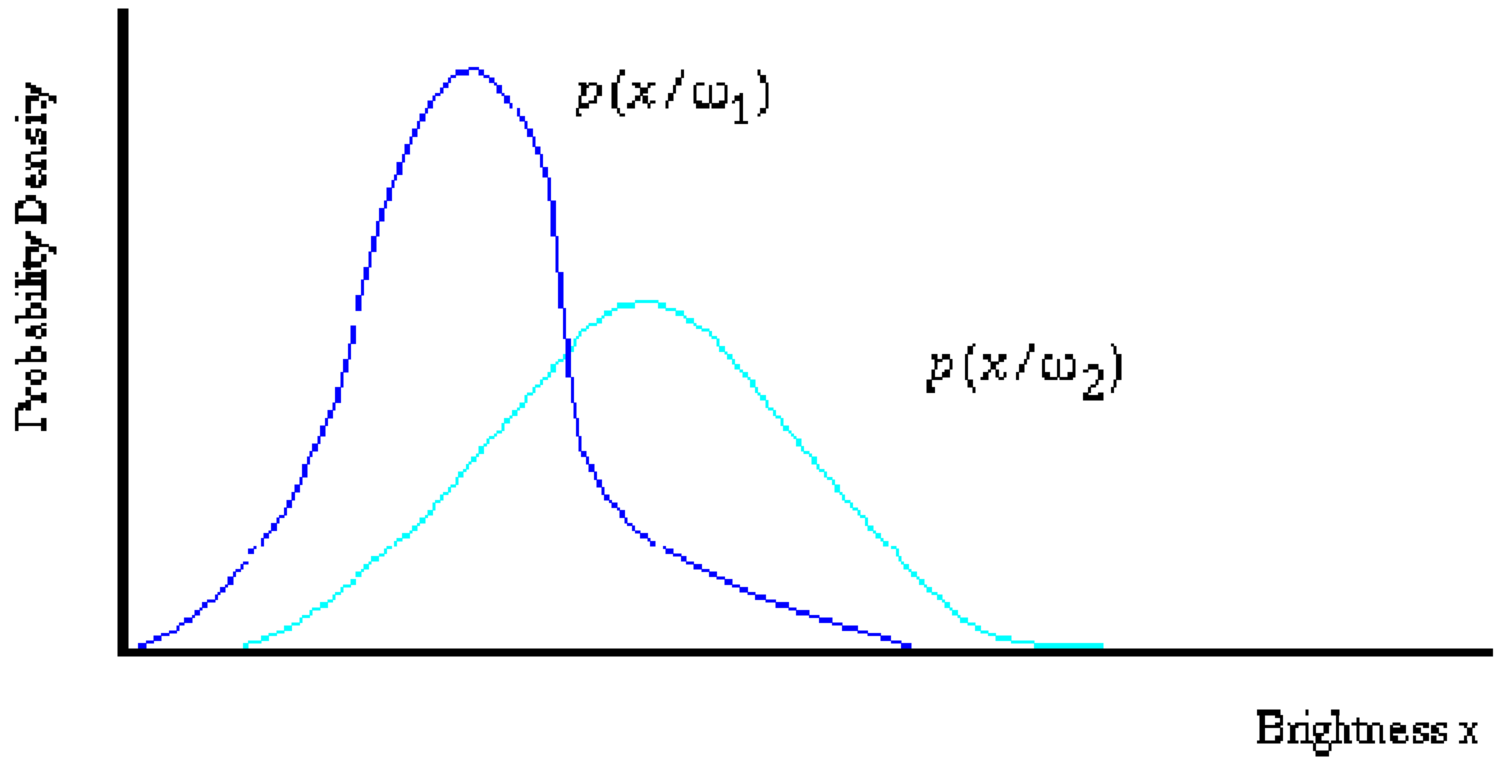
[a] Parametric

[b] Non-parametric

Generative classifier

- Samples of training data of a class assumed to come from a probability density function (class conditional pdf)
- If the form of pdf is assumed , such as uniform, gaussian, rayleigh, etc …one can estimate the parameters of the distribution.
- → Parametric classifier

Class Conditional Density



Generative classifier

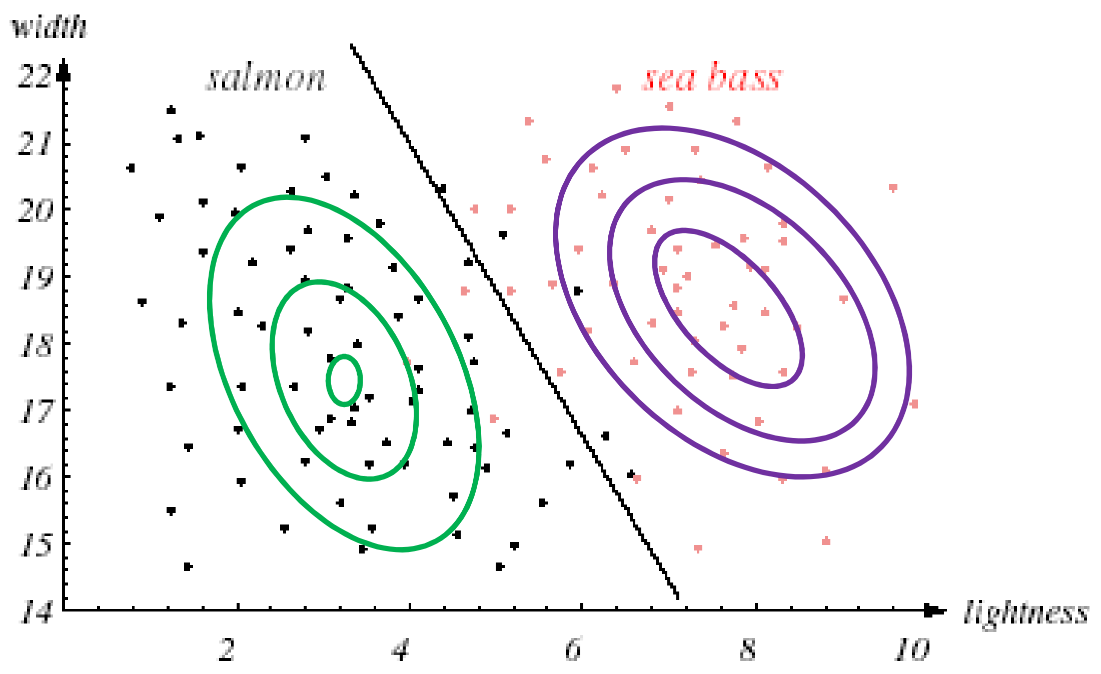


FIGURE 1.4. The two features of lightness and width for sea bass and salmon. The dark line could serve as a decision boundary of our classifier. Overall classification error on the data shown is lower than if we use only one feature as in Fig. 1.3, but there will still be some errors. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

- One can as well assume to use the training data to build a pdf -→ Non parametric approach

- Discriminative classifier → No such assumption of data being drawn from an underlying pdf. Models the decision boundary by adaptive gradient descent techniques.

