6. Linear Discriminant Functions

Linear Discriminant Functions

- Assumption: we *know* the proper *forms* for the *discriminant functions*, and use the samples to estimate the values of parameters of the classifier

- The problem of minimizing a criterion function
  - *Sample risk*, or *training error* – the average loss incurred in classifying the set of training samples

Linear Discriminant Functions

- A classifier that uses linear discriminant functions is called "*a linear machine*"

$$g_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

- The decision surfaces for a linear machine are pieces of hyperplanes defined by:

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

Linear Discriminant Functions

- **The two-category case**

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) = \mathbf{w}_1^T\mathbf{x} + w_{10} - (\mathbf{w}_2^T\mathbf{x} + w_{20})$$

$$= (\mathbf{w}_1 - \mathbf{w}_2)^T\mathbf{x} + (w_{10} - w_{20}) = \mathbf{w}^T\mathbf{x} + w_0$$

$\mathbf{w}$ is the weight vector

w_0 is the bias, or threshold weight

Decision rule: **decide ω_1 if g(x)>0, ω_2 otherwise**

Linear Discriminant Functions

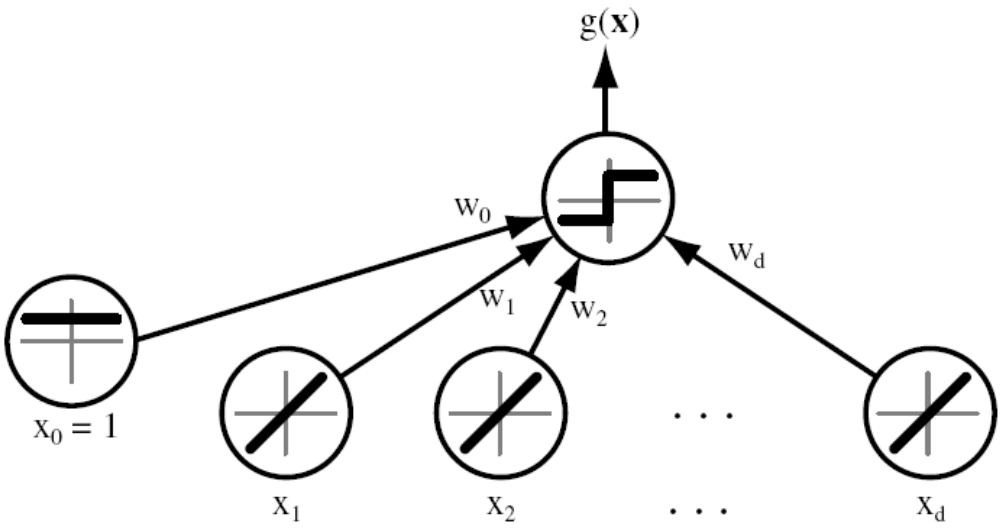
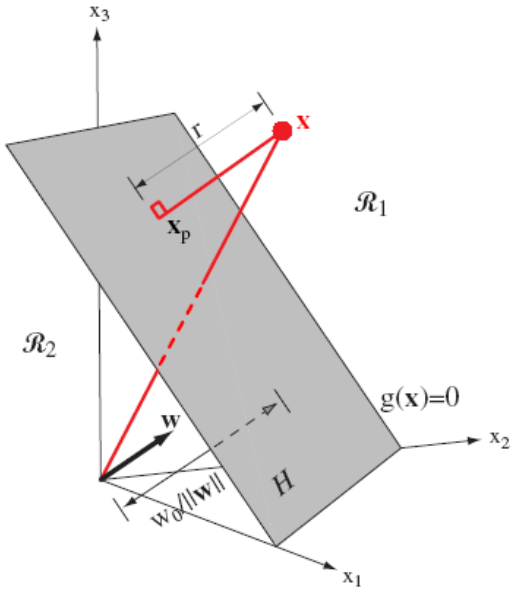


Figure 5.1: A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the output unit sums all these products and emits a $+1$ if $\mathbf{w}^t\mathbf{x} + w_0 > 0$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

Linear Discriminant Functions

- g(x) = 0 defines a hyperplane separating the feature space into decision regions R_1 and R_2
- g(x) gives an algebraic measure of the distance from x to the hyperplane H



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

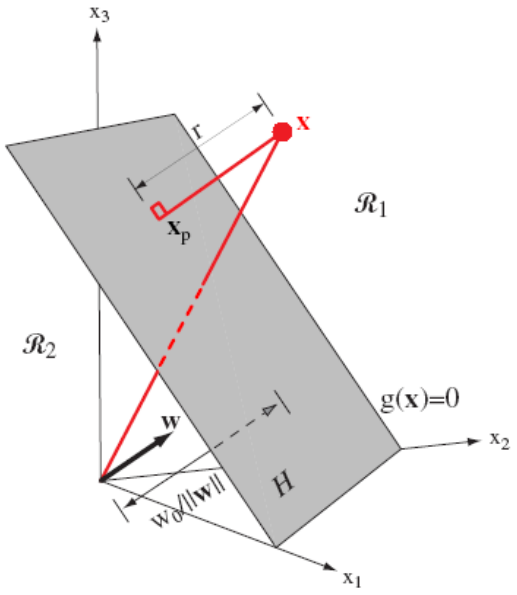
Since $g(\mathbf{x}_p) = 0$:

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

Linear Discriminant Functions

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

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



- **The orientation of H is determined by the normal vector w, and the location of H is determined by the bias** w_0

