Linear Discriminant Functions

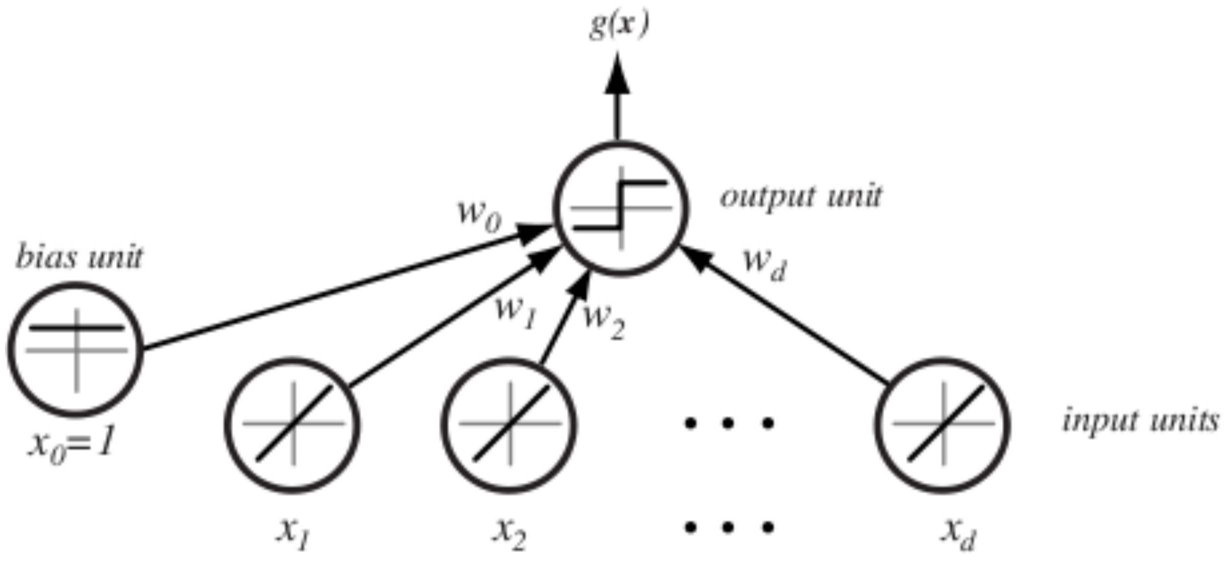
Zach Busser
Owen Roberts
Ganesh Sugunan

Linear discriminant functions:

the linear discriminant function:

$g(x) = w^t x + \omega_0$

x is the point, w is the weight vector, and ω_0 is the bias (t is the transpose).

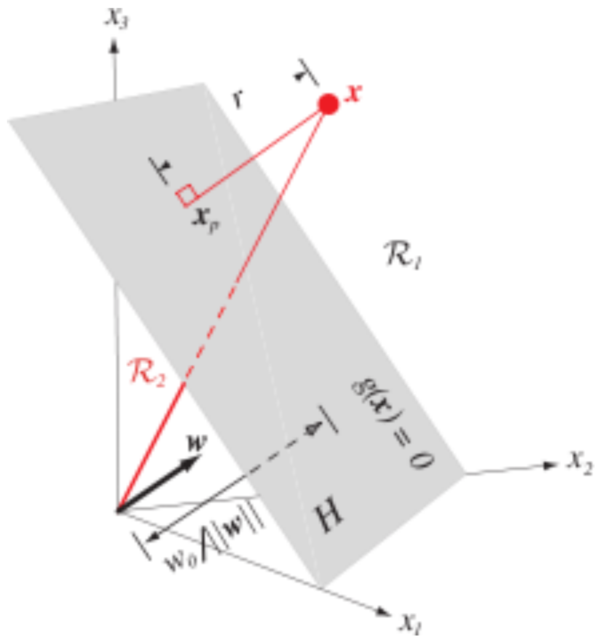


Two category case:
In the two category case, we have two classifiers (salmon and sea bass). We decide whether it belongs to each classifier by taking the discriminant function, and assigning points to ω_1 or ω_2, based upon whether $g(x) > 0$ or $g(x) < 0$. What this actually boils down to is whether $w^t x$ is greater or less than ω_0, or whether the value of the point multiplied by the transpose of the weight vecter is greater or less than the bias. The area where $g(x) = 0$ is the decision surface, creating two regions (R_1 and R_2).

If a point that is located on the decision surface H(where $w^t x = \omega_0$), the point is treated as ambiguous, and not necassarily belonging to either set. If two points are on the decision surface then they both have an equal output ($w^t x_1 + \omega_0 = w^t x_0 + \omega_0$ or, $w^t(x_1 - x_2) = 0$). $g(x)$ also acts as a measure of the distance from the decision surface. X can be expressed as the following function:

$$x = x_p + r(w/||w||)$$

Where x_p is the projection of x onto the hyperplane H, and r is the distance desired (positive for membership in R_1, negative for membership in R_2).

