5.2 Linear Discriminant Functions and Decision Surfaces

5.2.1 The Two-Category Case

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 in this chapter we shall leave the assignment undefined. Figure 5.1 shows a typical implementation, a clear example of the general structure of a pattern recognition system we saw in Chap. **??**.

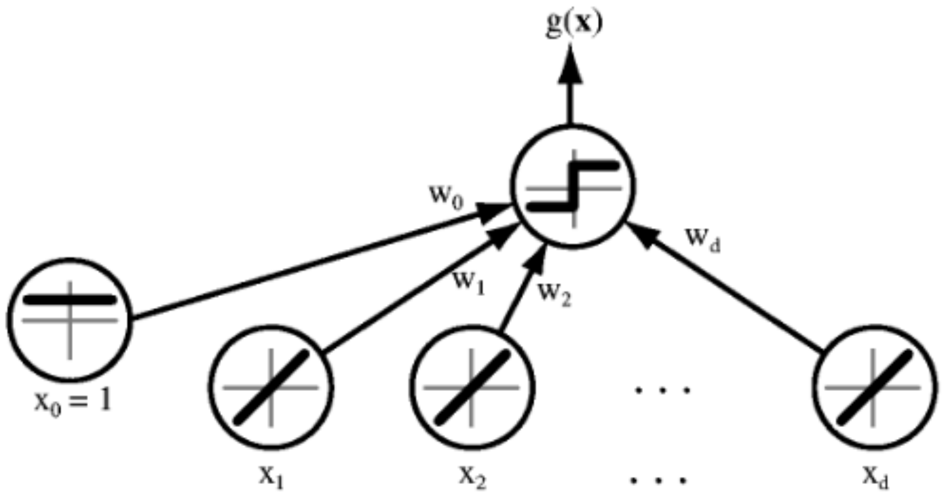

Figure 5.1: A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the output unit sums all these products and emits a $+1$ if $\mathbf{w}^t\mathbf{x} + w_0 > 0$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

The equation $g(\mathbf{x}) = 0$ defines the decision surface that separates points assigned to ω_1 from points assigned to ω_2. When g($\mathbf{x}$) 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

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

and this shows that $\mathbf{w}$ is normal to any vector lying in the hyperplane. In general, the hyperplane H divides the feature space into two halfspaces, decision region $\mathcal{R}_1$ for ω_1 and region $\mathcal{R}_2$ for ω_2. Since $g(\mathbf{x}) > 0$ if $\mathbf{x}$ is in $\mathcal{R}_1$, it follows that the normal vector $\mathbf{w}$ points into $\mathcal{R}_1$. It is sometimes said that any $\mathbf{x}$ in $\mathcal{R}_1$ is on the *positive* side of H, and any $\mathbf{x}$ in $\mathcal{R}_2$ is on the *negative* side.

The discriminant function g($\mathbf{x}$) gives an algebraic measure of the distance from $\mathbf{x}$ to the hyperplane. Perhaps the easiest way to see this is to express $\mathbf{x}$ as

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

where $\mathbf{x}_p$ is the normal projection of $\mathbf{x}$ onto H, and r is the desired algebraic distance — positive if $\mathbf{x}$ is on the positive side and negative if $\mathbf{x}$ is on the negative side. Then, since $g(\mathbf{x}_p) = 0$,

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

or

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

In particular, the distance from the origin to H is given by $w_0/\|\mathbf{w}\|$. If $w_0 > 0$ the origin is on the positive side of H, and if $w_0 < 0$ it is on the negative side. If $w_0 = 0$, then $g(\mathbf{x})$ has the homogeneous form $\mathbf{w}^t\mathbf{x}$, and the hyperplane passes through the origin. A geometric illustration of these algebraic results is given in Fig. 5.2.

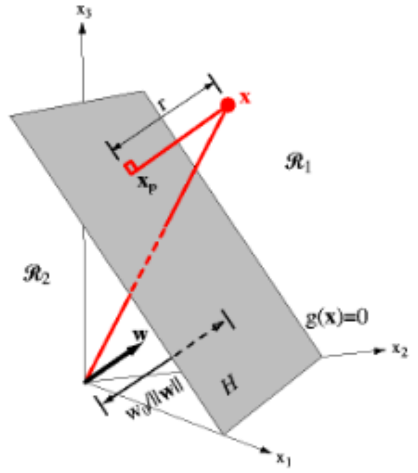

Figure 5.2: The linear decision boundary H, where $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(\mathbf{x}) > 0$) and $\mathcal{R}_2$ (where $g(\mathbf{x}) < 0$).

To summarize, 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 w_0. The discriminant function $g(\mathbf{x})$ is proportional to the signed distance from $\mathbf{x}$ to the hyperplane, with $g(\mathbf{x}) > 0$ when $\mathbf{x}$ is on the positive side, and $g(\mathbf{x}) < 0$ when $\mathbf{x}$ is on the negative side.

of every hyperplane. Thus, a solution vector must lie in the intersection of n half-spaces; indeed any vector in this region is a solution vector. The corresponding region is called the *solution region*, and should not be confused with the decision region in feature space corresponding to any particular category. A two-dimensional example illustrating the solution region for both the normalized and the unnormalized case is shown in Fig. 5.8.