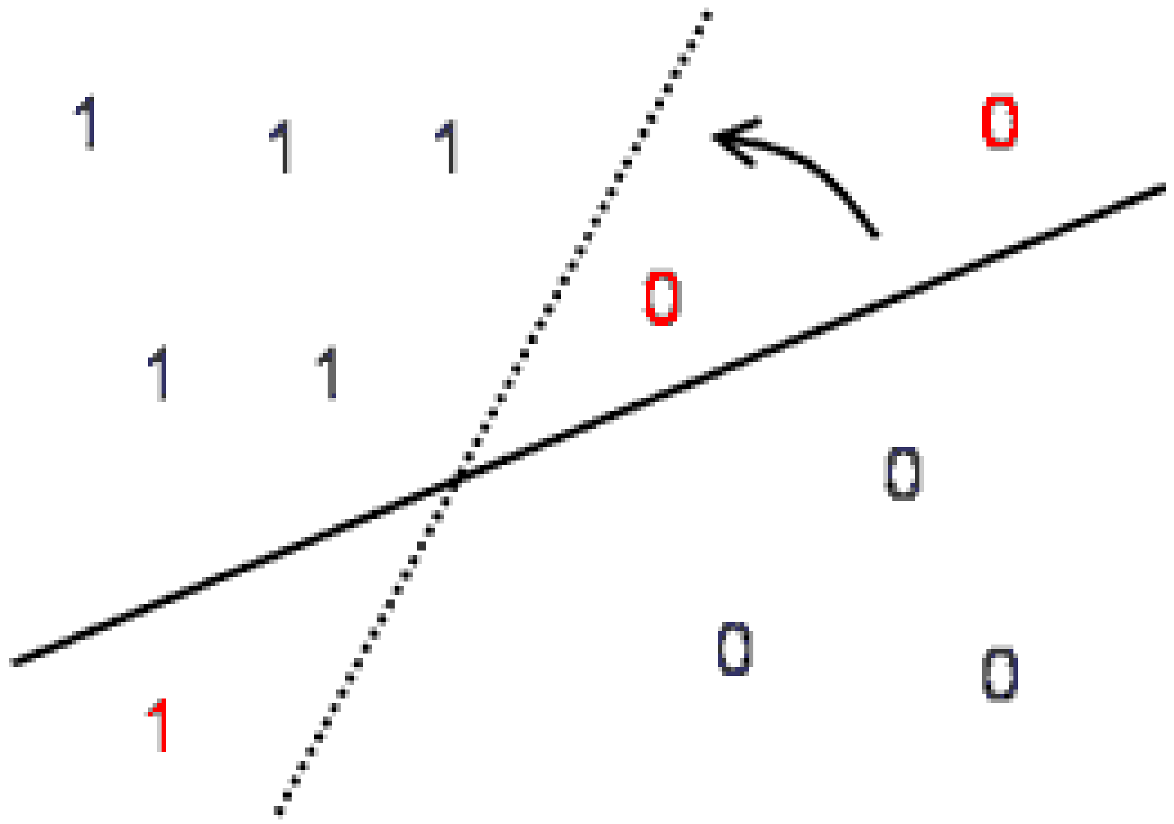
Discriminative Classifier

- Start with initial weights that define the decision surface
- Update the weights based on some optimization criterion….

- No need to model the distribution of samples of a given class…..class conditional density concept not required!

- Neural nets (such as MLP, Single layer perceptron, SVMs) fall in the category of discriminative classifiers.

Discriminative classifier



Linear Discriminant Functions

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

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

where $\mathbf{w}$ is the *weight vector* and w_0 the *bias* or *threshold weight*. A two-category linear classifier implements the following decision rule: Decide ω_1 if $g(\mathbf{x}) > 0$ and ω_2 if $g(\mathbf{x}) < 0$. Thus, $\mathbf{x}$ is assigned to ω_1 if the inner product $\mathbf{w}^t\mathbf{x}$ exceeds the threshold $-w_0$ and ω_2 otherwise. If $g(\mathbf{x}) = 0$, $\mathbf{x}$ can ordinarily be assigned to either class, but

Linear Discriminant Functions

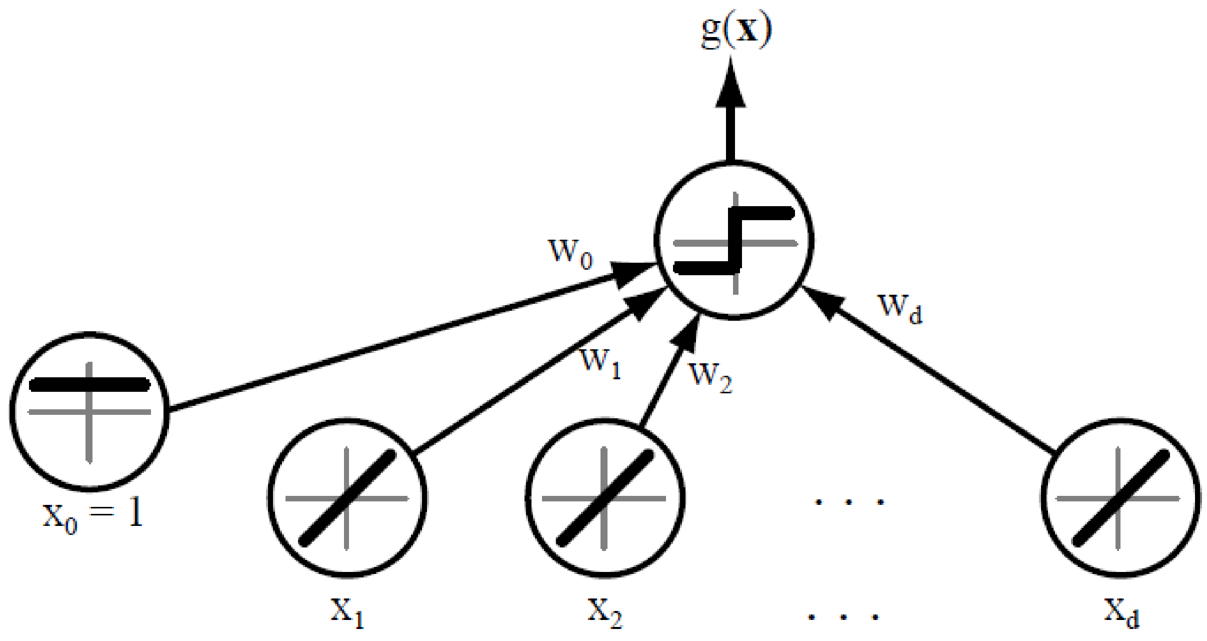


Figure 5.1: A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the output unit sums all these products and emits a $+1$ if $\mathbf{w}^t\mathbf{x} + w_0 > 0$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