The Multicategory Case

- One versus all
- One versus one

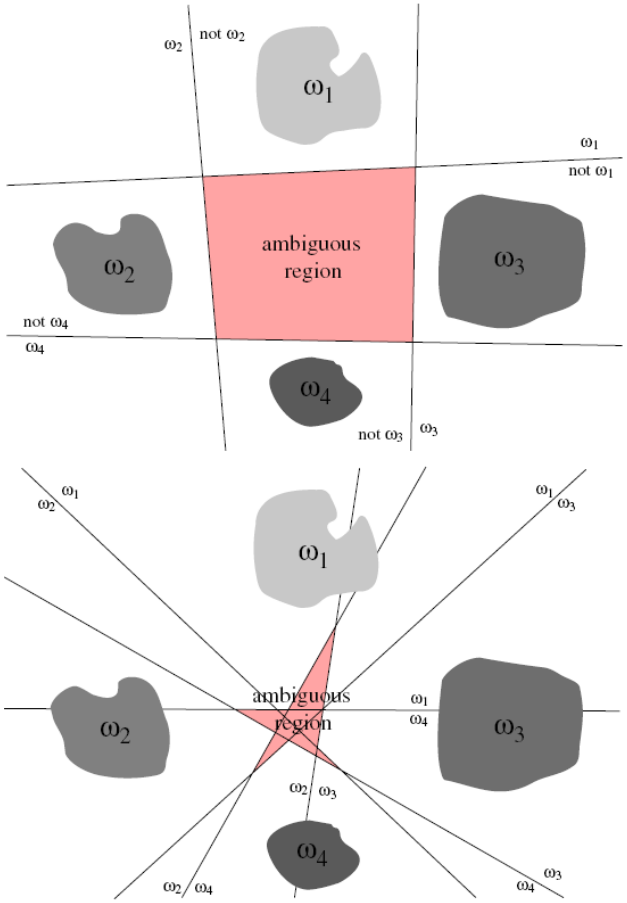


Figure 5.3: Linear decision boundaries for a four-class problem. The top figure shows ω_i/not ω_i dichotomies while the bottom figure shows ω_i/ω_j dichotomies. The pink regions have ambiguous category assigments.

The Multicategory Case

- To avoid ambiguous regions:
  - Define c linear discriminant functions

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

  - Assign x to class ω_i if $g_i(x) > g_j(x)$ for all j different from i

The Multicategory Case

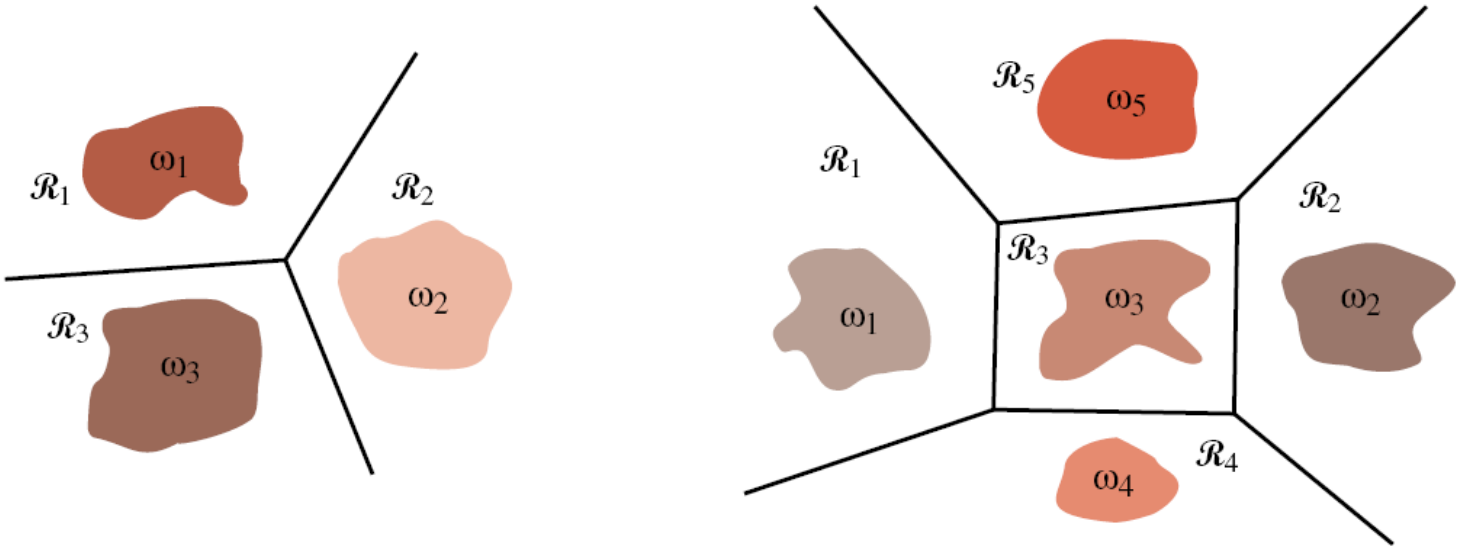


Figure 5.4: Decision boundaries produced by a linear machine for a three-class problem and a five-class problem.

Generalized Linear Discriminant Functions

- **Linear discriminant function:**

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

- **Polynomial discriminant functions:**

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

$$\sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d} w_{ijk} x_i x_j x_k + \ldots$$

Generalized Linear Discriminant Functions

- **Generalized linear discriminant function:**

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

$$\mathbf{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_{\hat{d}} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1(\mathbf{x}) \\ \vdots \\ y_{\hat{d}}(\mathbf{x}) \end{bmatrix}$$

Generalized Linear Discriminant Functions

- **Generalized linear discriminant function:**

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

- **The resulting discriminant function is not linear in x, but it is linear in y**
- **The functions $y_i(x)$ map points in d-dimensional x-space to points in $\hat{d}$ -dimensional y-space**