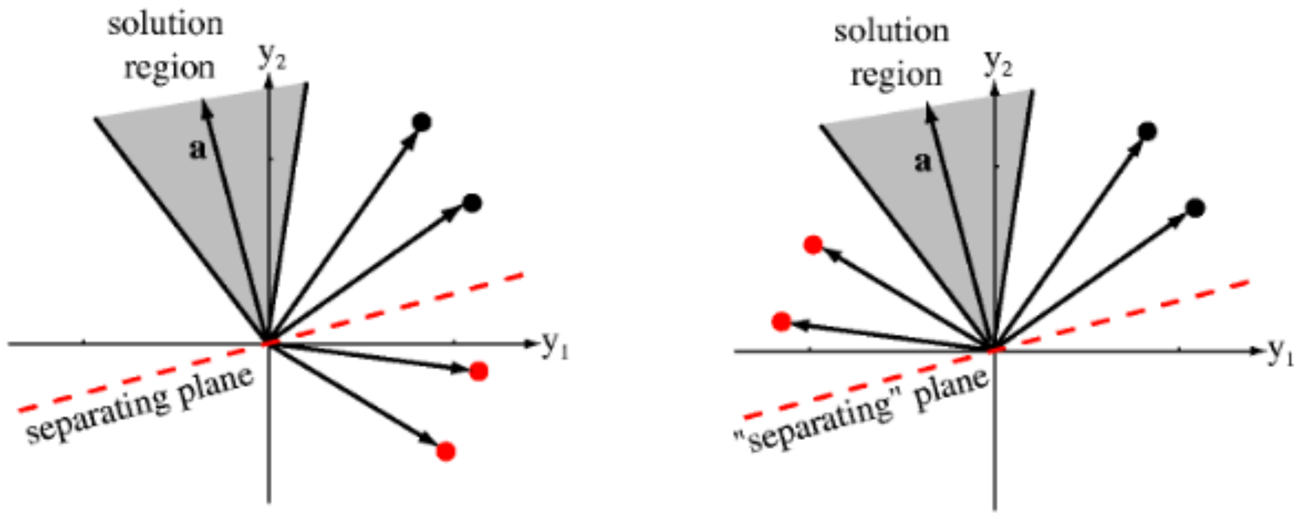

Figure 5.8: Four training samples (black for ω_1, red for ω_2) and the solution region in feature space. The figure on the left shows the raw data; the solution vectors leads to a plane that separates the patterns from the two categories. In the figure on the right, the red points have been "normalized"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

From this discussion, it should be clear that the solution vector — again, if it exists — is not unique. There are several ways to impose additional requirements to constrain the solution vector. One possibility is to seek a unit-length weight vector that maximizes the minimum distance from the samples to the separating plane. Another possibility is to seek the minimum-length weight vector satisfying $\mathbf{a}^t\mathbf{y}_i \geq b$ for all i, where b is a positive constant called the *margin*. As shown in Fig. 5.9, the solution region resulting form the intersections of the halfspaces for which $\mathbf{a}^t\mathbf{y}_i \geq b > 0$ lies within the previous solution region, being insultated from the old boundaries by the distance $b/\|\mathbf{y}_i\|$.

The motivation behind these attempts to find a solution vector closer to the "middle" of the solution region is the natural belief that the resulting solution is more likely to classify new test samples correctly. In most of the cases we shall treat, however, we shall be satisfied with any solution strictly within the solution region. Our chief concern will be to see that any iterative procedure used does not converge to a limit point on the boundary. This problem can always be avoided by the introduction of a margin, i.e., by requiring that $\mathbf{a}^t\mathbf{y}_i \geq b > 0$ for all i.