Linear Discriminant Functions

The equation $g(\mathbf{x}) = 0$ defines the decision surface that separates points assigned to ω_1 from points assigned to ω_2. When g(**x**) is linear, this decision surface is a *hyperplane*. If $\mathbf{x}_1$ and $\mathbf{x}_2$ are both on the decision surface, then

$$\mathbf{w}^t\mathbf{x}_1 + w_0 = \mathbf{w}^t\mathbf{x}_2 + w_0$$

or

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

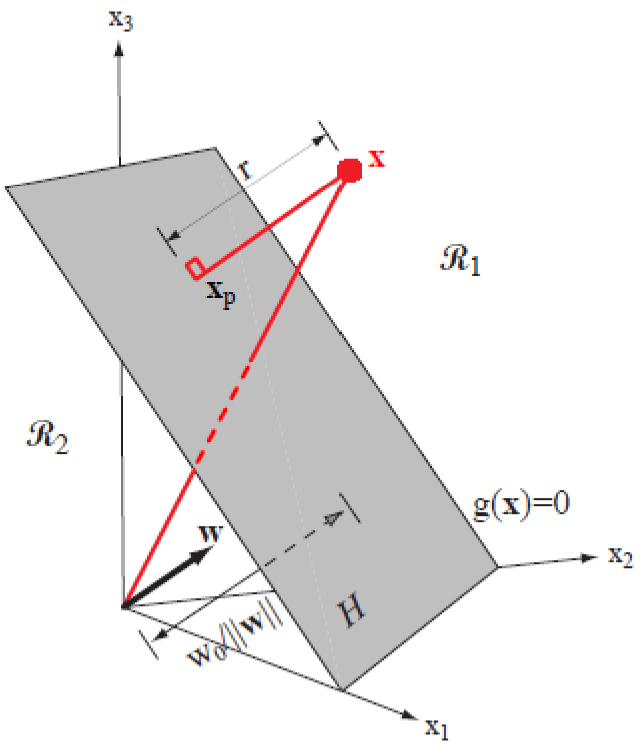
Linear Discriminant Functions

The discriminant function g(**x**) gives an algebraic measure of the distance from **x** to the hyperplane.

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|},$$

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

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}.$$



Linear Discriminant Functions

c linear discriminant functions

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

and assigning $\mathbf{x}$ to ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$; in case of ties, the classification is left undefined. The resulting classifier is called 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 region $\mathcal{R}_i$. If $\mathcal{R}_i$ and $\mathcal{R}_j$ are contiguous, the boundary between them is a portion of the hyperplane H_{ij} defined by

$$g_i(\mathbf{x}) = g_j(\mathbf{x})$$

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

Linear Discriminant Functions

The linear discriminant function $g(\mathbf{x})$ can be written as

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

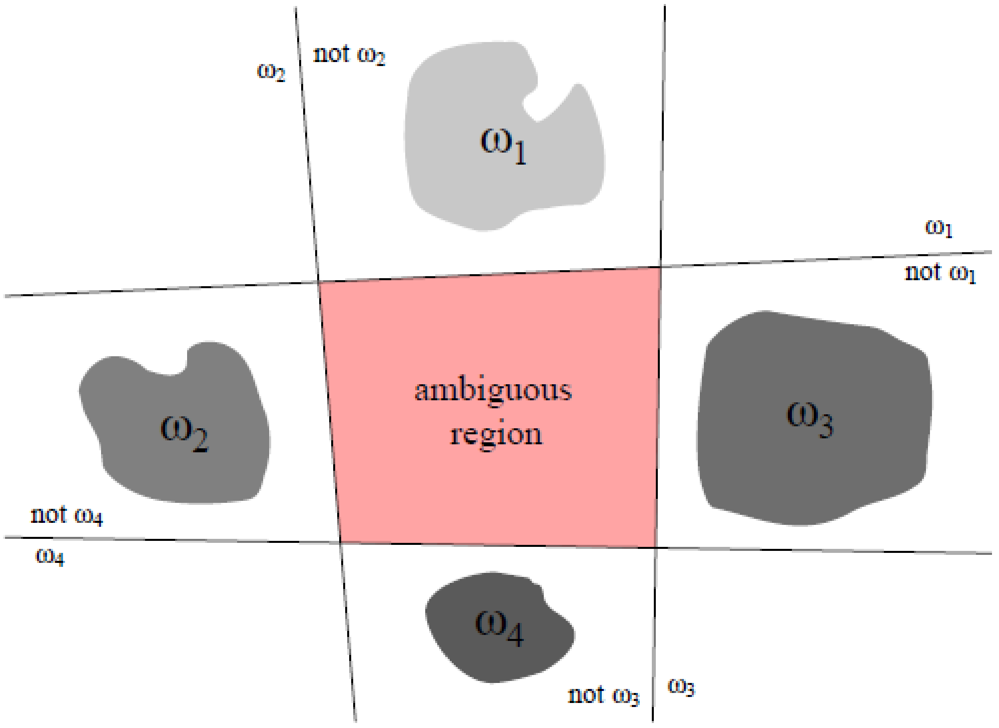
quadratic discriminant function

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j.$$

One versus all classification

- The OVA method, employs *c* discriminant functions for a *c*-class problem.
- The *i**th* discriminant function generates a decision boundary between class *i* and the other *c – 1* classes.
- The test sample is assigned to the class having the largest value of the decision function amongst all the *c* discriminant functions.

One versus all classification



One versus one classification

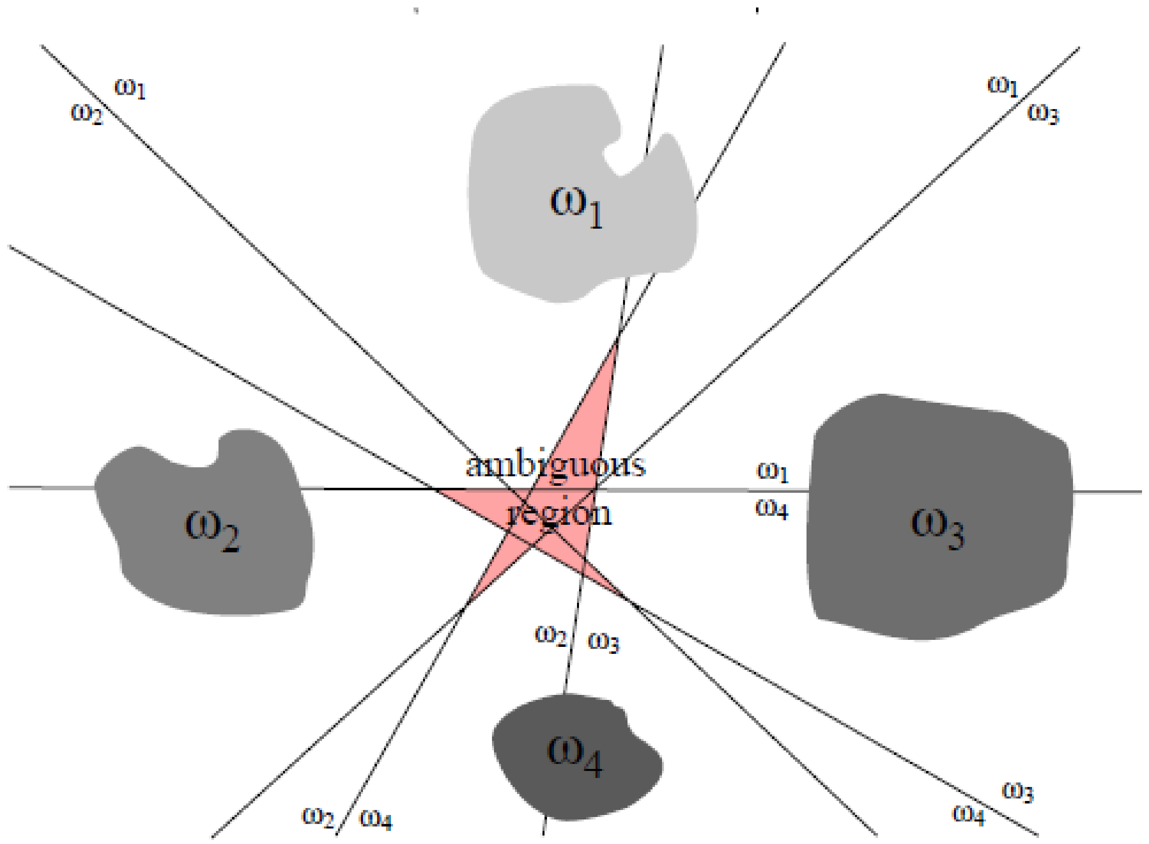
In OVO method, for a *c*-class problem, c(c–1)/2 discriminant functions are constructed.