Example

$$g(x) = a_1 + a_2 x + a_3 x^2$$

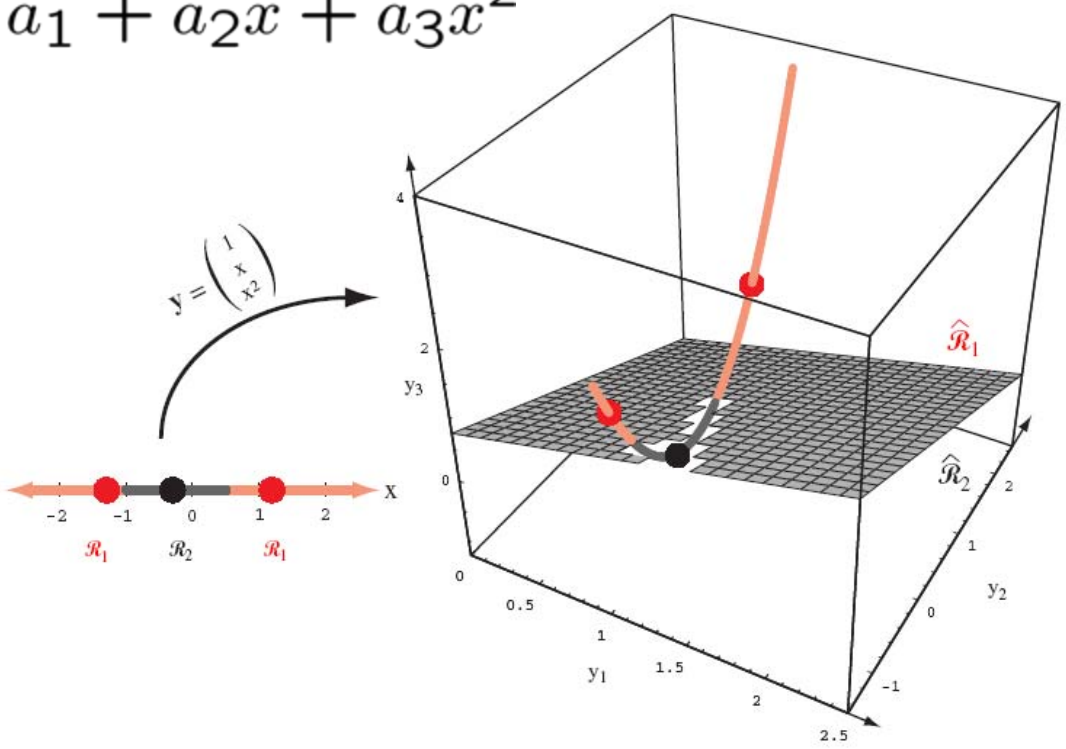


Figure 5.5: The mapping $\mathbf{y} = (1, x, x^2)^t$ takes a line and transforms it to a parabola in three dimensions. A plane splits the resulting $\mathbf{y}$ space into regions corresponding to two categories, and this in turn gives a non-simply connected decision region in the one-dimensional x space.

Example

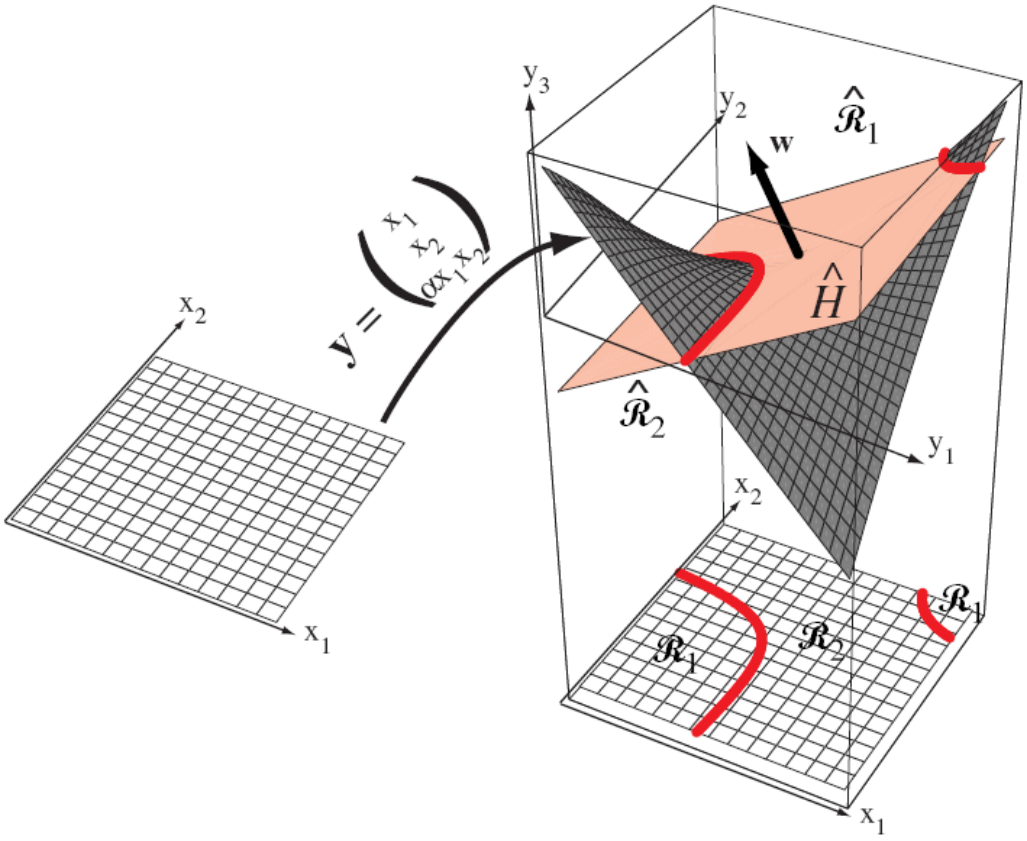


Figure 5.6: The two-dimensional input space $\mathbf{x}$ is mapped through a polynomial function f to $\mathbf{y}$. Here the mapping is $y_1 = x_1$, $y_2 = x_2$ and $y_3 \propto x_1 x_2$. A linear discriminant in this transformed space is a hyperplane, which cuts the surface. Points to the positive side of the hyperplane $\hat{H}$ correspond to category ω_1, and those beneath it ω_2. Here, in terms of the $\mathbf{x}$ space, $\mathcal{R}_1$ is a not simply connected.