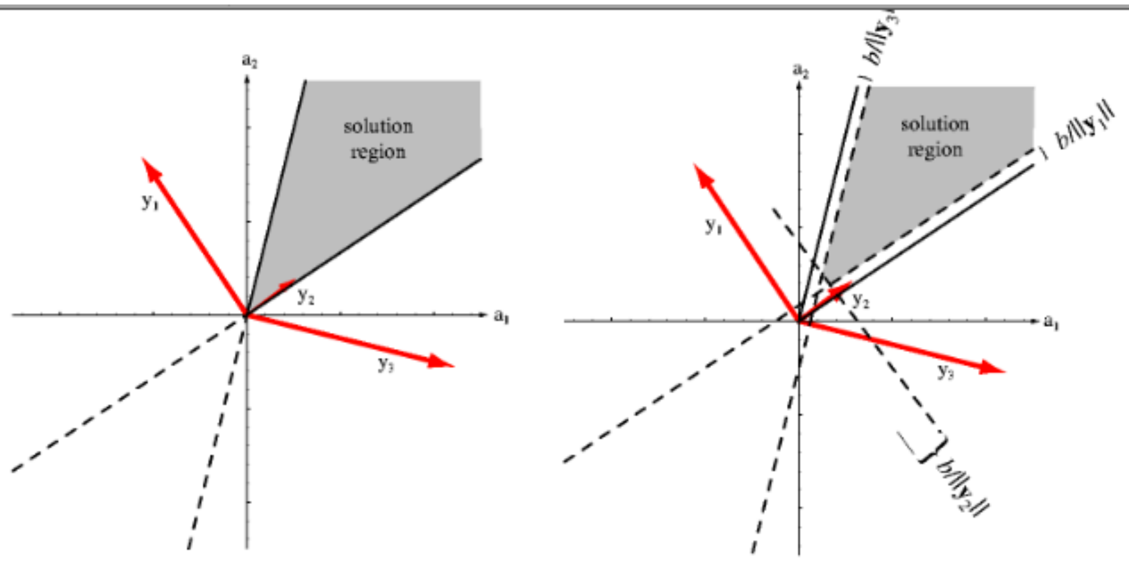

Figure 5.9: The effect of the margin on the solution region. At the left, the case of no margin ($b = 0$) equivalent to a case such as shown at the left in Fig. 5.8. At the right is the case $b > 0$, shrinking the solution region by margins $b/\|\mathbf{y}_i\|$.

distance from $\mathbf{a}(1)$ in the direction of steepest descent, i.e., along the negative of the gradient. In general, $\mathbf{a}(k+1)$ is obtained from $\mathbf{a}(k)$ by the equation

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k)), \tag{12}$$

where η is a positive scale factor or *learning rate* that sets the step size. We hope that such a sequence of weight vectors will converge to a solution minimizing $J(\mathbf{a})$. In algorithmic form we have:

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
```

The many problems associated with gradient descent procedures are well known. Fortunately, we shall be constructing the functions we want to minimize, and shall be able to avoid the most serious of these problems. One that will confront us repeatedly, however, is the choice of the learning rate $\eta(k)$. If $\eta(k)$ is too small, convergence is needlessly slow, whereas if $\eta(k)$ is too large, the correction process will overshoot and can even diverge (Sect. 5.6.1).

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

$$J(\mathbf{a}) \simeq J(\mathbf{a}(k)) + \nabla J^t(\mathbf{a} - \mathbf{a}(k)) + \frac{1}{2}(\mathbf{a} - \mathbf{a}(k))^t\mathbf{H}\ (\mathbf{a} - \mathbf{a}(k)), \tag{13}$$

where $\mathbf{H}$ is the *Hessian matrix* of second partial derivatives $\partial^2 J/\partial a_i \partial a_j$ evaluated at $\mathbf{a}(k)$. Then, substituting $\mathbf{a}(k+1)$ from Eq. 12 into Eq. 13 we find:

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

From this it follows (Problem 12) that $J(\mathbf{a}(k+1))$ can be minimized by the choice

$$\eta(k) = \frac{\|\boldsymbol{\nabla} J\|^2}{\boldsymbol{\nabla} J^t \mathbf{H} \boldsymbol{\nabla} J}, \tag{14}$$

where $\mathbf{H}$ depends on $\mathbf{a}$, and thus indirectly on k. This then is the optimal choice of $\eta(k)$ given the assumptions mentioned. Note that if the criterion function $J(\mathbf{a})$ is quadratic throughout the region of interest, then $\mathbf{H}$ is constant and η is a constant independent of k.

An alternative approach, obtained by ignoring Eq. 12 and by choosing $\mathbf{a}(k+1)$ to minimize the second-order expansion, is *Newton's algorithm* where line 3 in Algorithm 1 is replaced by

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \mathbf{H}^{-1}\boldsymbol{\nabla} J, \tag{15}$$

leading to the following algorithm:

Algorithm 2 (Newton descent)

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

Simple gradient descent and Newton's algorithm are compared in Fig. 5.10.

Generally speaking, Newton's algorithm will usually give a greater improvement *per step* than the simple gradient descent algorithm, even with the optimal value of $\eta(k)$. However, Newton's algorithm is not applicable if the Hessian matrix $\mathbf{H}$ is singular. Furthermore, even when $\mathbf{H}$ is nonsingular, the $O(d^3)$ time required for matrix inversion on each iteration can easily offset the descent advantage. In fact, it often takes less time to set $\eta(k)$ to a constant η that is smaller than necessary and make a few more corrections than it is to compute the optimal $\eta(k)$ at each step (Computer exercise 1).

5.5 Minimizing the Perceptron Criterion Function

5.5.1 The Perceptron Criterion Function

Consider now the problem of constructing a criterion function for solving the linear inequalities $\mathbf{a}^t\mathbf{y}_i > 0$. The most obvious choice is to let $J(\mathbf{a};\ \mathbf{y}_1, ..., \mathbf{y}_n)$ 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*

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} (-\mathbf{a}^t\mathbf{y}), \tag{16}$$

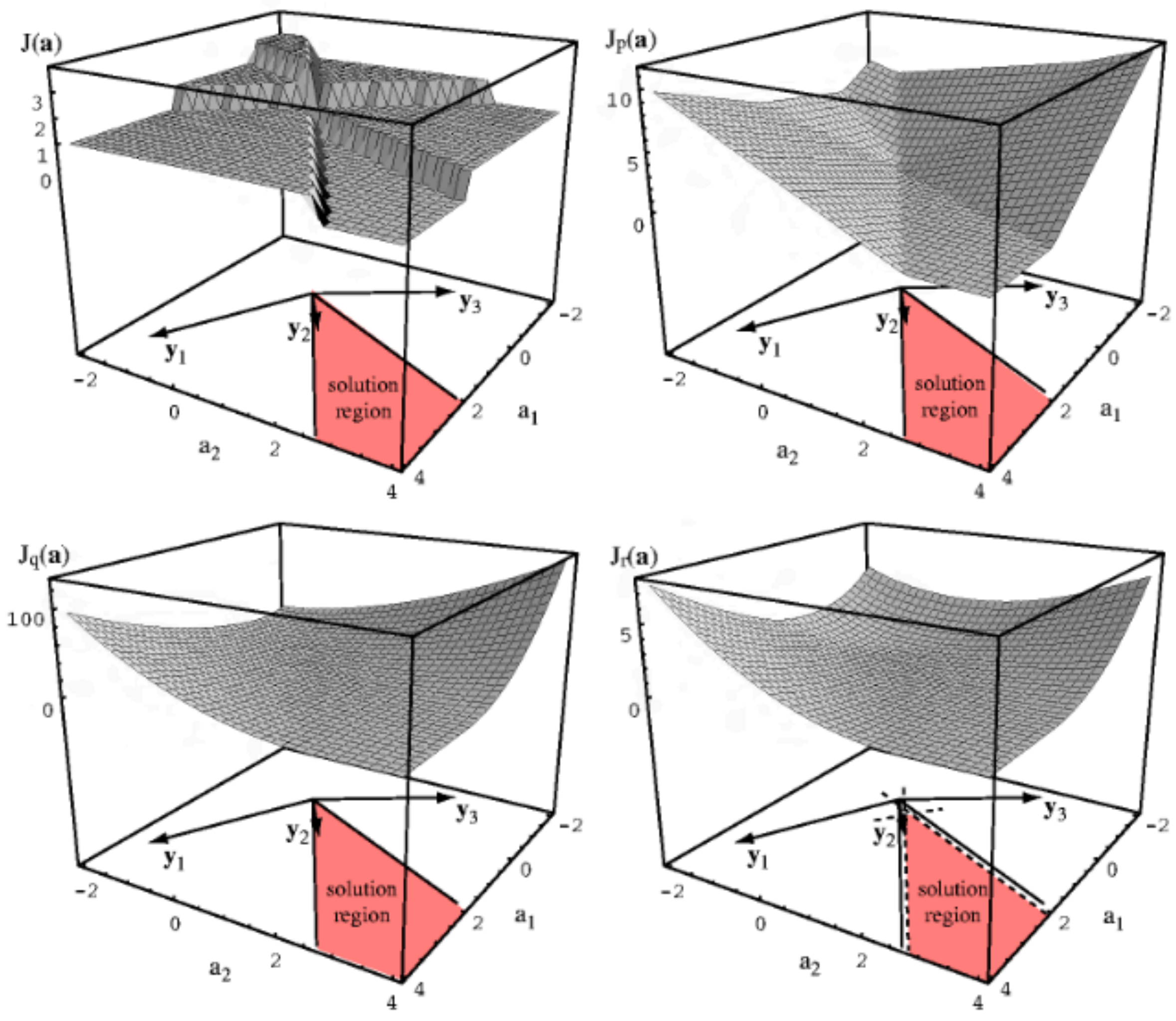

Figure 5.11: Four learning criteria as a function of weights in a linear classifier. At the upper left is the total number of patterns misclassified, which is piecewise constant and hence unacceptable for gradient descent procedures. At the upper right is the Perceptron criterion (Eq. 16), which is piecewise linear and acceptable for gradient descent. The lower left is squared error (Eq. 32), 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 (Eq. 33). 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.