A discriminant function C_{ij} is trained using samples from classes *i* and *j*, containing positive and negative samples, respectively.

Whenever the decision function value for a test sample is positive from C_{ij}, the vote for class *i* is incremented by one.

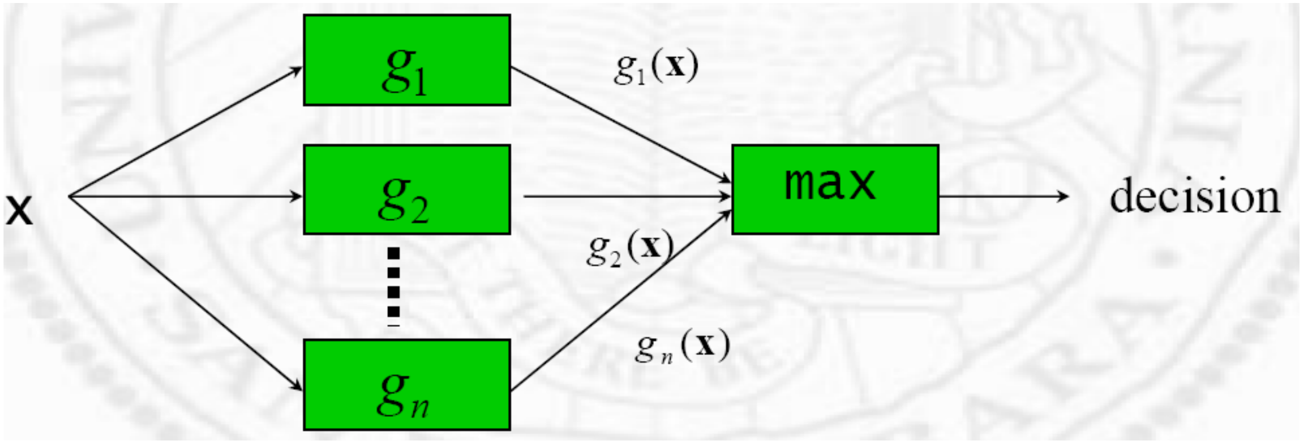
Otherwise, the vote for class *j* is increased by one. The sample is assigned to the class with the maximum number of votes.

One versus one classification



Linear Discriminant Functions: multi-category case (cont'd)

- To avoid the problem of ambiguous regions:
  - Define *c* linear discriminant functions
  - Assign **x** to ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$.
- The resulting classifier is called a linear machine



Linear Discriminant Functions

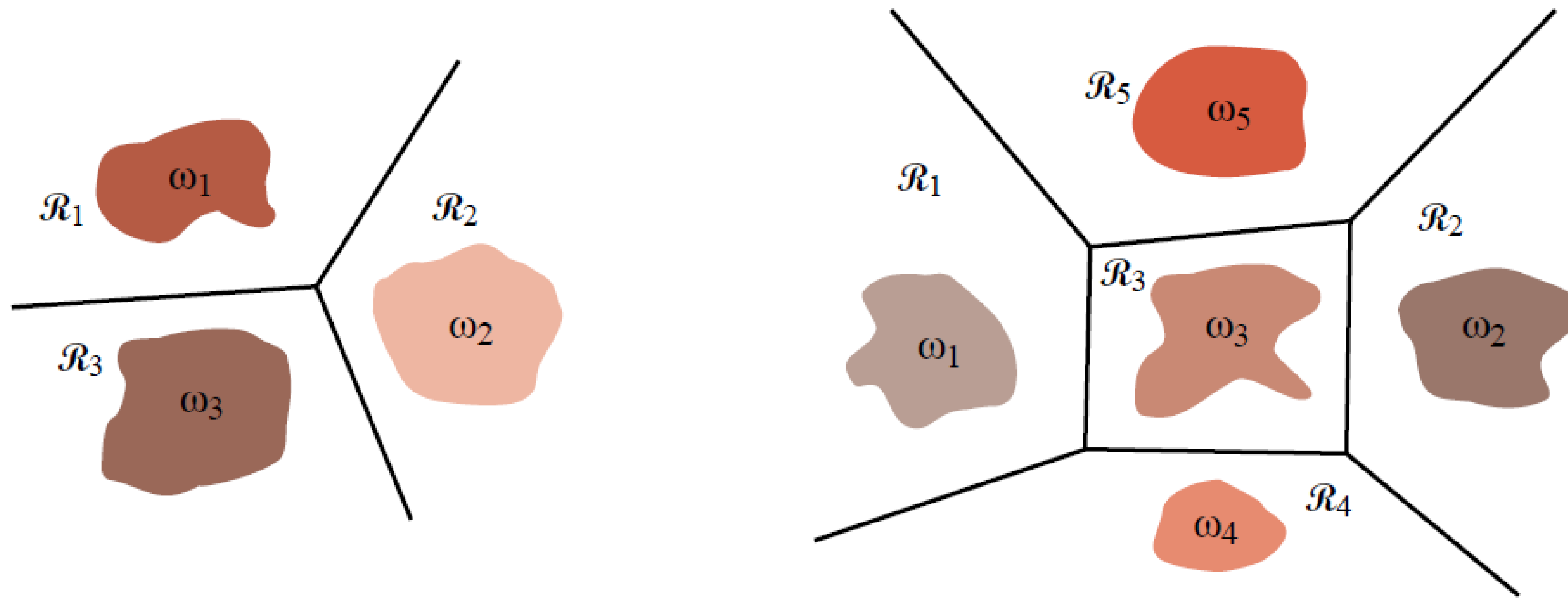


Figure 5.4: Decision boundaries produced by a linear machine for a three-class problem and a five-class problem.