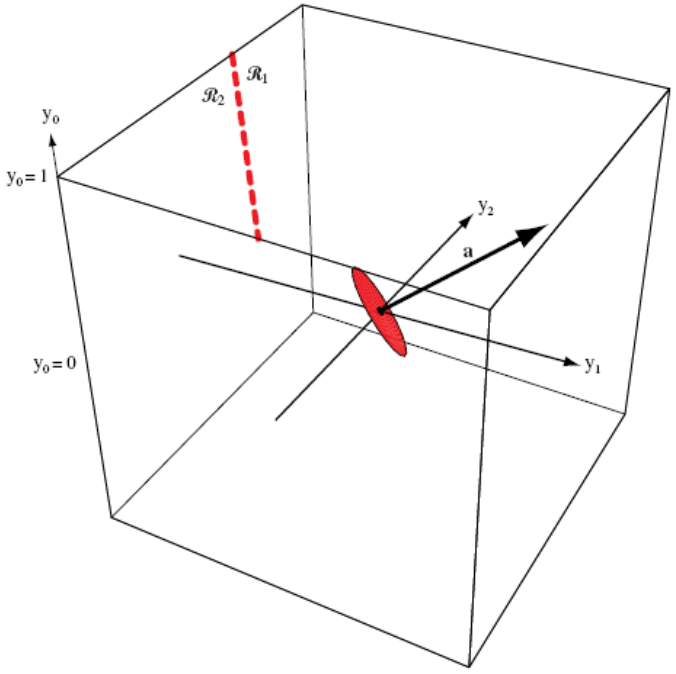
Example

http://www.youtube.com/watch?v=3liCbRZPrZA

Generalized Linear Discriminant Functions

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

$$\mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$



Two-Category Linearly Separable Case

- We have a set of n samples y_1, …, y_n some labelled ω_1 and some labelled ω_2

- We want to use these samples to determine **a** in $g(\mathbf{x}) = \mathbf{a}^T\mathbf{y}$

- If we can find such **a** that all of the samples are correctly classified, the samples are *linearly separable*

Two-Category Linearly Separable Case

- $g(\mathbf{x}) = \mathbf{a}^T\mathbf{y}$
- $\mathbf{y}_i$ is classified correctly if $\mathbf{a}^T\mathbf{y}_i > 0$ and $\mathbf{y}_i$ is labelled ω_1 or $\mathbf{a}^T\mathbf{y}_i < 0$ and $\mathbf{y}_i$ is labelled ω_2
- Normalization: replace all samples labelled ω_2 by their negatives

⇩

- We'll look for such *solution vector* **a** that $\mathbf{a}^T\mathbf{y}_i > 0$ for all of the samples

Two-Category Linearly Separable Case

- **a** can be thought as a point in weight space
- Each y_i places a constraint on the possible location of **a**
- The solution vector must be on the positive side of every hyperplane

$$\mathbf{a}^T\mathbf{y}_i = 0$$

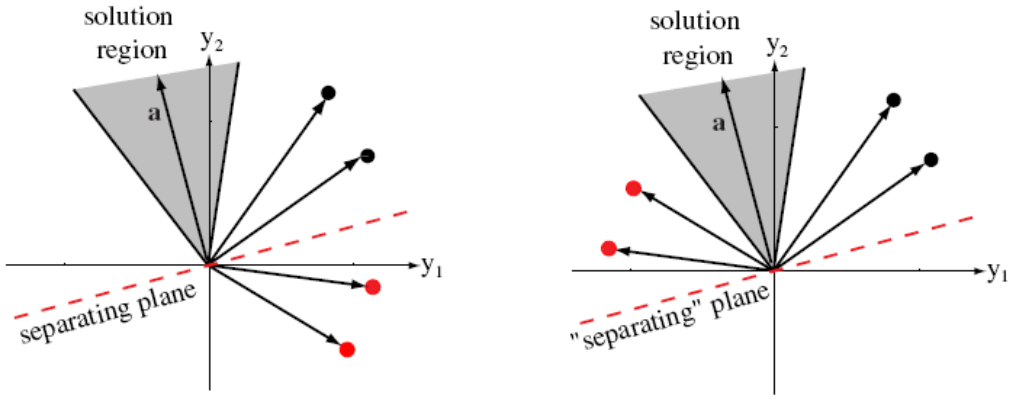


Figure 5.8: Four training samples (black for ω_1, red for ω_2) and the solution region in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been "normalized" — i.e., changed in sign. Now the solution vector leads to a plane that places all "normalized" points on the same side.