Thus, the batch Perceptron algorithm for finding a solution vector can be stated very simply: the next weight vector is obtained by adding some multiple of the sum of the misclassified samples to the present weight vector. We use the term "batch" to refer to the fact that (in general) a large group of samples is used when computing each weight update. (We shall soon see alternate methods based on single samples.) Figure 5.12 shows how this algorithm yields a solution vector for a simple two-dimensional example with $\mathbf{a}(1) = \mathbf{0}$, and $\eta(k) = 1$. We shall now show that it will yield a solution for any linearly separable problem.

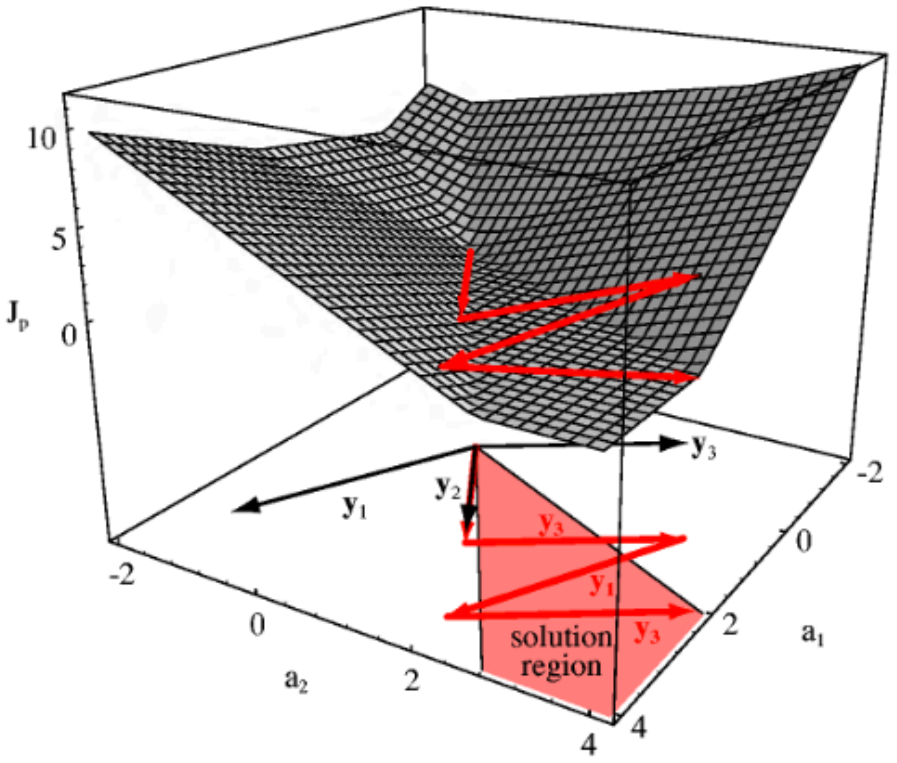

Figure 5.12: The Perceptron criterion, J_p is plotted as a function of the weights a_1 and a_2 for a three-pattern problem. The weight vector begins at $\mathbf{0}$, and the algorithm sequentially adds to it vectors equal to the "normalized" misclassified patterns themselves. In the example shown, this 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.)

5.5.2 Convergence Proof for Single-Sample Correction

We shall begin our examination of convergence properties of the Perceptron algorithm with a variant that is easier to analyze. Rather than testing $\mathbf{a}(k)$ on all of the samples and basing our correction of the set $\mathcal{Y}_k$ of misclassified training samples, we shall consider the samples in a sequence and shall modify the weight vector whenever it misclassifies a *single* sample. For the purposes of the convergence proof, the detailed nature of the sequence is unimportant as long as every sample appears in the sequence infinitely often. The simplest way to assure this is to repeat the samples cyclically, though from a practical point of view random selection is often to be preferred (Sec. 5.8.5). Clearly neither the batch nor this single-sample version of the Perceptron algorithm are on-line since we must store and potentially revisit all of the training patterns.

Two further simplifications help to clarify the exposition. First, we shall temporarily restrict our attention to the case in which $\eta(k)$ is constant — the so-called *fixed-increment* case. It is clear from Eq. 18 that if $\eta(t)$ is constant it merely serves to scale the samples; thus, in the fixed-increment case we can take $\eta(t) = 1$ with no loss in generality. The second simplification merely involves notation. When the samples are considered sequentially, some will be misclassified. Since we shall only change the weight vector when there is an error, we really need only pay attention to the misclassified samples. Thus we shall denote the sequence of samples using superscripts, i.e., by $\mathbf{y}^1,\ \mathbf{y}^2, ...,\ \mathbf{y}^k, ...$, where each $\mathbf{y}^k$ is one of the n samples $\mathbf{y}_1, ..., \mathbf{y}_n$, and where each $\mathbf{y}^k$ is misclassified. For example, if the samples $\mathbf{y}_1$, $\mathbf{y}_2$, and $\mathbf{y}_3$ are considered cyclically, and if the marked samples

$$\overset{\downarrow}{\mathbf{y}_1},\ \mathbf{y}_2,\ \overset{\downarrow}{\mathbf{y}_3},\ \overset{\downarrow}{\mathbf{y}_1},\ \overset{\downarrow}{\mathbf{y}_2},\ \mathbf{y}_3,\ \mathbf{y}_1,\ \overset{\downarrow}{\mathbf{y}_2},\ ... \tag{19}$$

are misclassified, then the sequence $\mathbf{y}^1,\ \mathbf{y}^2,\ \mathbf{y}^3,\ \mathbf{y}^4,\ \mathbf{y}^5, ...$ denotes the sequence $\mathbf{y}_1,\ \mathbf{y}_3,\ \mathbf{y}_1,\ \mathbf{y}_2,\ \mathbf{y}_2, ...$ With this understanding, the *fixed-increment rule* for generating a sequence of weight vectors can be written as

$$\left.\begin{array}{ll} \mathbf{a}(1) & \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \mathbf{y}^k & k \geq 1 \end{array}\right\} \tag{20}$$

where $\mathbf{a}^t(k)\mathbf{y}^k \leq 0$ for all k. If we let n denote the total number of patterns, the algorithm is:

Algorithm 4 (Fixed-increment single-sample Perceptron)