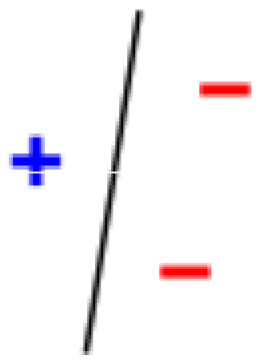
Linear Discriminant Functions

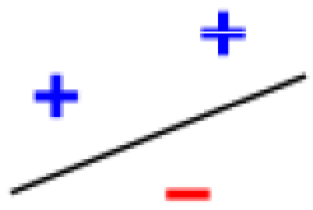
$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\mathbf{x})$$

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

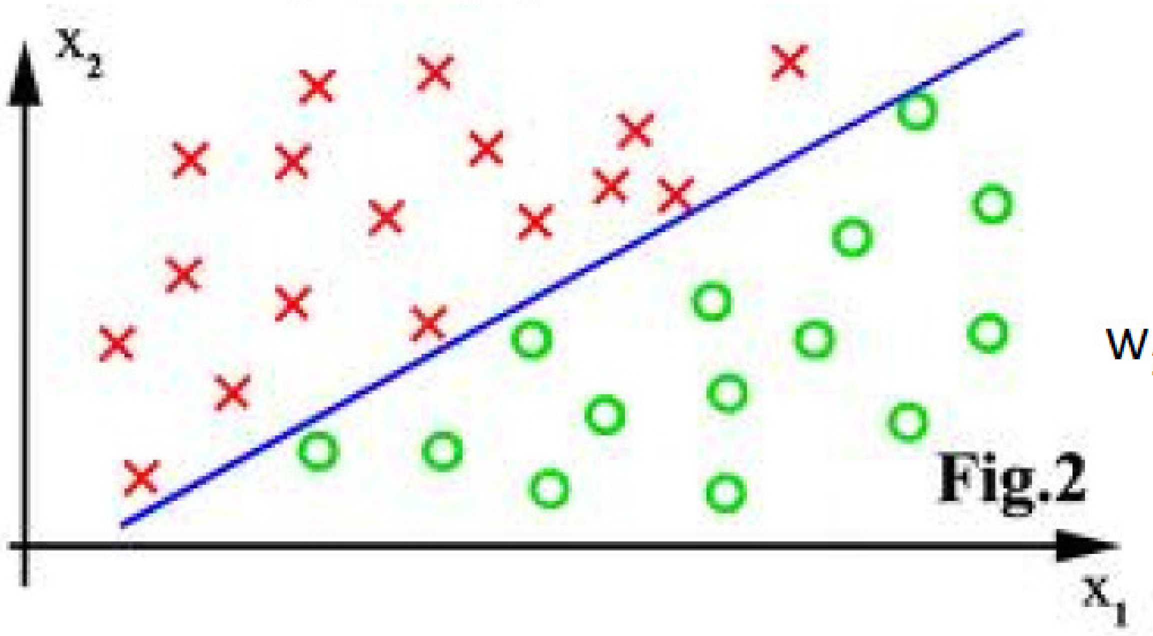
where **a** is now a $\hat{d}$-dimensional weight vector, and where the $\hat{d}$ functions $y_i(\mathbf{x})$ — sometimes called φ functions

Linearly separable data





$w_1x_1+w_2x_2+b>0$

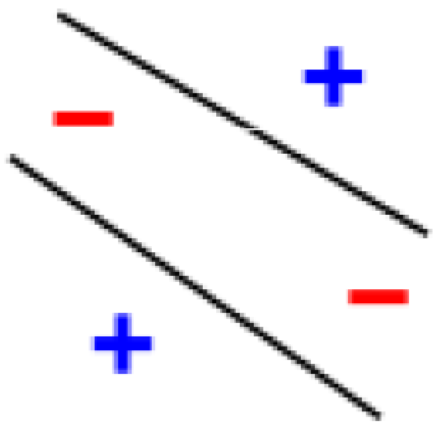


$w_1x_1+w_2x_2+b<0$

Linearly separable data

Separating line： $w_1x_1+w_2x_2+b=0$

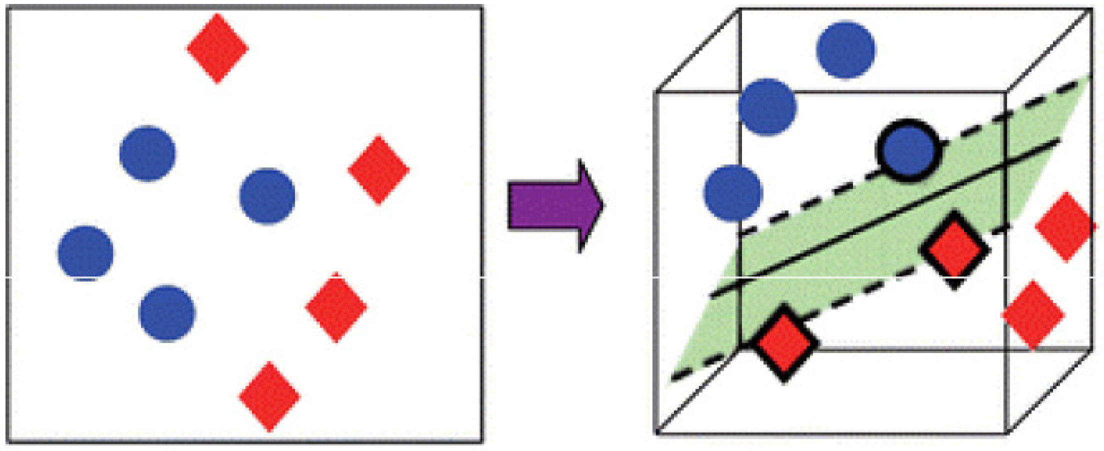
Non- linearly separable data



Covers Theorem

- The theorem states that given a set of training data that is not linearly separable, one can transform it into a training set that is linearly separable by mapping it into a possibly higher-dimensional space via some non-linear transformation.