Two-Category Linearly Separable Case

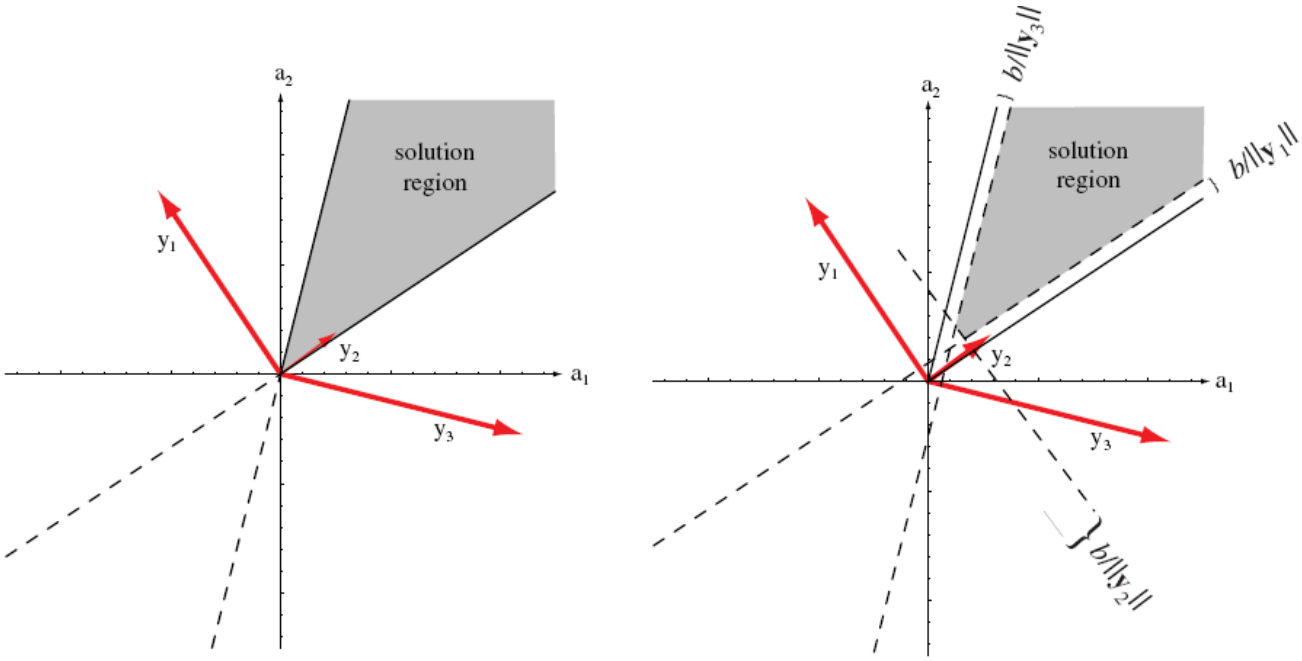


Figure 5.9: The effect of the margin on the solution region. At the left, the case of no margin ($b = 0$) equivalent to a case such as shown at the left in Fig. 5.8. At the right is the case $b > 0$, shrinking the solution region by margins $b/\|\mathbf{y}_i\|$.

Two-Category Linearly Separable Case

- ***Gradient descent procedures:*** **define a criterion function J(a) that is minimized if a is a solution vector**
- ***Basic gradient descent:***
  - **Start with arbitrary weight vector a(1)**
  - **Compute** $\nabla J(\mathbf{a}(1))$
  - **Repeat iterations (moving weight vector in the direction of steepest descent):**

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

$\eta(k)$ is learning rate

Two-Category Linearly Separable Case

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

- Choice of the learning rate is important
  - If it is too small, convergence is too slow
  - If it is too large, the correction process will overshoot

Two-Category Linearly Separable Case

- **Setting the learning rate:**

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

H is the Hessian matrix of second partial derivatives $\partial^2 J / \partial a_i \partial a_j$

- Algorithm 2 (Newton descent)

```
1 begin initialize  a, criterion θ
2           do
3               a ← a − H^{-1} ∇J(a)
4           until H^{-1} ∇J(a) < θ
5     return a
6 end
```

Two-Category Linearly Separable Case

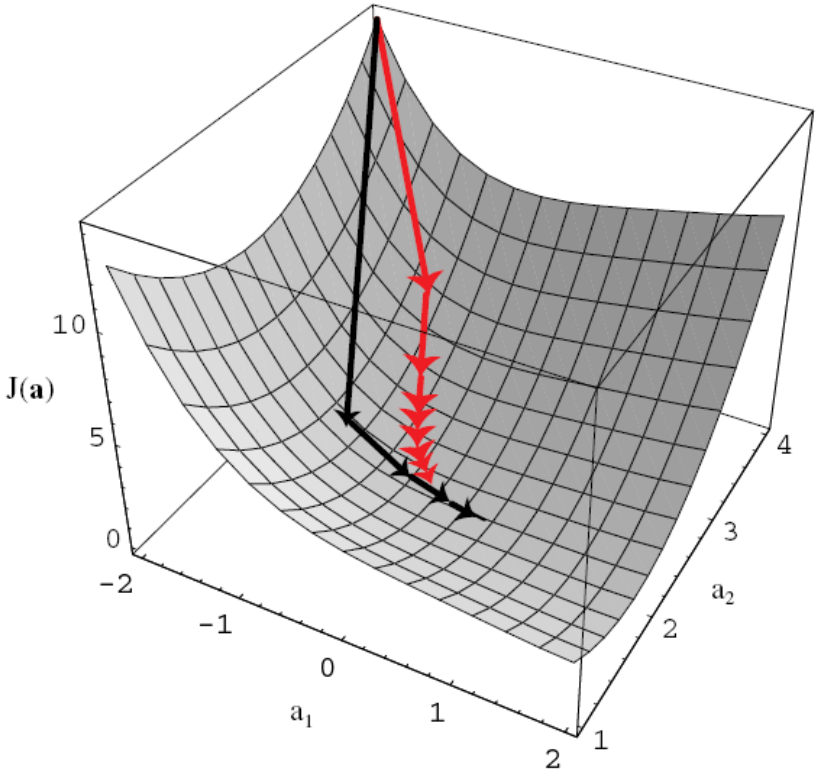


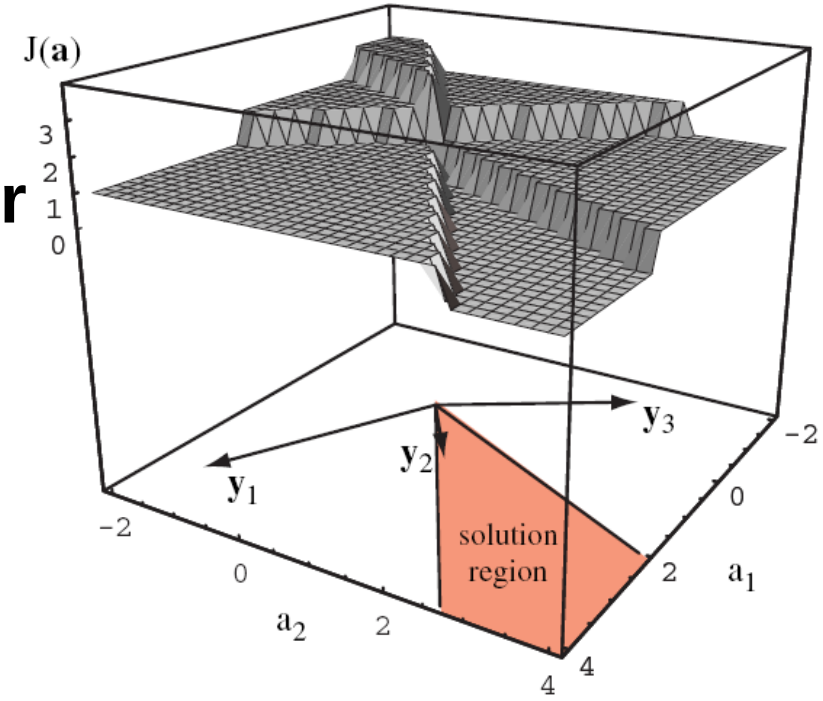
Figure 5.10: The sequence of weight vectors given by a simple gradient descent method (red) and by Newton's (second order) algorithm (black). Newton's method typically leads to greater improvement per step, even when using optimal learning rates for both methods. However the added computational burden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