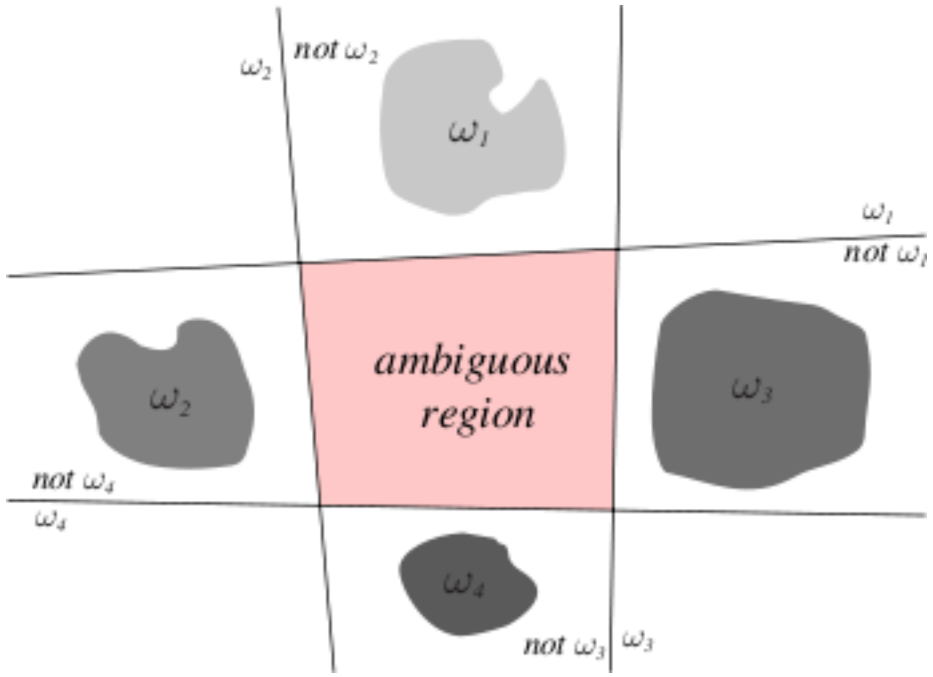


Because x_p is projected on H, $g(x_p) = 0$. As a result, $g(x) = r||w||$, in other words:

$$r = g(x)/||w||$$

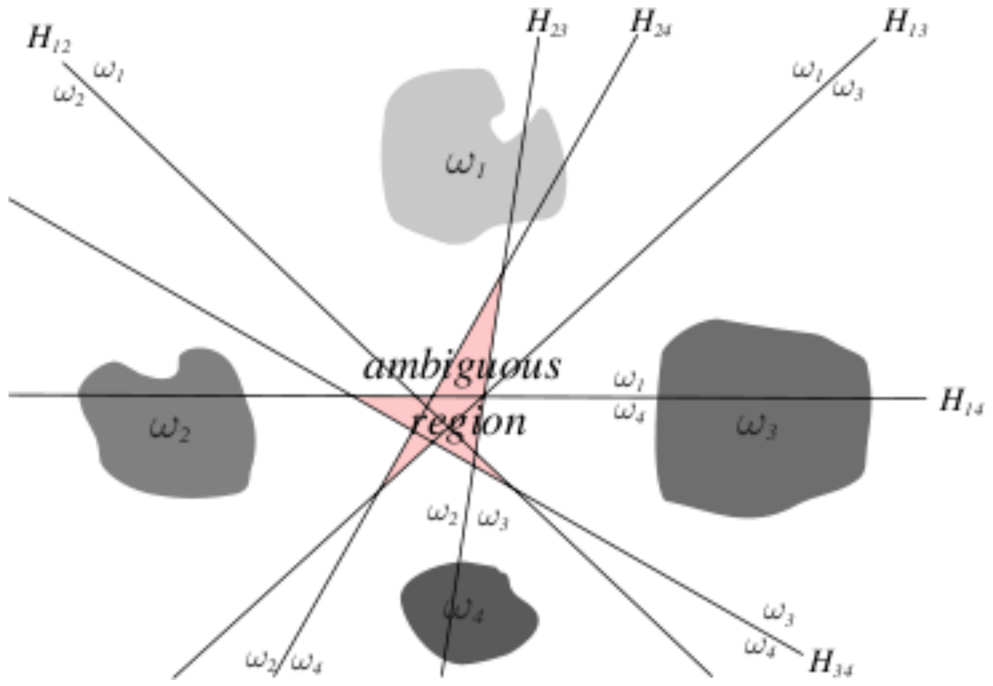
one idea is to split each region R_i into membership in a set. For example, R_i is split it into salmon, trout, and sea bass, with regions for not trout, trout, salmon, not salmon, sea bass, and not seabass, along with an ambigous region.

a classifier H_i separating R_i into ω_i and $!\omega_i$



Another approach is to have split them by pairs, so that when trout borders salmon, we split the region into trout and sea bass, but still have the ambiguous region where all of the boundaries meet.
In this approach, we will have $c(c-1)/2$ classifiers, one for each pair of subspaces so we get

$$H_{ij} = \text{the linear discriminant between } \omega_i \text{and } \omega_j$$