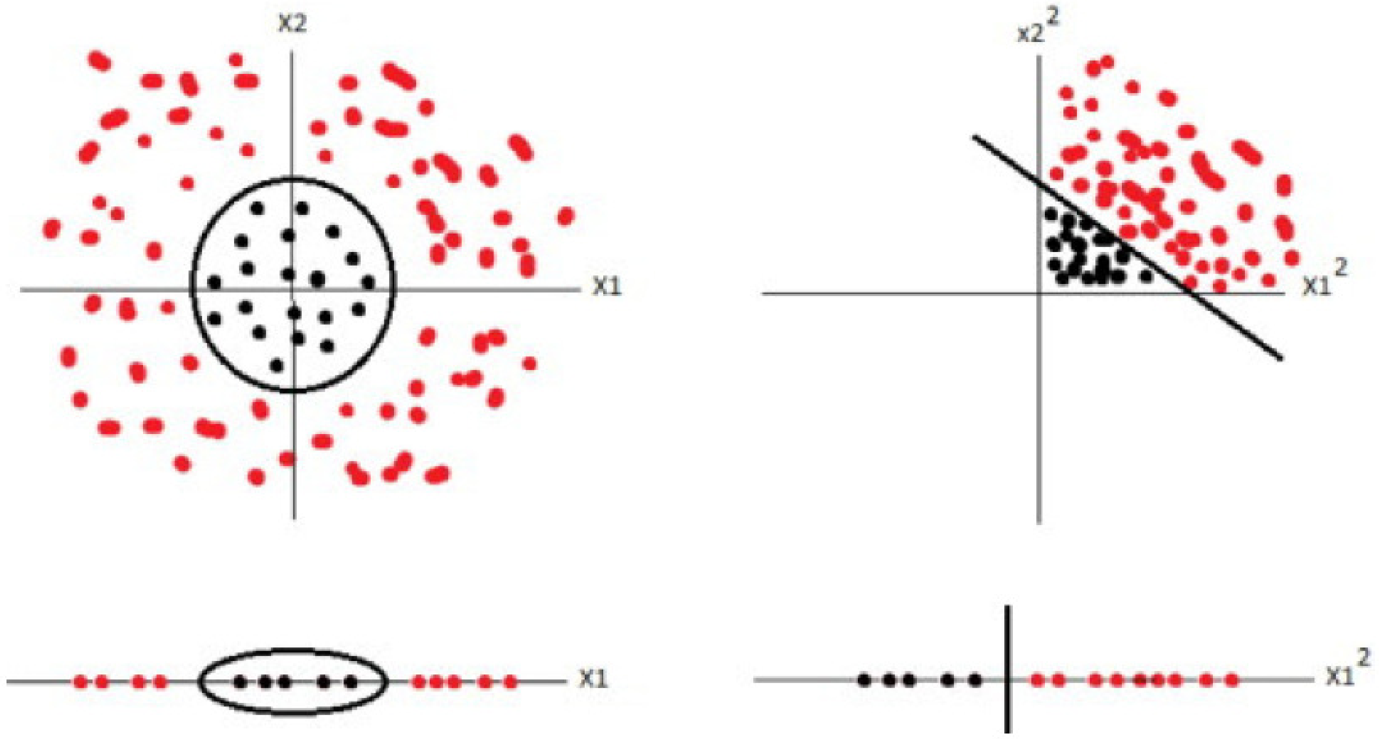
Cover's Theorem



(a) (b)

The samples of the original data is in 2D. After a non-linear transformation , it becomes linearly separable in three dimensions as shown in (b).

Cover’s Theorem



- Note there we had obtained linear /quadratic discriminant functions with generative classifiers as well (Nearest mean and Mahalanobis based classifiers)

- The focus in the remaining part of the lectures will be on learning the weights of the discriminant functions using discriminative modeling techniques.

Linear Discriminant Functions

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i \qquad \text{where we set } x_0 = 1.$$

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

$\mathbf{y}$ is sometimes called an *augmented feature vector*

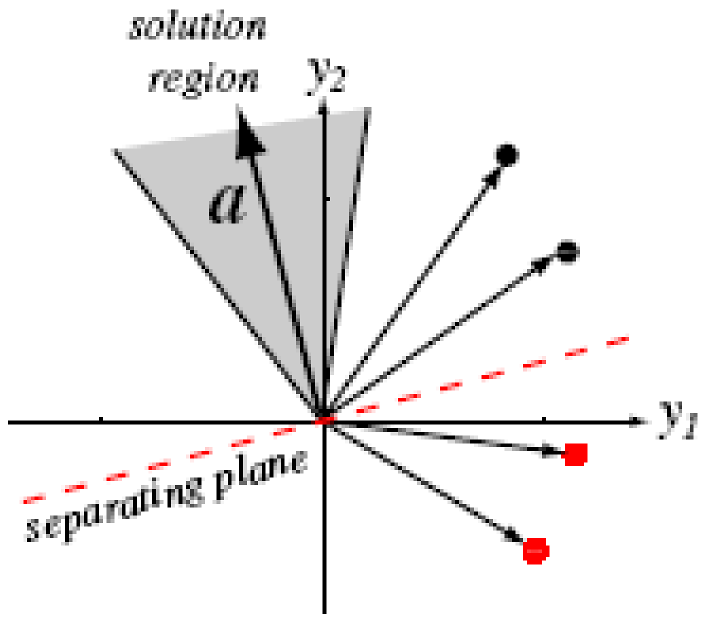
Linear Discriminant Functions

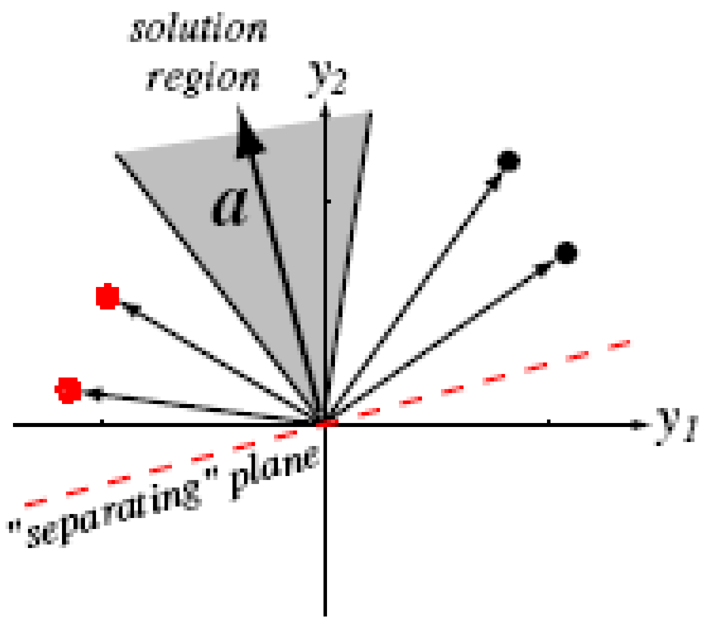
$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \\ \mathbf{w} \\ \\ \end{bmatrix}$$

A sample $\mathbf{y}_i$ is classified correctly if $\mathbf{a}^t\mathbf{y}_i > 0$ and $\mathbf{y}_i$ is labelled ω_1 or if $\mathbf{a}^t\mathbf{y}_i < 0$ and $\mathbf{y}_i$ is labelled ω_2.

Normalized Version

- If $\mathbf{y}_i$ in ω_2, replace $\mathbf{y}_i$ by $-\mathbf{y}_i$
- Find $\mathbf{a}$ such that: $\mathbf{a}^t\mathbf{y}_i>0$





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.*

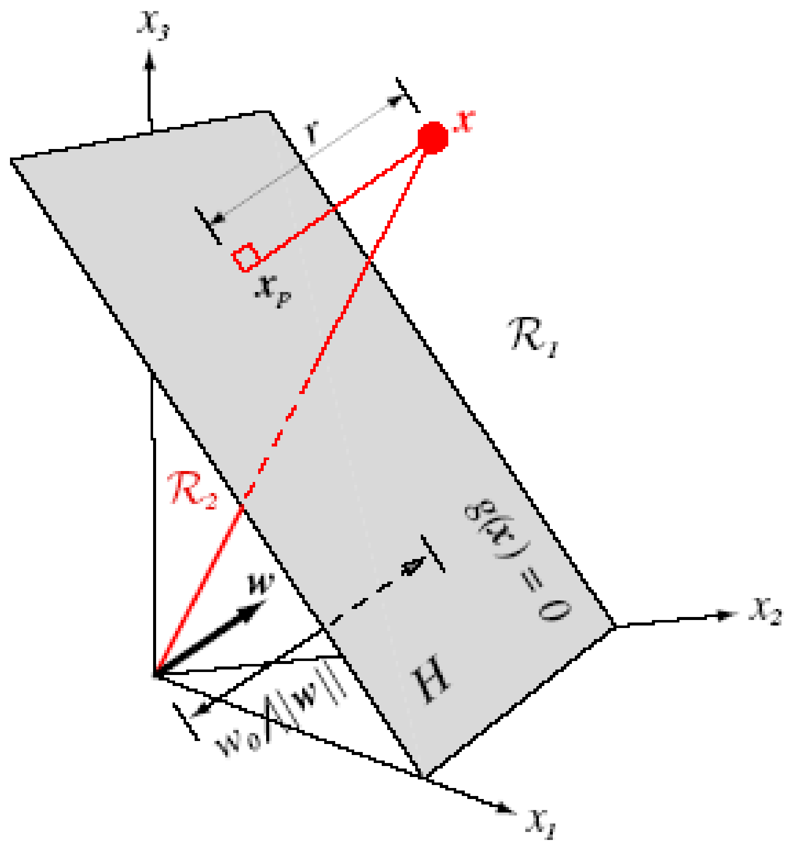
- Given a linear discriminant function

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

the goal is to **learn** the weights (parameters) **a** using a set of ***n*** labeled samples $\mathbf{y}_i$ where each $\mathbf{y}_i$ has a class label ω_1 or ω_2.