Minimizing the Perceptron Criterion Function

- **Problem: to construct a criterion function for solving** $\mathbf{a}^T\mathbf{y}_i > 0$

- **Possibility: Let** $J(\mathbf{a}; \mathbf{y}_1, ..., \mathbf{y}_n)$ **be the number of samples misclassified by a.**

- **Disadvantage: this function is piecewise constant**

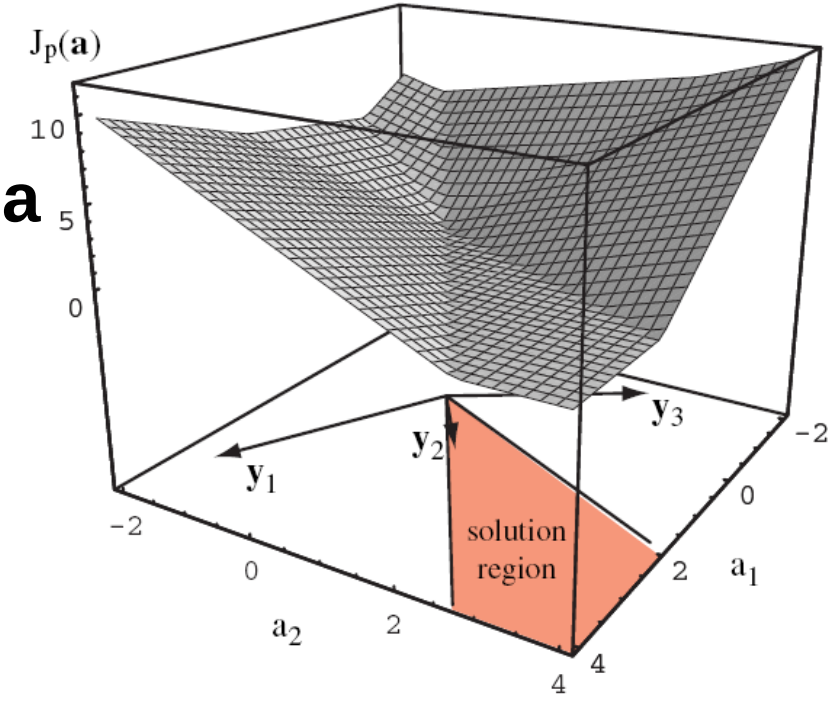


Minimizing the Perceptron Criterion Function

- ***Perceptron criterion function***:

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

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



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

Minimizing the Perceptron Criterion Function

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

(since the jth component of the gradient of J_p is $\partial J_p / \partial a_j$)

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

Minimizing the Perceptron Criterion Function

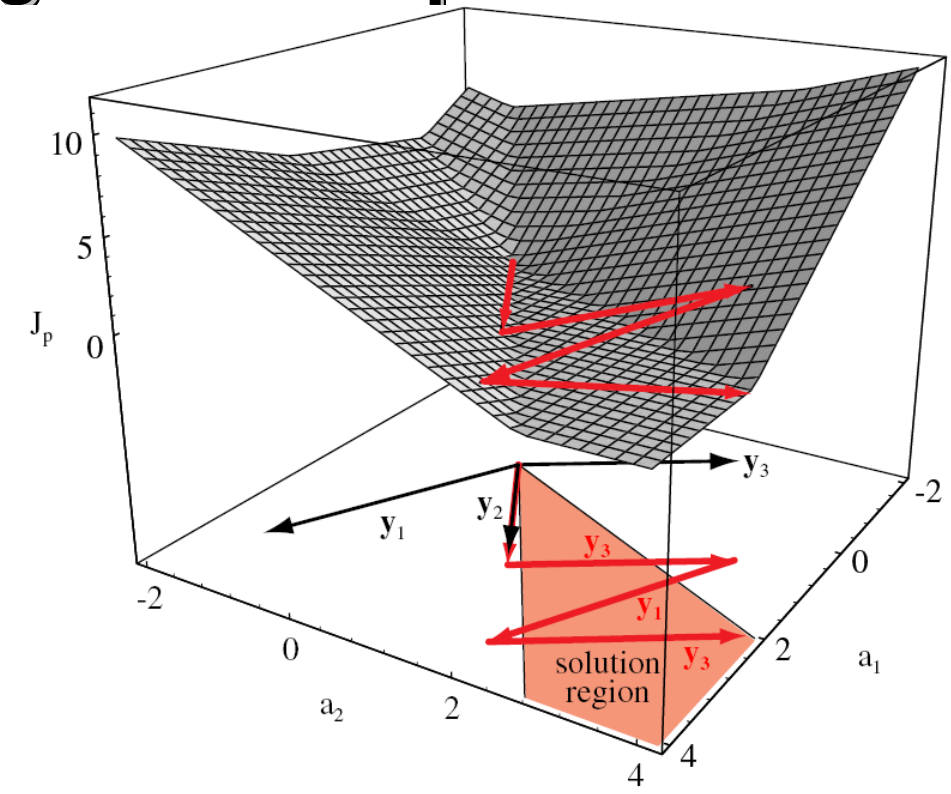


Figure 5.12: The Perceptron criterion, J_p is plotted as a function of the weights a_1 and a_2 for a three-pattern problem. The weight vector begins at $\mathbf{0}$, and the algorithm sequentially adds to it vectors equal to the "normalized" misclassified patterns themselves. In the example shown, this sequence is $\mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_1, \mathbf{y}_3$, 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 (cf. Theorem 5.1. (In an alternate, batch method, *all* the misclassified points are added at each iteration step leading to a smoother trajectory in weight space.)