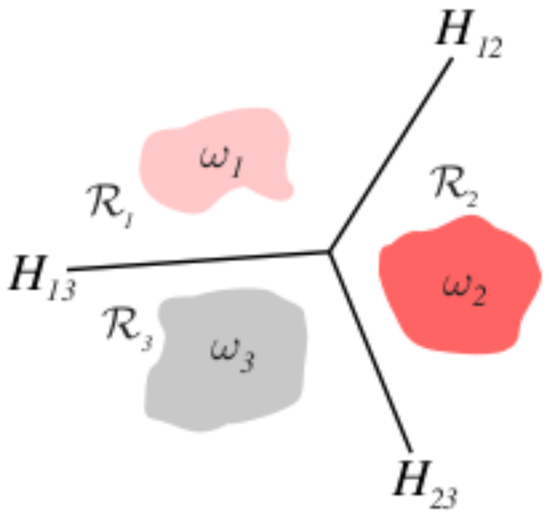
however both approaches do leave ambiguous regions, so we use the linear machine: in this case we use a discriminant function for each classifier, each independent of the other, and each with its own region. This is different from the first approach because now, every point in the region is given membership within a set, there are no more regions of the form $!\omega_i$, which removes that large ambiguous region, but still leaves ambiguity when points are located directly on the decision boundaries. It is different from the second approach because instead of having $c(c-1)/2$ discriminants, we now have c discriminants, each of which can possibly be more complex.

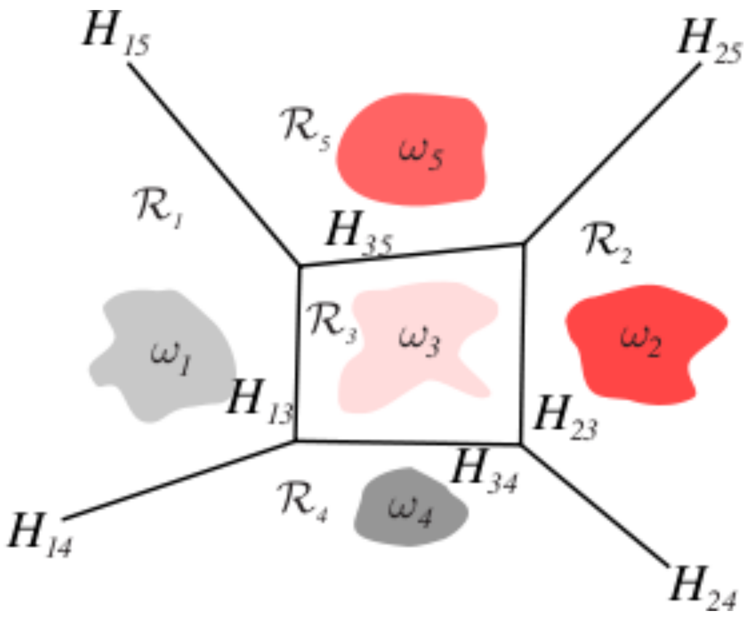
In this version a point is evaluated for each classifier one by one. $g_i(x) = w_i^t x + \omega_{i0}$ $i = 1, ..., c$, $\omega_i = x$ if $g_i(x) > g_j(x)$for all $j \neq i$ the discriminant H_{ij} shows up between two subspaces when $g_i(x) = g_j(x)$ so $0 = g_i(x) - g_j(x)$; using the formula for the space $g_i(x)$, $g_i(x) = w_i^t x + \omega_{i0}$, we get $0 = (w_i - w_j)^t x + (\omega_{i0} - \omega_{j0})$, H_{ij} is normal to $w_i - w_j$ and the distance and direction of x from $H_{ij}$is $(g_i(x) - g_j(x))/||w_i - w_j||$ so:

$$H_{ij} = ((w_i - w_j)^t x + (\omega_{i0} - \omega_{j0}))/||w_i - w_j||$$

because of the normalization, it is the difference between weight vectors, and not the magnitude of any one vector, that becomes important in determining the location of the discriminant.

The decision regions are convex, as was seen in the images, and that leaves it with some definitive restrictions, however. The main restriction is that densities need to have a single area with maximum density (unimodal), instead of having multiple points with high density (multimodal).





Linear Discriminant Function

We can generalize the linear discriminant function as:

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

Where $[w_i]$ is the weight vector, and d is the dimensionality of $\mathbf{x}$ and $\mathbf{w}$.

The sum is what we're used to seeing as $\mathbf{w}^T\mathbf{x}$, and w_0 is the bias term.

The surface defined by $g(\mathbf{x}) = 0$ is a hyperplane.

Quadratic Discriminant Function

We can add a squared term to get the quadratic discriminant function:

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

Note that we now have a **W** matrix, $[w_{ij}]$, in the third term.

Because $x_i x_j = x_j x_i$, we let $w_{ij} = w_{ji}$, so our matrix is symmetric.

The surface defined by $g(\mathbf{x}) = 0$ is hyper-quadric.

Separating Surface

Scale the matrix to get a new matrix:

$$\bar{\mathbf{W}} = \frac{\mathbf{W}}{\mathbf{w}^T \mathbf{W} \mathbf{w}_{-4w_0}}$$

If $\bar{\mathbf{W}} = n\mathbf{I}$ for $n > 0$, then the separating surface is a hypersphere.

If $\bar{\mathbf{W}}$ is positive definite, then the separating surface is a hyperellipsoid.

If $\bar{\mathbf{W}}$ has both positive and negative eigenvalues, then the separating surface is a hyperhyperboloid.

Polynomial Discriminant Function

We can continue to add more terms to the discriminant function to get a polynomial of the nth degree.