Solution vector is usually not unique; we can impose certain constraints to enforce uniqueness,

- Example 1: Find unit-length weight vector that maximizes the minimum distance from the samples to the separating plane



Example 2: Find minimum-length weight vector satisfying the constraint shown below where b is a positive constant called margin.

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

–

Method of steepest descent

- Taylor series expansion when **x** is displaced by **d**, a feasible direction

$$f(\mathbf{x}+\mathbf{d}) = f(\mathbf{x}) + \mathbf{d}^T \nabla f(\mathbf{x})$$

- For minimizing a function

$$f(\mathbf{x}+\mathbf{d}) < f(\mathbf{x})$$

$$=> \mathbf{d}^T \nabla f(\mathbf{x}) < 0$$

Cauchy Schwartz Inequality

$$\left\|\mathbf{d}^T \nabla f(\mathbf{x})\right\| \leq \left\|\mathbf{d}\right\| \; \left\|\nabla f(\mathbf{x})\right\|$$

Method of steepest descent

Equality sign is satisfied when

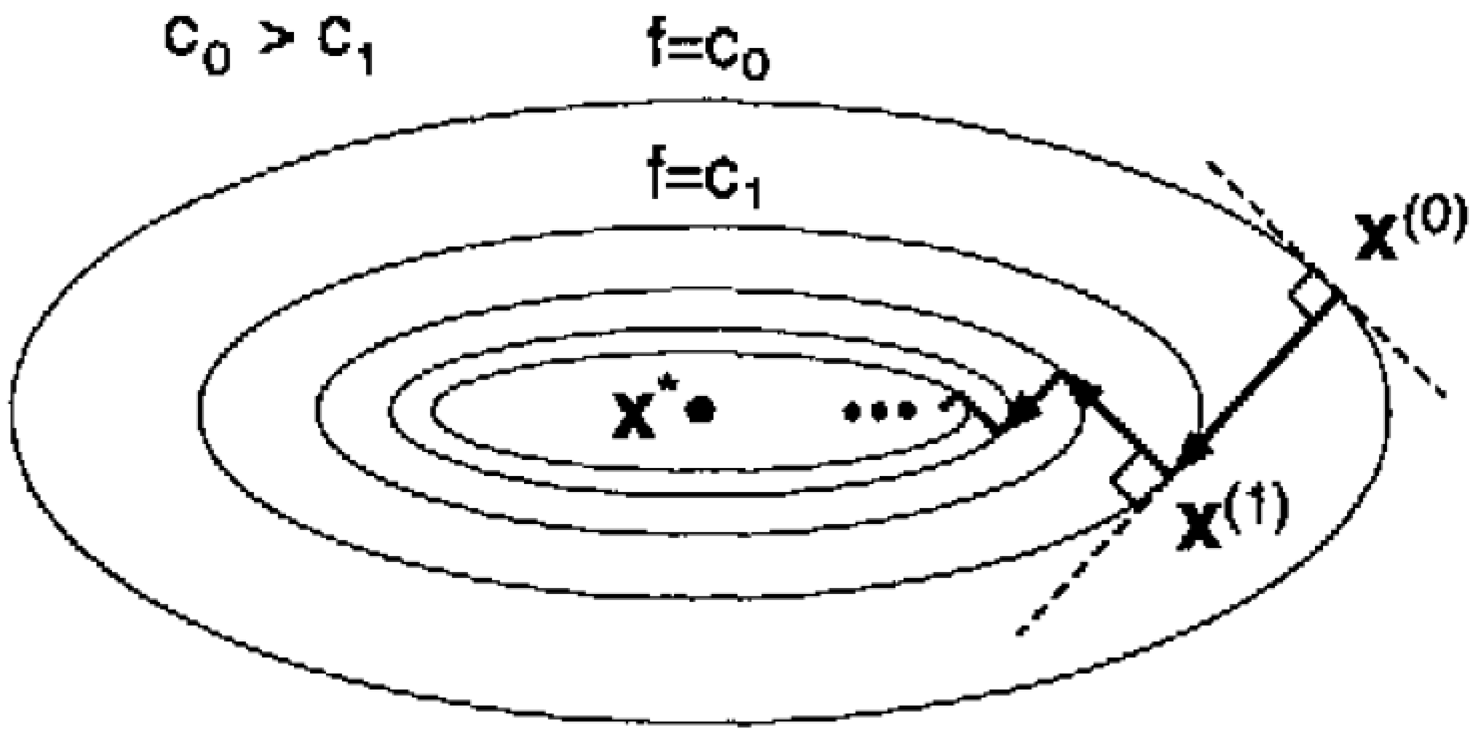
$$\mathbf{d} = \nabla f(\mathbf{x})$$

For minimizing a function , we need to move in the direction negative of the gradient direction

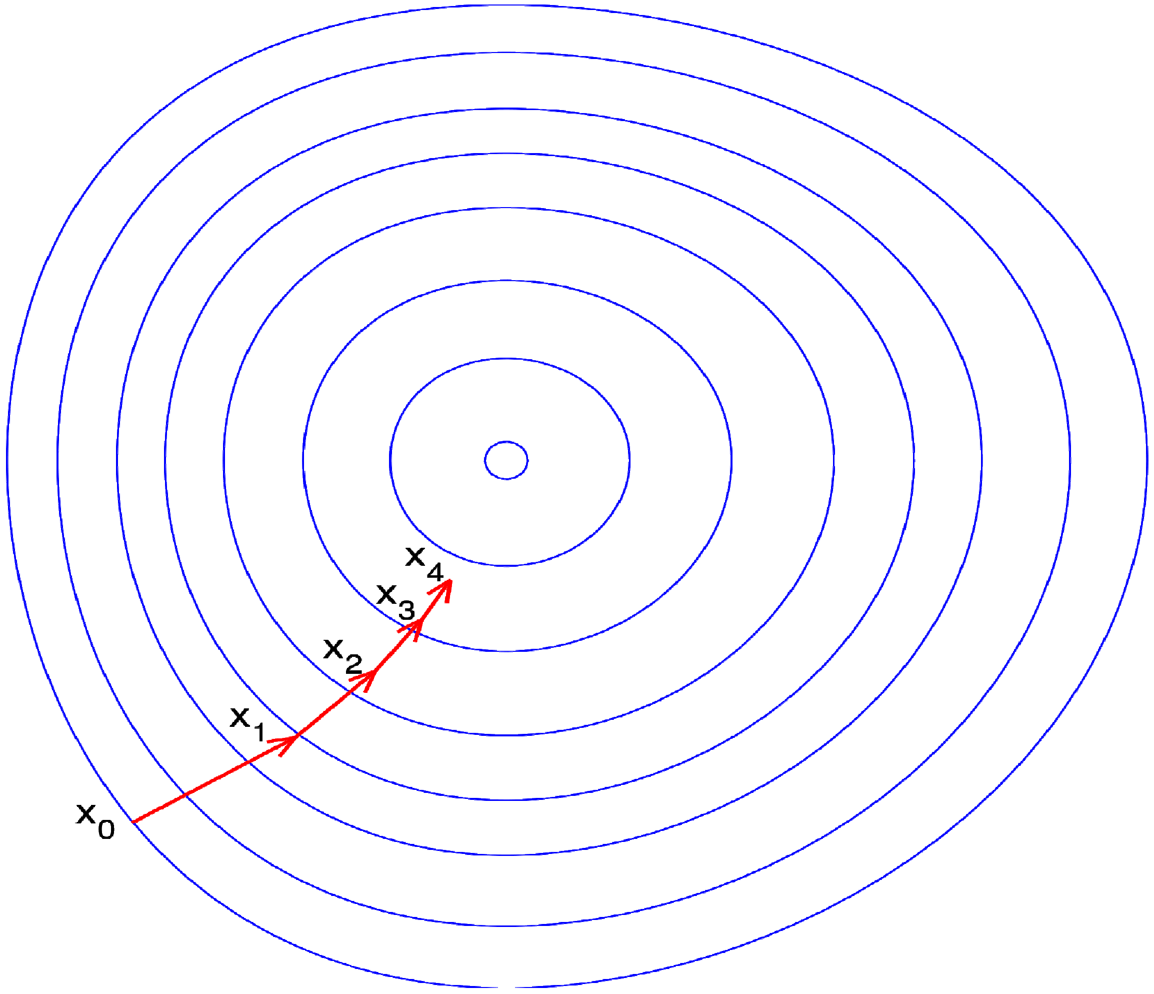
$$\left\| \mathbf{d}^T \nabla f(\mathbf{x}) \right\| = -\left\| \mathbf{d}^T \mathbf{d} \right\|$$

–

Method of steepest descent



Method of steepest descent



Algorithm 1 (Basic gradient descent)

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

Perceptron Algorithm

The Perceptron Criterion Function

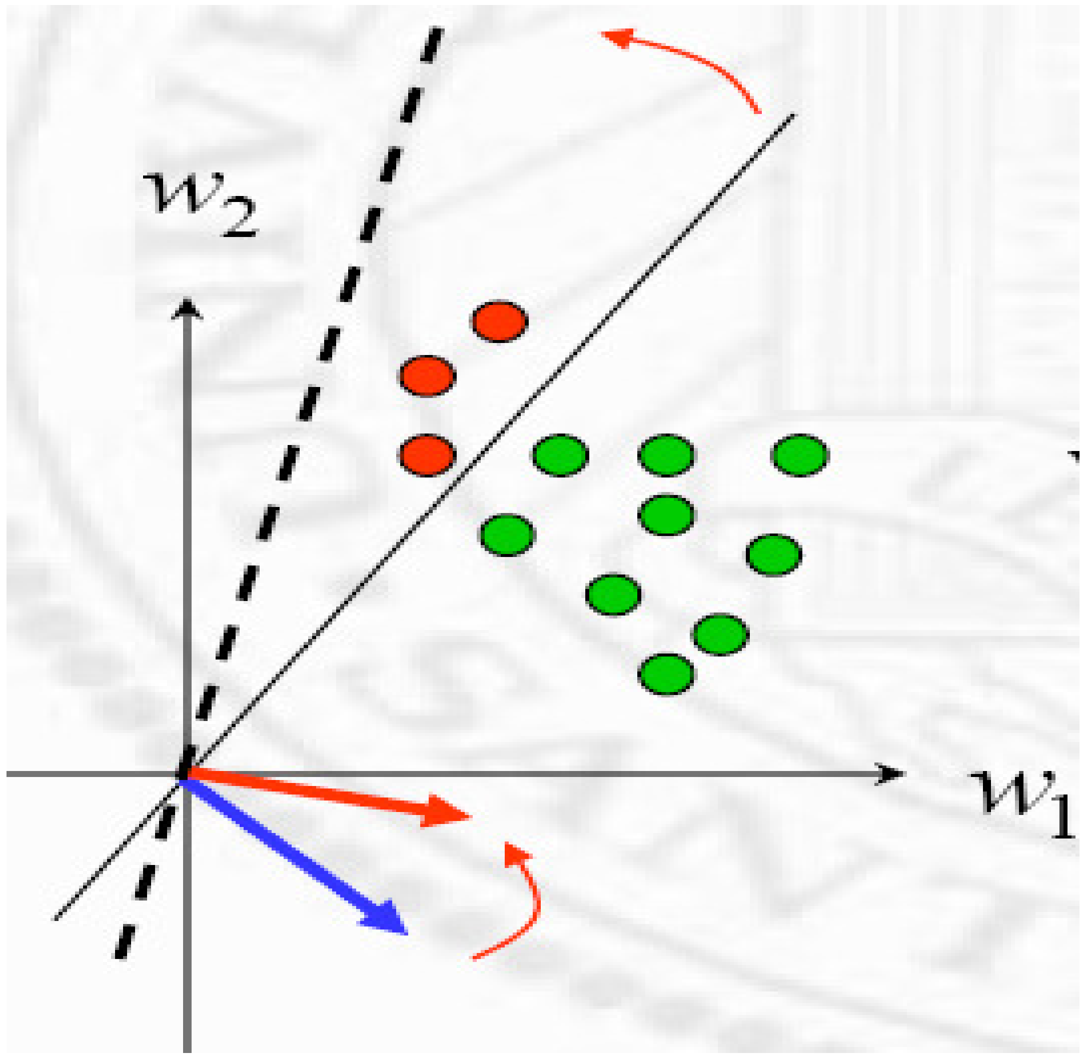
$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^t \mathbf{y})$$

where $\mathcal{Y}(\mathbf{a})$ is the set of samples *misclassified* by $\mathbf{a}$. (If no samples are misclassified, $\mathcal{Y}$ is empty and we define J_p to be zero.) Since $\mathbf{a}^t\mathbf{y} \leq 0$ if $\mathbf{y}$ is misclassified, $J_p(\mathbf{a})$ is never negative, being zero only if $\mathbf{a}$ is a solution vector.

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

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

Perceptron rule



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 a
6 end
```

Algorithm 4 (Fixed-increment single-sample Perceptron)

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

Algorithm 5 (Variable increment Perceptron with margin)

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

Perceptron Convergence Theorem: If training samples are linearly separable, then the sequence of weight vectors by the above algorithm will terminate at a solution vector in a finite number of steps.