```
1 begin initialize  $\mathbf{a}, k = 0$
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
```

The fixed-increment Perceptron rule is the simplest of many algorithms that have been proposed for solving systems of linear inequalities. Geometrically, its interpretation in weight space is particularly clear. Since $\mathbf{a}(k)$ misclassifies $\mathbf{y}^k$, $\mathbf{a}(k)$ is not on the positive side of the $\mathbf{y}^k$ hyperplane $\mathbf{a}^t\mathbf{y}^k = 0$. The addition of $\mathbf{y}^k$ to $\mathbf{a}(k)$ moves the weight vector directly toward and perhaps across this hyperplane. Whether the hyperplane is crossed or not, the new inner product $\mathbf{a}^t(k+1)\mathbf{y}^k$ is larger than the old inner product $\mathbf{a}^t(k)\mathbf{y}^k$ by the amount $\|\mathbf{y}^k\|^2$, and the correction is clearly moving the weight vector in a good direction (Fig. 5.13).

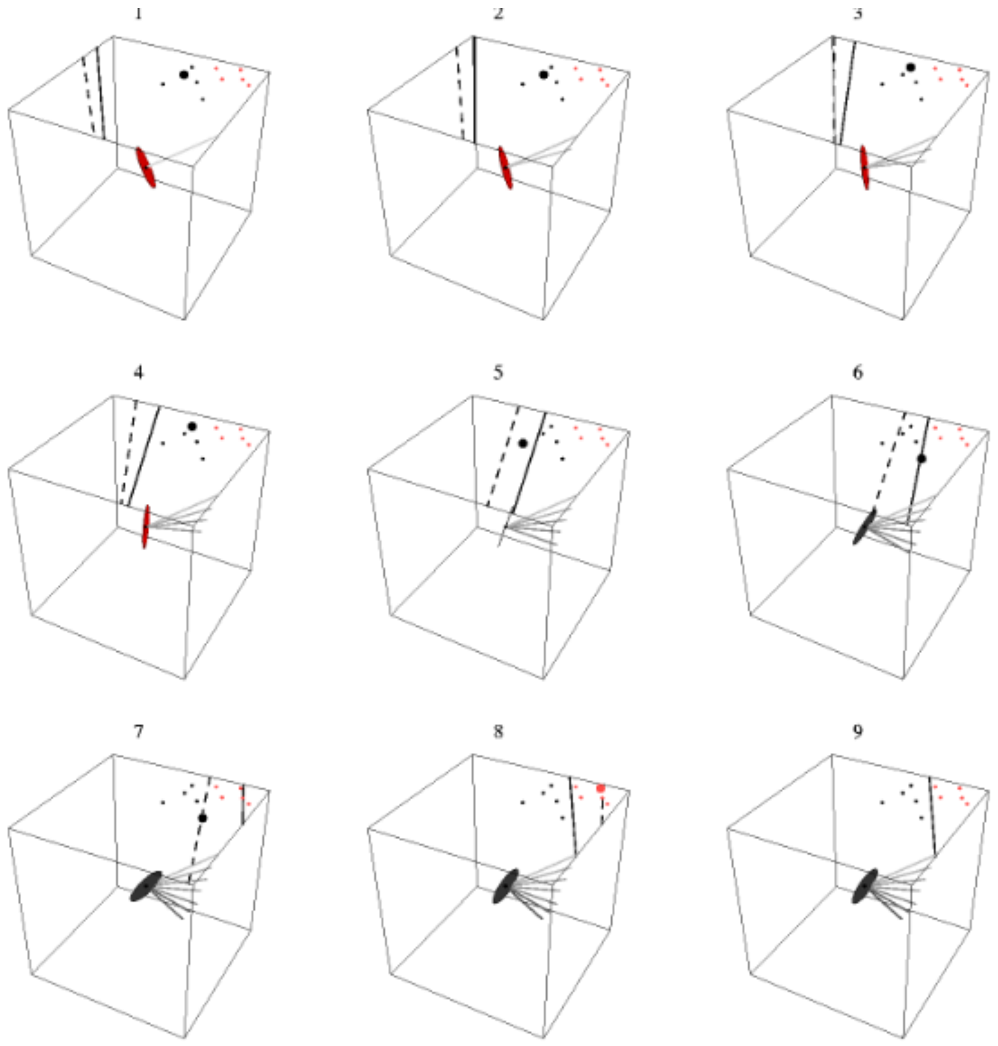

Figure 5.13: Samples from two categories, ω_1 (black) and ω_2 (red) are shown in augmented feature space, along with an augmented weight vector $\mathbf{a}$. At each step in a fixed-increment rule, one of the misclassified patterns, $\mathbf{y}^k$, is shown by the large dot. A correction $\Delta\mathbf{a}$ (proportional to the pattern vector $\mathbf{y}^k$) is added to the weight vector — towards an ω_1 point or away from an ω_2 point. This changes the decision boundary from the dashed position (from the previous update) to the solid position. The sequence of resulting $\mathbf{a}$ vectors is shown, where later values are shown darker. In this example, by step 9 a solution vector has been found and the categories successfully separated by the decision boundary shown.

Clearly this algorithm can only terminate if the samples are linearly separable; we now prove that indeed it terminates so long as the samples are linearly separable.

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:

In seeking a proof, it is natural to try to show that each correction brings the weight vector closer to the solution region. That is, one might try to show that if $\hat{\mathbf{a}}$ is any solution vector, then $\|\mathbf{a}(k+1) - \hat{\mathbf{a}}\|$ is smaller than $\|\mathbf{a}(k) - \hat{\mathbf{a}}\|$. While this turns out not to be true in general (cf. steps 6 & 7 in Fig. 5.13), we shall see that it is true for solution vectors that are sufficiently long.

Let $\hat{\mathbf{a}}$ be any solution vector, so that $\hat{\mathbf{a}}^t\mathbf{y}_i$ is strictly positive for all i, and let α be a positive scale factor. From Eq. 20,

$$\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}} = (\mathbf{a}(k) - \alpha\hat{\mathbf{a}}) + \mathbf{y}^k$$

and hence

$$\|\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}}\|^2 = \|\mathbf{a}(k) - \alpha\hat{\mathbf{a}}\|^2 + 2(\mathbf{a}(k) - \alpha\hat{\mathbf{a}})^t\mathbf{y}^k + \|\mathbf{y}^k\|^2.$$

Since $\mathbf{y}^k$ was misclassified, $\mathbf{a}^t(k)\mathbf{y}^k \leq 0$, and thus

$$\|\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}}\|^2 \leq \|\mathbf{a}(k) - \alpha\hat{\mathbf{a}}\|^2 - 2\alpha\hat{\mathbf{a}}^t\mathbf{y}^k + \|\mathbf{y}^k\|^2.$$

Because $\hat{\mathbf{a}}^t\mathbf{y}^k$ is strictly positive, the second term will dominate the third if α is sufficiently large. In particular, if we let β be the maximum length of a pattern vector,

$$\beta^2 = \max_i \|\mathbf{y}_i\|^2, \tag{21}$$

and γ be the smallest inner product of the solution vector with any pattern vector, i.e.,

$$\gamma = \min_i \left[\hat{\mathbf{a}}^t\mathbf{y}_i\right] > 0, \tag{22}$$

then we have the inequality

$$\|\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}}\|^2 \leq \|\mathbf{a}(k) - \alpha\hat{\mathbf{a}}\|^2 - 2\alpha\gamma + \beta^2.$$

If we choose

$$\alpha = \frac{\beta^2}{\gamma}, \tag{23}$$

we obtain

$$\|\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}}\|^2 \leq \|\mathbf{a}(k) - \alpha\hat{\mathbf{a}}\|^2 - \beta^2.$$

Thus, the squared distance from $\mathbf{a}(k)$ to $\alpha\hat{\mathbf{a}}$ is reduced by at least β^2 at each correction, and after k corrections

$$\|\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}}\|^2 \leq \|\mathbf{a}(k) - \alpha\hat{\mathbf{a}}\|^2 - k\beta^2. \tag{24}$$

Since the squared distance cannot become negative, it follows that the sequence of corrections must terminate after no more than k_0 corrections, where

$$k_0 = \frac{\|\mathbf{a}(1) - \alpha\hat{\mathbf{a}}\|^2}{\beta^2}. \tag{25}$$

Since a correction occurs whenever a sample is misclassified, and since each sample appears infinitely often in the sequence, it follows that when corrections cease the resulting weight vector must classify all of the samples correctly. ∎

The number k_0 gives us a bound on the number of corrections. If $\mathbf{a}(1) = \mathbf{0}$, we get the following particularly simple expression for k_0:

$$k_0 = \frac{\alpha^2 \|\hat{\mathbf{a}}\|^2}{\beta^2} = \frac{\beta^2 \alpha^2 \|\hat{\mathbf{a}}\|^2}{\gamma^2} = \frac{\max\limits_i \|\mathbf{y}_i\|^2 \|\hat{\mathbf{a}}\|^2}{\min\limits_i [\mathbf{y}_i^t \hat{\mathbf{a}}]^2}. \tag{26}$$

The denominator in Eq. 26 shows that the difficulty of the problem is essentially determined by the samples most nearly orthogonal to the solution vector. Unfortunately, it provides no help when we face an unsolved problem, since the bound is expressed in terms of a solution vector which is unknown. In general, it is clear that linearly-separable problems can be made arbitrarily difficult to solve by making the samples almost coplanar (Computer exercise 2). Nevertheless, if the training samples are linearly separable, the fixed-increment rule will yield a solution after a finite number of corrections.

5.5.3 Some Direct Generalizations

The fixed increment rule can be generalized to provide a variety of related algorithms. We shall briefly consider two variants of particular interest. The first variant introduces a *variable increment* $\eta(k)$ and a margin b, and calls for a correction whenever $\mathbf{a}^t(k)\mathbf{y}^k$ fails to excede the margin. The update is given by

$$\left.\begin{array}{ll} \mathbf{a}(1) & \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{y}^k & k \geq 1, \end{array}\right\} \tag{27}$$

where now $\mathbf{a}^t(k)\mathbf{y}^k \leq b$ for all k. Thus for n patterns, our algorithm is:

Algorithm 5 (Variable increment Perceptron with margin)

```
1 begin initialize  $\mathbf{a}$, 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  $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\mathbf{y}_k$
4          until $\mathbf{a}^t\mathbf{y}_k + b \leq 0$ for all $k$
5    return $\mathbf{a}$
6 end
```

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

$$\eta(k) \geq 0, \tag{28}$$

$$\lim_{m \to \infty} \sum_{k=1}^{m} \eta(k) = \infty \tag{29}$$

and

$$\lim_{m\to\infty} \frac{\sum_{k=1}^{m} \eta^2(k)}{\left(\sum_{k=1}^{m} \eta(k)\right)^2} = 0, \tag{30}$$

then $\mathbf{a}(k)$ converges to a solution vector $\mathbf{a}$ satisfying $\mathbf{a}^t\mathbf{y}_i > b$ for all i (Problem 18). In particular, these conditions on $\eta(k)$ are satisfied if $\eta(k)$ is a positive constant, or if it decreases like $1/k$.

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

$$\left.\begin{array}{ll} \mathbf{a}(1) & \text{arbitrary} \\ \mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k) \sum\limits_{\mathbf{y}\in\mathcal{Y}_k} \mathbf{y}, \end{array}\right\} \tag{31}$$

where $\mathcal{Y}_k$ is the set of training samples misclassified by $\mathbf{a}(k)$. It is easy to see that this algorithm will also yield a solution once one recognizes that if $\hat{\mathbf{a}}$ is a solution vector for $\mathbf{y}_1, ..., \mathbf{y}_n$, then it correctly classifies the correction vector

$$\mathbf{y}^k = \sum_{\mathbf{y}\in\mathcal{Y}_k} \mathbf{y}.$$

In greater detail, then, the algorithm is

Algorithm 6 (Batch variable increment Perceptron)

```
begin initialize $\mathbf{a}, \eta(\cdot), k = 0$
          do $k \leftarrow k + 1$
             $\mathcal{Y}_k = \{\}$
             $j = 0$
             do $j \leftarrow j + 1$
                if $\mathbf{y}_j$ is misclassified then Append $\mathbf{y}_j$ to $\mathcal{Y}_k$
             until $j = n$
             $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y}\in\mathcal{Y}_k} \mathbf{y}$
          until $\mathcal{Y}_k = \{\}$
return $\mathbf{a}$
end
```

The benefit of batch gradient descent is that the trajectory of the weight vector is smoothed, compared to that in corresponding single-sample algorithms (e.g., Algorithm 5), since at each update the full set of misclassified patterns is used — the local statistical variations in the misclassified patterns tend to cancel while the large-scale trend does not. Thus, if the samples are linearly separable, all of the possible correction vectors form a linearly separable set, and if $\eta(k)$ satisfies Eqs. 28–30, the sequence of weight vectors produced by the gradient descent algorithm for $J_p(\cdot)$ will always converge to a solution vector.

It is interesting to note that the conditions on $\eta(k)$ are satisfied if $\eta(k)$ is a positive constant, if it decreases as $1/k$, or even if it increases as k. Generally speaking, one would prefer to have $\eta(k)$ become smaller as time goes on. This is particularly true if there is reason to believe that the set of samples is not linearly separable, since it reduces the disruptive effects of a few "bad" samples. However, in the separable case it is a curious fact that one can allow $\eta(k)$ to become larger and still obtain a solution.