Minimizing the Perceptron Criterion Function

- Batch *Perceptron algorithm* : the next weight vector is obtained by adding some multiple of the sum of the misclassified samples to the present weight vector

Minimizing the Perceptron Criterion Function

Algorithm 5 (Variable increment Perceptron with margin)

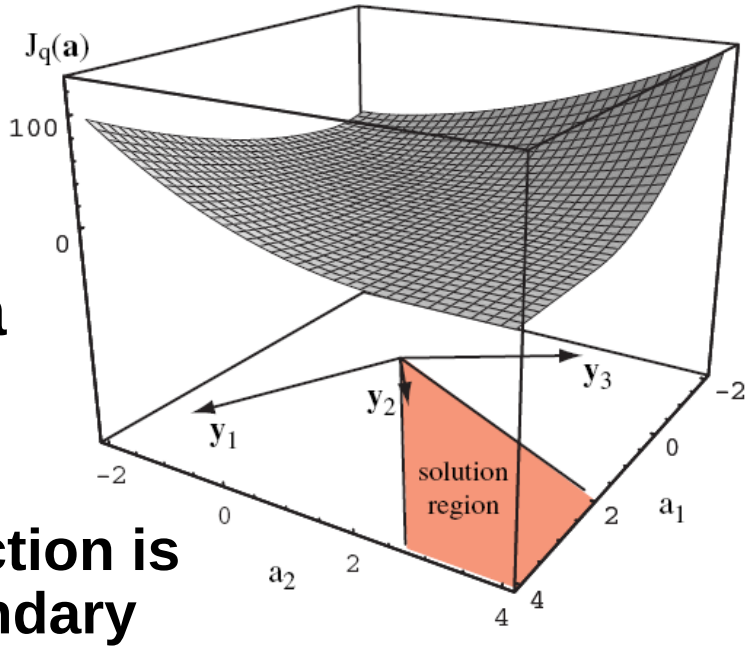
1 **begin initialize** $\mathbf{a}$, criterion θ, margin $b, \eta(\cdot), k = 0$
2 **do** $k \leftarrow k + 1$
3 **if** $\mathbf{a}^t \mathbf{y}_k + b < 0$ **then** $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\mathbf{y}_k$
4 **until** $\mathbf{a}^t \mathbf{y}_k + b > 0$ for all k
5 **return** $\mathbf{a}$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

Relaxation Procedures

Descent algorithm:

- Another possible criterion function is:

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



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

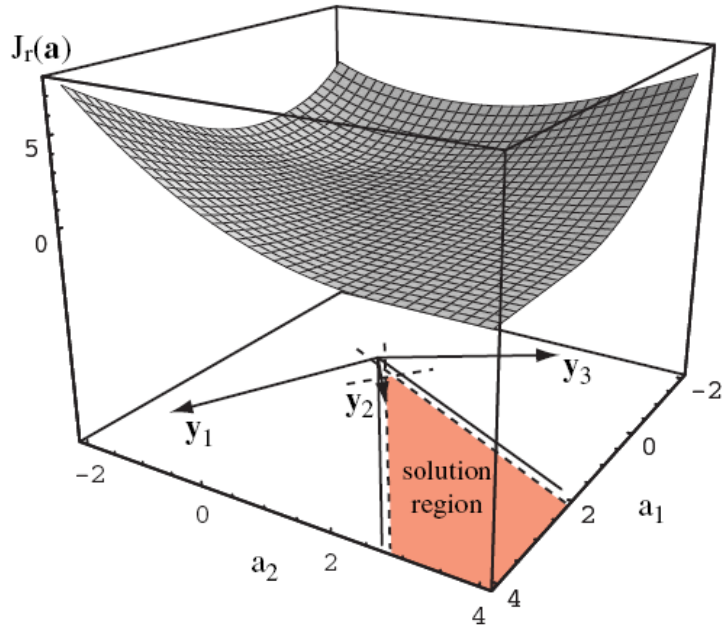
- Advantage: gradient is continuous
- Disadvantages: 1) the function is very smooth near the boundary of the solution region; 2) value is dominated by the longest sample vectors

Relaxation Procedures

- To avoid these problems, we introduce the criterion function as:

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

where $\mathcal{Y}(\mathbf{a})$ is the set of samples for which $\mathbf{a}^T\mathbf{y} \le b$



$$\nabla J_r = \sum_{y\in\mathcal{Y}}\frac{\mathbf{a}^T\mathbf{y}-b}{\|\mathbf{y}\|^2}\mathbf{y}$$

Relaxation Procedures

- The relaxation algorithm becomes:

Algorithm 8 (Batch relaxation with margin)

1 begin initialize $\mathbf{a}, \eta(\cdot), k = 0$
2 do $k \leftarrow k + 1 \bmod n$
3 $\mathcal{Y}_k = \{\}$
4 $j = 0$
5 do $j \leftarrow j + 1$
6 if $\mathbf{a}^T\mathbf{y}_j \leq b$ 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 a
11 end

Relaxation Procedures

- **Single-sample relaxation rule with margin:**

Algorithm 9 (Single-sample relaxation with margin)