For example, the third term would look like: $\sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d} w_{ijk}x_ix_jx_k$

This gets unwieldy fairly quickly, so let's define a vector **y** that consists of $\hat{d}$ functions of **x**, an a general weight vector **a**. Then we have:

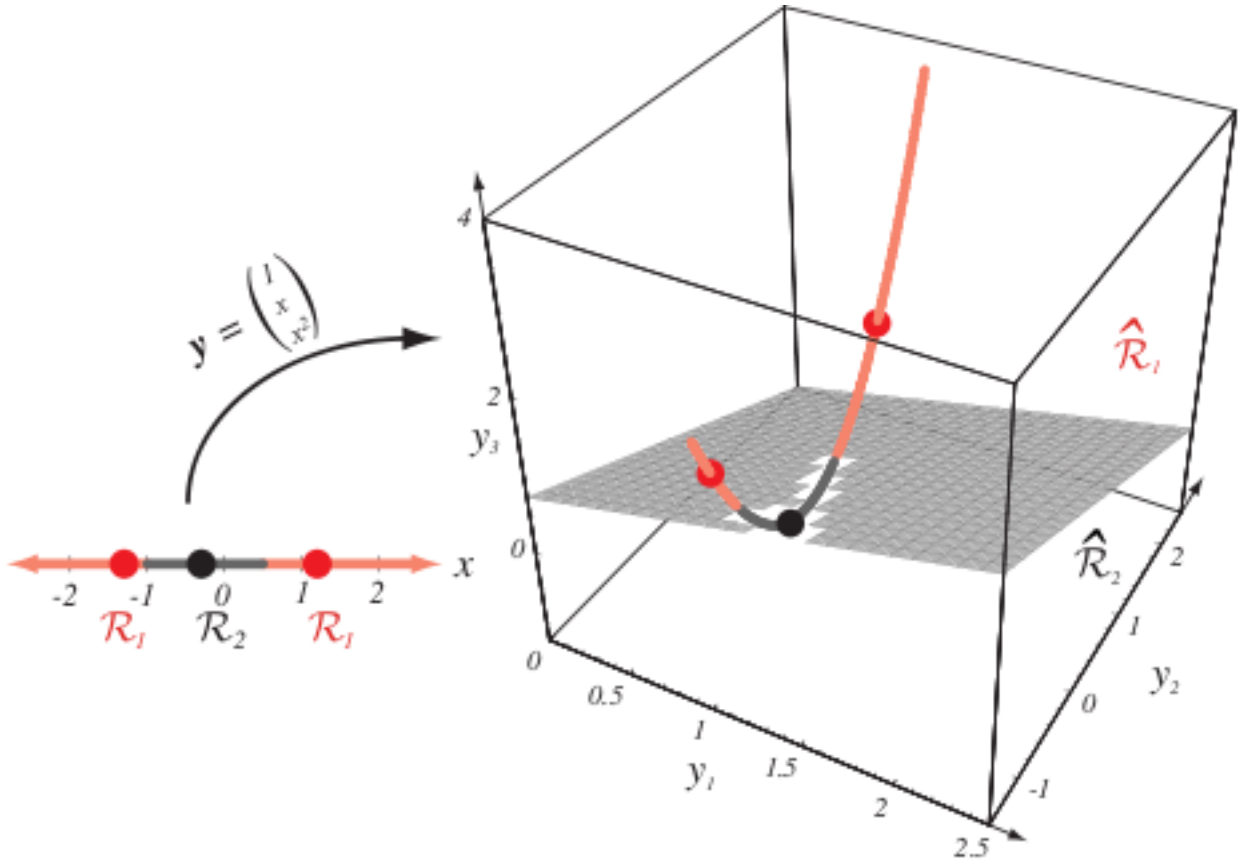
$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\mathbf{x})$$

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

Transformation

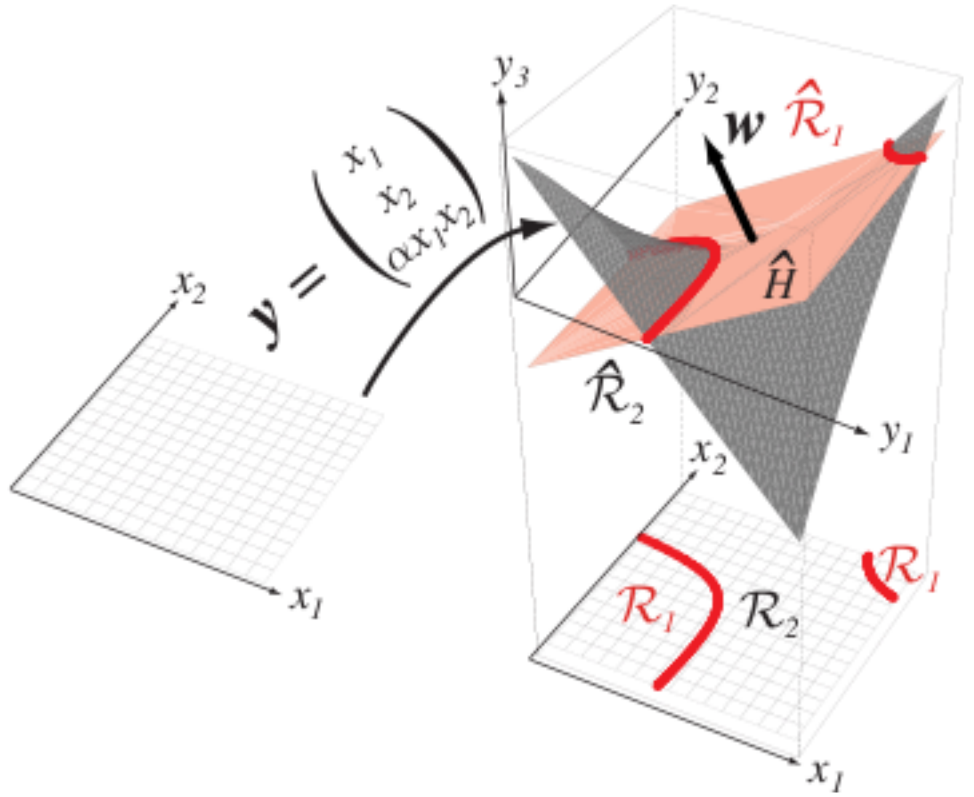
What happened? We performed a transformation from the d-dimensional $\mathbf{x}$-space to $\hat{d}$-dimensional $\mathbf{y}$-space. By picking the $y_i(\mathbf{x})$ functions intelligently, we can thus transform any feature space with painfully complex separating surfaces into a $\mathbf{y}$-space where the separating surface is a hyperplane that passes through the origin.

Example



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

Example


$\mathbf{y} = \begin{pmatrix} x_1 \\ x_2 \\ \alpha x_1 x_2 \end{pmatrix}$
x_2
x_1
y_3
y_2
y_1
$\mathbf{w}$
$\hat{\mathcal{R}}_1$
$\hat{H}$
$\hat{\mathcal{R}}_2$
x_2
$\mathcal{R}_1$
$\mathcal{R}_2$
$\mathcal{R}_1$
x_1

Problems

Take the quadratic example. We have $\frac{d(d+1)}{2}$ terms in **W**, d terms in **w**, and 1 term in the bias, w_0. Adding all these together gives us $\frac{(d+1)(d+2)}{2}$ terms in $\hat{d}$. If we generalize to the kth power, we have $O(d^k)$ terms. Worse, all of the terms in **a** need to be learned from training samples. This is simply too much to deal with.

Linear Discriminant Revisited

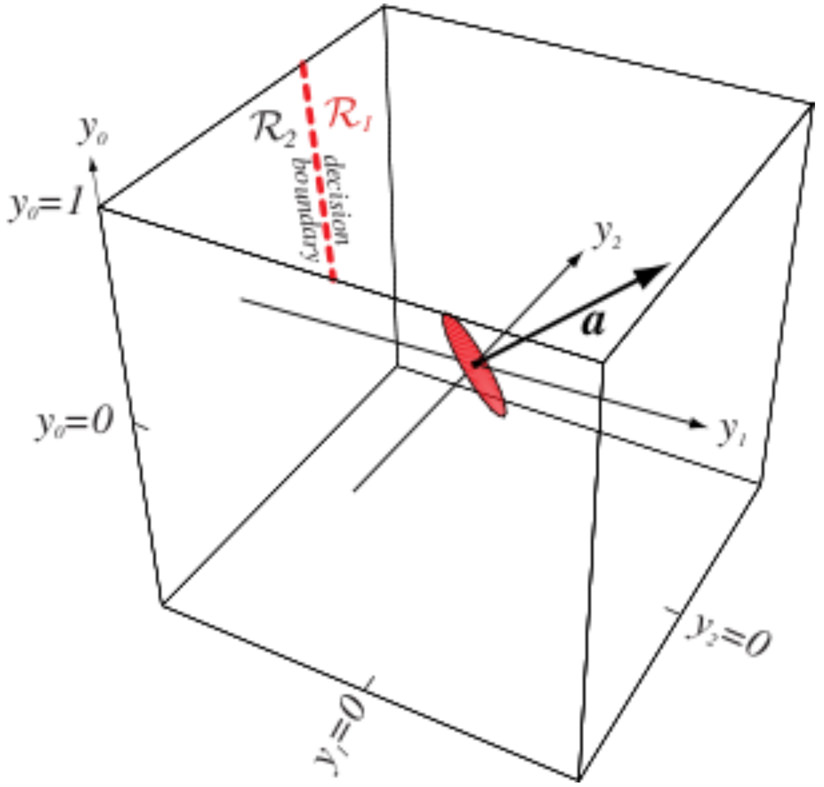
That's not to say this approach is useless. Recall the original linear discriminant function:

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

Let $\mathbf{y} = [\begin{smallmatrix} 1 \\ \mathbf{x} \end{smallmatrix}]$ and $\mathbf{a} = [\begin{smallmatrix} w_0 \\ \mathbf{w} \end{smallmatrix}]$.

We've defined a very simple mapping from d-dimensional $\mathbf{x}$-space to $d+1$-dimensional $\mathbf{y}$-space. But now our decision hyperplane, $\hat{H}$, which could be anywhere in the feature space, passes through the origin. And, in line with section 5.2, the distance from $\mathbf{y}$ to $\hat{H} = |\mathbf{a}^T\mathbf{y}|/||\mathbf{a}|| = g(\mathbf{x})/||\mathbf{a}||$. So we no longer need to find both w_0 and $\mathbf{w}$, but rather a single weight vector $\mathbf{a}$.

Example


y_0
$y_0=1$
$y_0=0$
$\mathcal{R}_2$
$\mathcal{R}_1$
decision boundary
y_2
$\mathbf{a}$
y_1
$y_2=0$
$y_1=0$

Two-Category Linear Separable Case

Suppose we have n samples $\overrightarrow{y_1}, \overrightarrow{y_2}, \ldots, \overrightarrow{y_n}$, where each sample is either labeled $\omega_1$or ω_2. Our goal is to use these samples to find weights in a linear discriminant function.

Let the weights be represented by the vector $\overrightarrow{a}$.

Let the discriminant function be $g(\overrightarrow{x}) = \overrightarrow{a^t} \cdot \overrightarrow{y}$.

Ideally, we want a single weight vector to classify correctly all the samples. If we can obtain the ideal and find one such vector then the samples are *linearly separable.*

Given a sample $\overrightarrow{y_i}$, we know it is classified correctly

if $\overrightarrow{a^t} \cdot \overrightarrow{y_i} > 0$ and $\overrightarrow{y_i}$ is labeled ω_1

or

if $\overrightarrow{a^t} \cdot \overrightarrow{y_i} < 0$ and $\overrightarrow{y_i}$ is labeled ω_2

To "normalize" something, is to make everything "normal". That is, everything in question will be the same or similar in some regard. Here we will normalize the samples by taking the negatives of the ones that have a negative sign. This makes the expression $\overrightarrow{a^t} \cdot \overrightarrow{y_i} > 0$ for all i. Why do this? Having done this we can now ignore the labels and focus more on finding $\overrightarrow{a}$. None of the essential information has been lost by this transformation. The space prior to the transformation is called the *Feature Space* and the one after is now called the *Weight Space.*

Separating Vector a.k.a. Solution Vector

If there exists a vector $\overrightarrow{a}$ such that for all the normalized $\overrightarrow{y_i}$

$\overrightarrow{a^t} \cdot \overrightarrow{y_i} > 0$

then $\overrightarrow{a}$ is the separating vector.

The vector $\overrightarrow{a}$ can be looked as a specified point in weight space. All $\overrightarrow{y_i}$ place constraints on where that point may be.

For all i, $\overrightarrow{a^t} \cdot \overrightarrow{y_i} = 0$ defines a hyperplane through the origin of the Weight Space. By the definition of normal then, each $\overrightarrow{y_i}$ is normal to its corresponding hyperplane.

Now, for all $\overrightarrow{y_i}$ and each corresponding hyperplane, if $\overrightarrow{a}$ exists then it is on the positive side of every hyperplane. Thus we know, if a solution exists, it must lie in the intersection of the n-half spaces. The region in which it can exist is called the *Solution Region.*

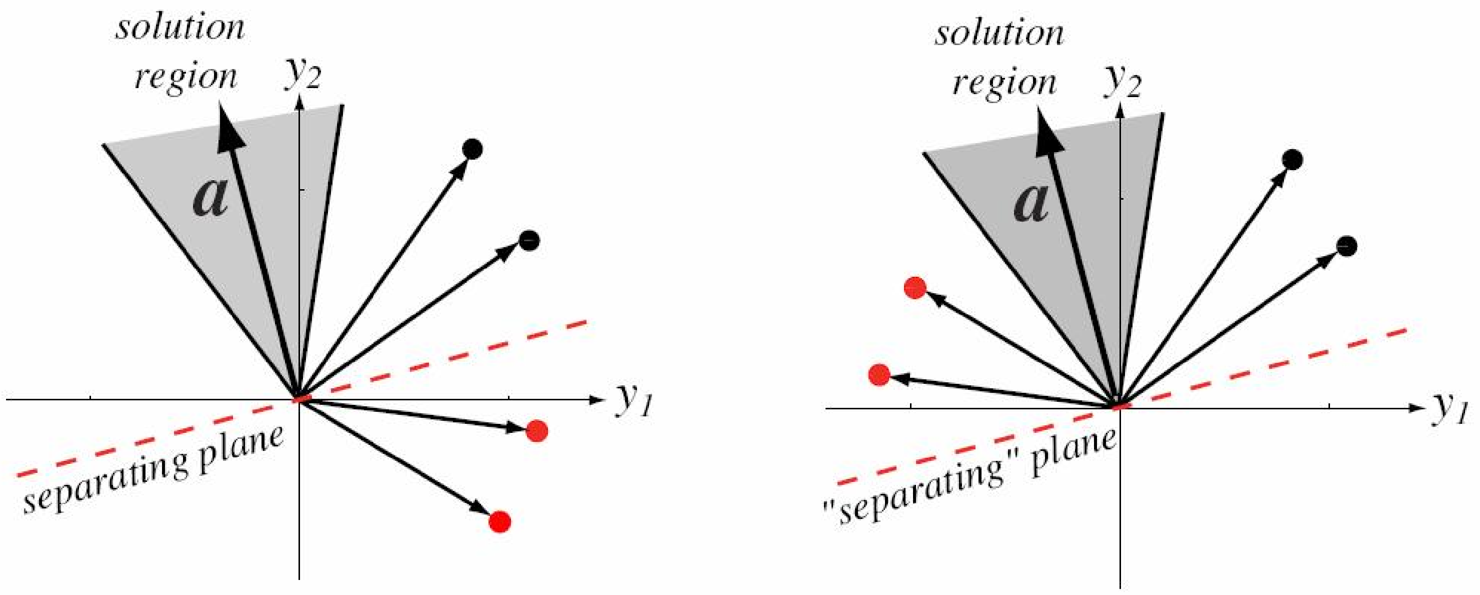


FIGURE 5.8. Four training samples (black for ω_1, red for ω_2) and the solution region in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been "normalized"—that is, changed in sign. Now the solution vector leads to a plane that places all "normalized" points on the same side. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Simplifying finding $\overrightarrow{a}$

Note, the solution vector $\overrightarrow{a}$ is not unique.

Here are two ways to constrain the solution space some when finding the solution vector $\overrightarrow{a}$.

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

2. Given some positive real number b, find the minimum length weight vector satisfying $\overrightarrow{a^t} \cdot \overrightarrow{y_i} > b$. b is called the *margin*. This new region will lie completly inside the old solution region. It is $\frac{b}{\|y_i\|}$ from the old region.

Our motivation for these two methods is to find a solution vector $\overrightarrow{a}$ more towards the middle of the solution region. Our main concern is that we want to prevent an iterative process trying to find $\overrightarrow{a}$ from coming up with a solution on the boundary. We want a solution vector closer to the middle based on the idea that it is more likely new data will be classified correctly.

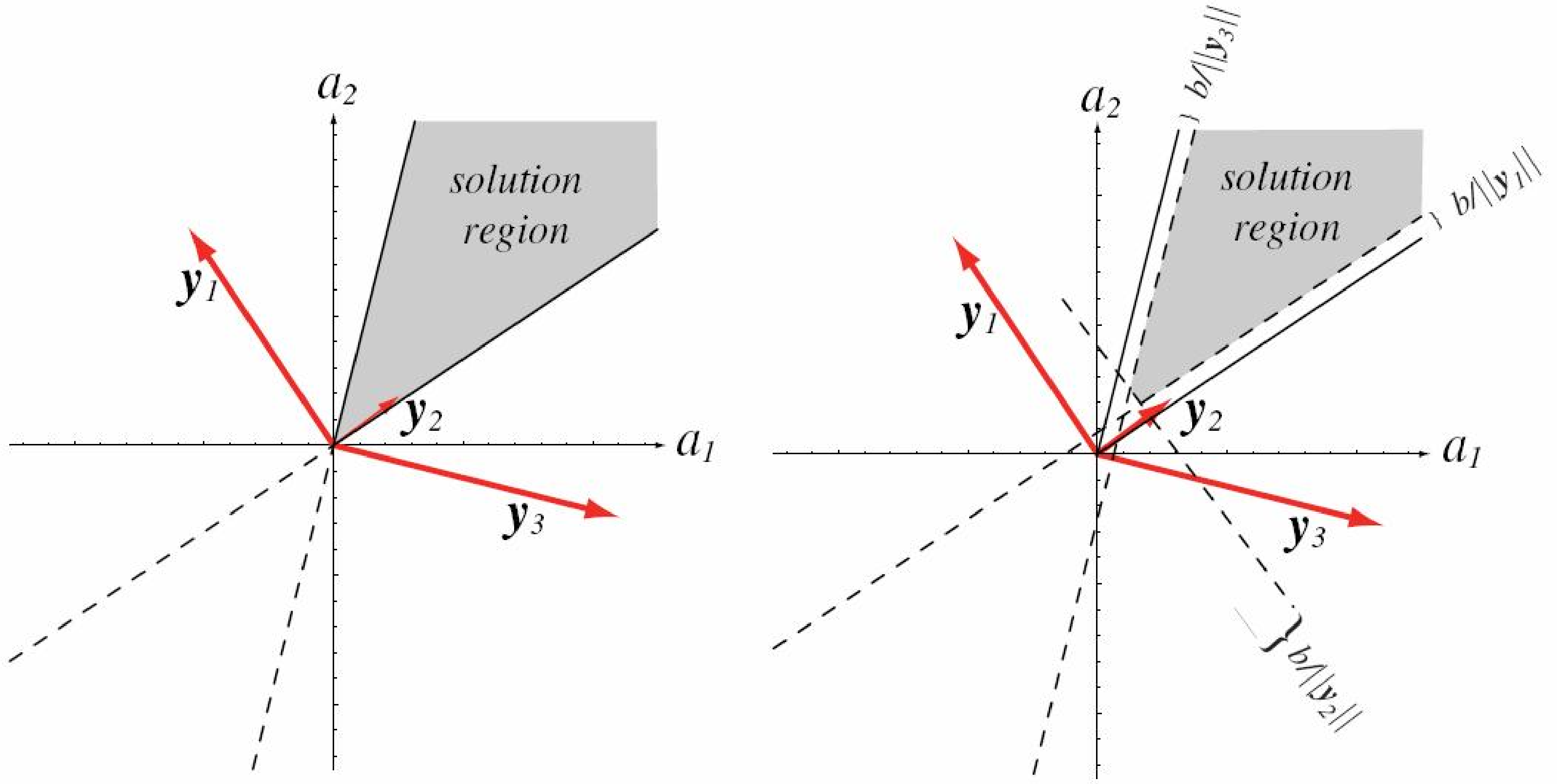


FIGURE 5.9. The effect of the margin on the solution region. At the left is the case of no margin ($b = 0$) equivalent to a case such as shown at the left in Fig. 5.8. At the right is the case $b > 0$, shrinking the solution region by margins $b/\|\mathbf{y}_i\|$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Methods for finding $\overrightarrow{a}$

We now have a set of linear inequality equations $\overrightarrow{a^t} \cdot \overrightarrow{y_i} > 0$ where we want to find a solution vector $\overrightarrow{a}$ that solves them. We will define a criterion function $J(\overrightarrow{a})$ that is minimized if $\overrightarrow{a}$ is a solution vector. This criterion function will be the means by which we find the solution vector.

The mathematical process we will use to minimize the criterion function $J(\overrightarrow{a})$ to find the solution vector $\overrightarrow{a}$ is called gradient descent.

How it works.

Take an arbitrary weight vector $\overrightarrow{a(1)}$. Then take the gradient vector of that arbitrary initial weight vector, $\nabla J(\overrightarrow{a(1)})$.

Then move some small distance in the direction of greatest descent away from $\overrightarrow{a(1)}$ and this will be $\overrightarrow{a(2)}$. The greatest descent is the negative of the gradient by defintion.

For the $(k+1)$ step, $\overrightarrow{a(k+1)}$ is found by:

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

η is a positve real number that is the scaling factor or *learning rate*. It is used to set up the step size. That is how far down the steepest descent from $\overrightarrow{a(k)}$ to $\overrightarrow{a(k+1)}$ we will go.

Algorithm 1. Basic Gradient Descent

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

Setting the Learning Rate η

The biggest problem we face using this method for the purpose of finding the soluton vector is that if poorly selected our learning rate can move too slow or it can over shoot and then even possibly diverge.

To address this concern with the learning rate we will give a method of setting the learning rate.

Start with the assumption that the criterion function $J(\overrightarrow{a})$ can be approximated well by the second order Taylor series expansion around the value of $\overrightarrow{a(k)}$.

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

H is the Hessian matrix of second partial deriviatives about $\overrightarrow{a(k)}$.

Our next step in the derivation for finding a way to compute η efficiently is to substitute $\overrightarrow{a(k+1)}$ into the above equation. This gives

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

Optimal Learning Rate
It follows that $J(\overrightarrow{a(k+1)})$ is minimized when

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

This is the formula for the optimized learning rate η.

Newton's Algorithm
Another method that could be used besides the gradient descent is *Newton's algorithm*. In this we do not solve for $\overrightarrow{a(k+1)}$ by taking a small step down the negative gradient with a learning rate, but instead with the inverse Hessian, and thus the Hessian must be nonsingular. It should be noted though that the inversion is of time $O(d^3)$. A time which can add up quickly and make gradient descent more applicable. Newton's algorithm works well on quadratic error functions because you are minimizing $\overrightarrow{a(k+1)}$ a second order expansion by inserting the following in replace of line 3 in Algorithm 1.

$$\overrightarrow{a(k+1)} = \overrightarrow{a(k)} - H^{-1}\nabla J$$

Algorithm 2. Newton Descent

```
1  begin initialize a, threshold $\theta$
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
```

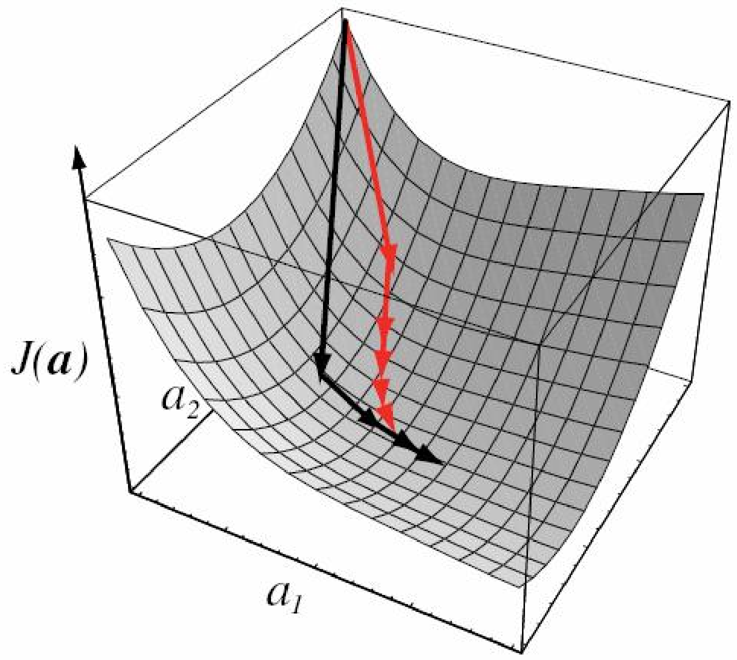


FIGURE 5.10. The sequence of weight vectors given by a simple gradient descent method (red) and by Newton's (second order) algorithm (black). Newton's method typically leads to greater improvement per step, even when using optimal learning rates for both methods. However the added computational burden of inverting the Hessian matrix used in Newton's method is not always justified, and simple gradient descent may suffice. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

References

Duda, Richard, Hart, Peter, and Stork, David. Pattern Classification. New York: John Wily & Sons, 2001.

Russell, Stuart and Norvig, Peter. Artificial Intelligence: A Modern Approach. New Jersey: Pearson Education Inc, 2003.

Larson, Ron, Hostetler, Robert, and Edwards, Bruce. Calculus. Boston: Houghton Mifflin, 2001