1 begin initialize $\mathbf{a}, \eta(\cdot), k = 0$
2 do $k \leftarrow k + 1 \mod n$
3 if $\mathbf{a}^T\mathbf{y}_k \leq b$ then $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\frac{b-\mathbf{a}^t\mathbf{y}}{\|\mathbf{y}_k\|^2}\mathbf{y}_k$
4 until $\mathbf{a}^T\mathbf{y}_k > b$ for all $\mathbf{y}_k$
5 return a
6 end

Relaxation Procedures

- **In general,** $0 < \eta < 2$

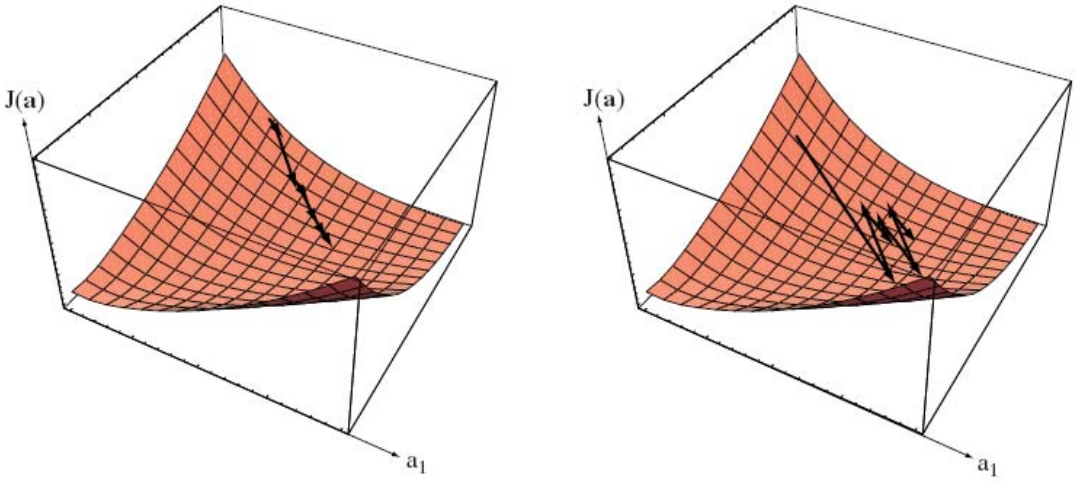


Figure 5.15: At the left, underrelaxation ($\eta < 1$) leads to needlessly slow descent, or even failure to converge. Overrelaxation ($1 < \eta < 2$, shown in the middle) describes overshooting; nevertheless convergence will ultimately be achieved.

Nonseparable behavior

- Perceptron and relaxation procedures are called error-correcting procedures
- In practice they are used if one can believe that the error rate for the optimal LDF is low
- Which technique should one apply if sample set is unseparable?

Minimum Squared Error Procedures

- We will consider now a criterion function that involves ***all*** of the samples
- We shall try to resolve $\mathbf{a}^T\mathbf{y}_i = b_i$ where b_i are some arbitrarily specified positive constants

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

Minimum Squared Error Procedures

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

- If Y were nonsingular, we could obtain a solution as $\mathbf{a} = \mathbf{Y}^{-1}\mathbf{b}$

- Usually Y has more rows than columns → no exact solution for a exists

- We'll search to minimize the squared length of the error vector $\mathbf{e} = \mathbf{Ya} - \mathbf{b}$:

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

Minimum Squared Error Procedures

- We minimize the sum of squared error by a gradient search procedure:

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

$$\nabla J_s = 2\mathbf{Y}^T(\mathbf{Ya} - \mathbf{b}) = 0$$

$$\mathbf{Y}^T\mathbf{Ya} = \mathbf{Y}^T\mathbf{b}$$

- If $\mathbf{Y}^T\mathbf{Y}$ is nonsingular, we can solve for a uniquely as:

$$\mathbf{a} = (\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{Y}^T\mathbf{b} = \mathbf{Y}^\dagger\mathbf{b}$$

where $\mathbf{Y}^\dagger \equiv (\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{Y}^T$ is called the *pseudoinverse* of Y

Example

Example 1: Constructing a linear classifier by matrix pseudoinverse

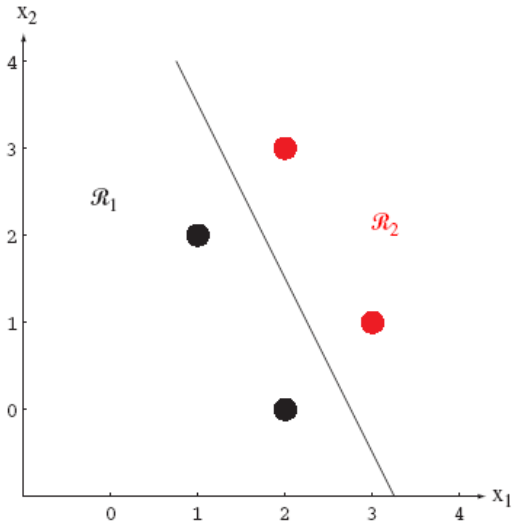
Suppose we have the following two-dimensional points for two categories: ω_1: $(1,2)^t$ and $(2,0)^t$, and ω_2: $(3,1)^t$ and $(2,3)^t$, as shown in black and red, respectively, in the figure.
Our matrix $\mathbf{Y}$ is therefore

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

and after a few simple calculations we find that its pseudoinverse is

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

Example



Four training points and the decision boundary $\mathbf{a}^t \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} = 0$, where $\mathbf{a}$ was found by means of a pseudoinverse technique.

We arbitrarily let all the margins be equal, i.e., $\mathbf{b} = (1,1,1,1)^t$. Our solution is $\mathbf{a} = \mathbf{Y}^\dagger \mathbf{b} = (11/3, -4/3, -2/3)^t$, and leads to the decision boundary shown in the figure. Other choices for $\mathbf{b}$ would typically lead to different decision boundaries, of course.

Demo of MSE procedure:
http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletMSE.html

Support Vector Machines

- Tutorial
  - http://videolectures.net/mlss06tw_lin_svm/
- Library
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm/