02.Classification

We now discuss an analogous class of models for solving classification problems. The goal in classification is to take an input vector x and to assign it to one of K discrete classes C_k where k = 1,...,K. In the most common scenario, the classes are taken to be disjoint, so that each input is assigned to one and only one class. The input space is thereby divided into decision regions whose boundaries are called decision boundaries or decision surfaces.
Data sets whose classes can be separated exactly by linear decision surfaces are said to be linearly separable.
For probabilistic models, the
most convenient, in the case of two-class problems, is the binary representation in which there is a single target variable t ∈ {0, 1} such that t = 1 represents class C_1 and t = 0 represents class C_2. We can interpret the value of t as the probability that the class is C_1, with the values of probability taking only the extreme values of 0 and 1. For K > 2 classes, it is convenient to use a 1-of-K coding scheme in which t is a vector of length K such that if the class is C_j , then all elements t_k of t are zero except element t_j , which takes the value 1. For instance, if we have K = 5 classes, then a pattern from class 2 would be given the target vector $t = (0, 1, 0, 0, 0)^T$.
we identified three distinct approaches to the classification prob□lem. The simplest involves constructing a discriminant function that directly assigns each vector x to a specific class. A more powerful approach, however, models the conditional probability distribution $p(C_k|x)$ in an inference stage, and then subse□quently uses this distribution to make optimal decisions. By separating inference and decision, we gain numerous benefits. There are two different approaches to determining the conditional probabilities $p(C_k|x)$. One technique is to model them directly, for example by representing them as parametric models and then optimizing the parameters using a training set. Alternatively, we can adopt a generative approach in which we model the class-conditional densities given by $p(x|C_k)$, together with the prior probabilities $p(C_k)$ for the classes, and then we compute the required posterior probabilities using Bayes' theorem $p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$
In the simplest case, the model is also linear in the input variables and therefore takes the form $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \omega_0$, so that y is a real number. For classification problems, however, we wish to predict discrete class labels, or more generally posterior probabilities that lie in the range (0, 1). To achieve this, we consider a generalization of this model in which we transform the linear function of w using a nonlinear function f(·) so that $y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + w_0)$. In the machine learning literature f(·) is known as an activation function, whereas its inverse is called a link function in the statistics literature.
The decision surfaces correspond to $y(x) = constant$, so that $\mathbf{w}^T\mathbf{x} + w_0 = constant$ and hence the deci□sion surfaces are linear functions of x, even if the function f(·) is nonlinear. For this reason, the class of models described by $y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + w_0)$ are called generalized linear models_. They are no longer linear in the parameters due to the presence of the nonlinear function f(·).

Discriminant Functions

A discriminant is a function that takes an input vector $\mathbf{x}$ and assigns it to one of K classes, denoted C_k. We shall restrict attention to linear discriminants, namely those for which the decision surfaces are hyperplanes. To simplify the dis□cussion, we consider first the case of two classes and then investigate the extension to K > 2 classes.

Two classes

The simplest representation of a linear discriminant function is obtained by tak□ing a linear function of the input vector so that $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ where $\mathbf{w}$ is called a weight vector, and w_0 is a bias. The negative of the bias is sometimes called a threshold.
An input vector $\mathbf{x}$ is assigned to class C_1 if y(\mathbf x)□\ge0 and to class C_2 otherwise. The cor□responding decision boundary is therefore defined by the relation y(x)=0, which corresponds to a (D − 1)-dimensional hyperplane within the D-dimensional input space. Consider two points $\mathbf{x}_A$ and $\mathbf{x}_B$ both of which lie on the decision surface. Because $y(\mathbf{x}_A)$ = y(\mathbf x_B) $= 0$, we have $w^T(\mathbf{x}_A - \mathbf{x}_B) = 0$ and hence the vector $\mathbf{w}$ is orthogonal to every vector lying within the decision surface, and so $\mathbf{w}$ determines the orientation of the decision surface. Similarly, if $mathbfx$ is a point on the decision surface, then $y(\mathbf{x}) = 0$, and so the normal distance from the origin to the decision surface is given by

$$\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} = -\frac{w_0}{||\mathbf{w}||}$$

We therefore see that the bias parameter w_0 determines the location of the decision surface.

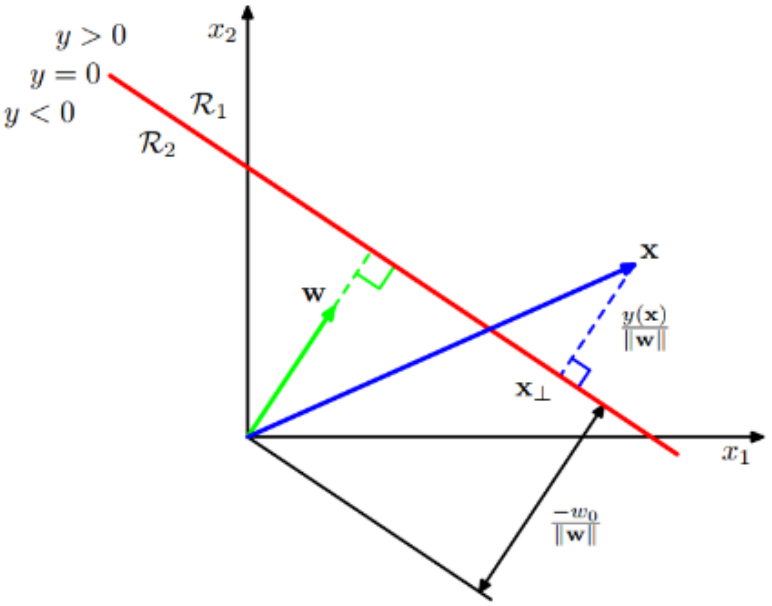


Figure 4.1 Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\mathbf{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\mathbf{x}$ from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$.

it is sometimes convenient to use a more compact notation in which we introduce an additional dummy 'input' value $x_0 = 1$ and then define $\tilde{\mathbf{w}} = (w_0, \mathbf{w})$ and $\tilde{\mathbf{x}} = (x_0, \mathbf{x})$ so that $y(\mathbf{x}) = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$. In this case, the decision surfaces are D-dimensional hyperplanes passing through the origin of the D + 1-dimensional expanded input space

Multiple classes

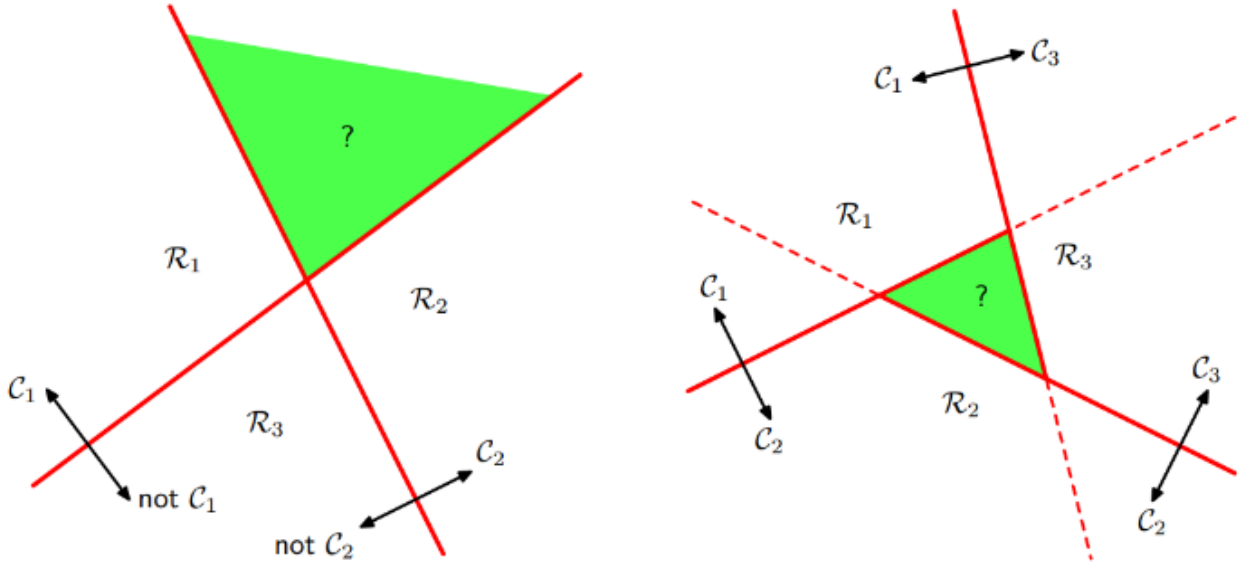


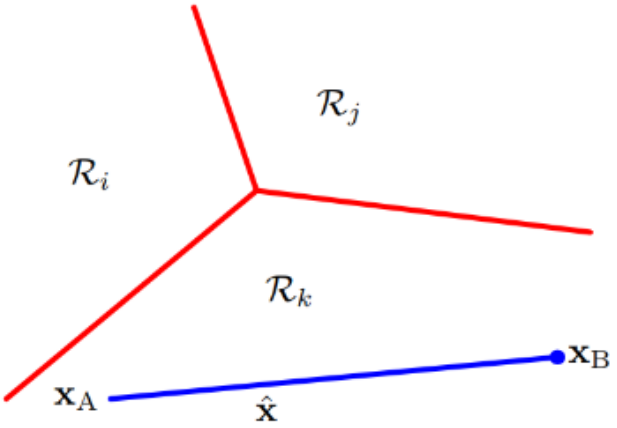
Figure 4.2 Attempting to construct a K class discriminant from a set of two class discriminants leads to ambiguous regions, shown in green. On the left is an example involving the use of two discriminants designed to distinguish points in class $\mathcal{C}_k$ from points not in class $\mathcal{C}_k$. On the right is an example involving three discriminant functions each of which is used to separate a pair of classes $\mathcal{C}_k$ and $\mathcal{C}_j$.

We might be tempted be to build a K-class discriminant by combining a number of two-class discriminant functions. However, this leads to some serious difficulties.

Consider the use of K−1 classifiers each of which solves a two-class problem of separating points in a particular class C_k from points not in that class. This is known as a one-versus-the-rest classifier. An alternative is to introduce K(K − 1)/2 binary discriminant functions, one for every possible pair of classes. This is known as a one-versus-one classifier. Each point is then classified according to a majority vote amongst the discriminant func□tions. However, this too runs into the problem of ambiguous regions.

We can avoid these difficulties by considering a single K-class discriminant comprising K linear functions of the form $y_k(\mathbf{x}) = \mathbf{w}_k^T\mathbf{x} + w_{k0}$ and then assigning a point $\mathbf{x}$ to class C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$. The decision boundary between class C_k and class C_j is therefore given by $y_k(\mathbf{x}) = y_j(\mathbf{x})$ and hence corresponds to a (D − 1)-dimensional hyperplane defined by $(\mathbf{w}_k - \mathbf{w}_j)^T\mathbf{x} + (w_{k0} - w_{j0}) = 0$.

Illustration of the decision regions for a multiclass linear discriminant, with the decision boundaries shown in red. If two points $\mathbf{x}_A$ and $\mathbf{x}_B$ both lie inside the same decision region $\mathcal{R}_k$, then any point $\hat{\mathbf{x}}$ that lies on the line connecting these two points must also lie in $\mathcal{R}_k$, and hence the decision region must be singly connected and convex.



Least squares for classification

we considered models that were linear functions of the parame□ters, and we saw that the minimization of a sum-of-squares error function led to a simple closed-form solution for the parameter values. It is therefore tempting to see if we can apply the same formalism to classification problems. Consider a general classification problem with K classes, with a 1-of-K binary coding scheme for the target vector $\mathbf{t}$. One justification for using least squares in such a context is that it approximates the conditional expectation $E[\mathbf{t}|\mathbf{x}]$ of the target values given the input vector.

Each class C_k is described by its own linear model so that $y_k(\mathbf{x}) = \mathbf{w}_k^T\mathbf{x} + w_{k0}$ where k = 1,...,K. We can conveniently group these together using vector nota□tion so that $y(\mathbf{x}) = \tilde{W}^T\tilde{\mathbf{x}}$ where $\tilde{\mathbf{W}}$, is a matrix whose k^{th} column comprises the D + 1-dimensional vector $\tilde{\mathbf{w}}_k = (w_{k0}, \mathbf{w}_k^T)^T$ and $\tilde{\mathbf{x}}$ is the corresponding augmented input vector $(1, \mathbf{x}^T)^T$ with a dummy input $x_0 = 1$. A new input $\mathbf{x}$ is then assigned to the class for which the output $y_k = \tilde{\mathbf{w}}_k^T\tilde{\mathbf{x}}$ is largest. We now determine the parameter matrix $\tilde{\mathbf{W}}$, by minimizing a sum-of-squares error function, as we did for regression. Consider a training data set $\mathbf{x}_n, \mathbf{t}_n$ where n = 1,...,N, and define a matrix $\mathbf{T}$ whose n^{th} row is the vector t_n^T, together with a matrix $\tilde{\mathbf{X}}$ whose n^{th} row is $\tilde{\mathbf{x}_n^T}$. The sum-of-squares error function can then be written as:

$$E_D(\tilde{\mathbf{W}}) = \frac{1}{2}Tr\{(\tilde{\mathbf{X}}\tilde{\mathbf{W}} - T)^T(\tilde{\mathbf{X}}\tilde{\mathbf{W}} - T)\}$$

Setting the derivative with respect to $\tilde{\mathbf{W}}$, to zero, and rearranging, we then obtain the solution for $\tilde{\mathbf{W}}$, in the form $\tilde{\mathbf{W}} = (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T\mathbf{T} = \tilde{\mathbf{X}}^\dagger\mathbf{T}$ where $\tilde{\mathbf{X}}^\dagger$ is the pseudo-inverse of the matrix $\tilde{\mathbf{X}}$. We then obtain the discriminant function in the form $y(\mathbf{x}) = \tilde{\mathbf{W}}^T\tilde{\mathbf{x}} = \mathbf{T}^T(\tilde{\mathbf{X}}^\dagger)^T\tilde{\mathbf{x}}$.

An interesting property of least-squares solutions with multiple target variables is that if every target vector in the training set satisfies some linear constraint $a^T\mathbf{t}_n + b = 0$ for some constants a and b, then the model prediction for any value of x will satisfy the same constraint so that $\mathbf{a}^T\mathbf{y}(\mathbf{x}) + b = 0$.

The least-squares approach gives an exact closed-form solution for the discrimi□nant function parameters. However, even as a discriminant function (where we use it to make decisions directly and dispense with any probabilistic interpretation) it suf□fers from some severe problems. The least-squares solutions lack robustness to outliers, and this applies equally to the classification application. The sum-of-squares error function penalizes predictions that are ‘too correct’ in that they lie a long way on the correct side of the decision boundary. The failure of least squares should not surprise us when we recall that it cor□responds to maximum likelihood under the assumption of a Gaussian conditional distribution, whereas binary target vectors clearly have a distribution that is far from Gaussian.

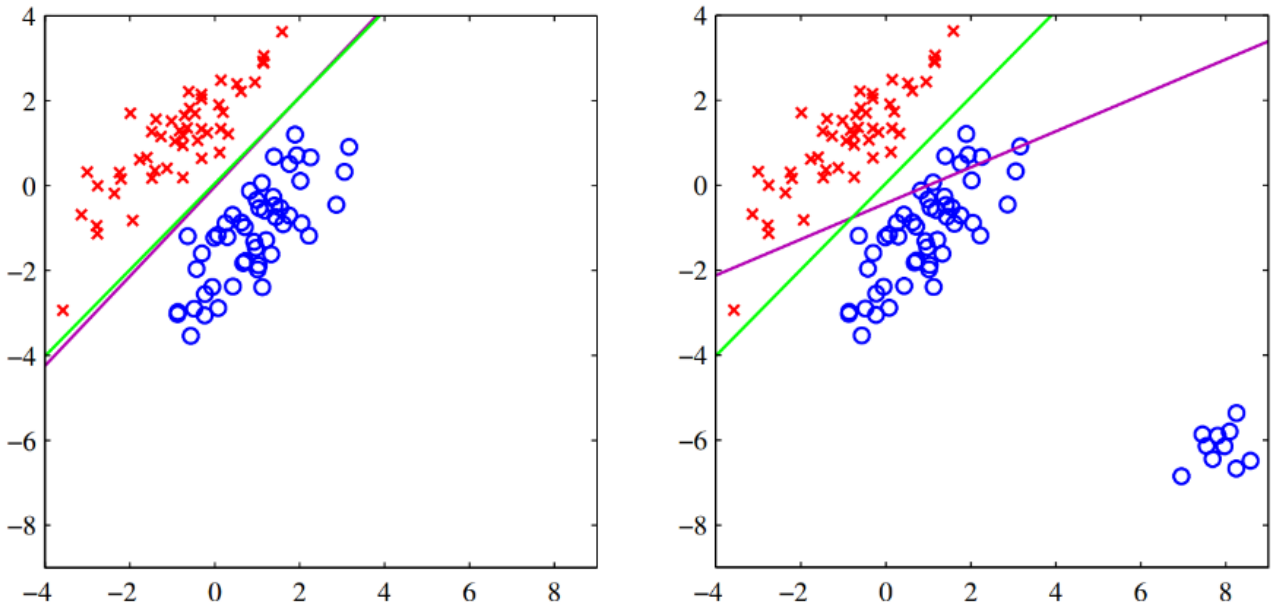


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

The perceptron algorithm

Example of a linear discriminant model.

It corresponds to a two-class model in which the input vector $\mathbf{x}$ is first transformed using a fixed nonlinear transformation to give a feature vector $\phi(\mathbf{x})$, and this is then used to construct a generalized linear model of the form $y(\mathbf{x}) = f(\mathbf{w}^T\phi(\mathbf{x}))$ where the nonlinear activation function f(·) is given by a step function of the form:

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

The vector $\phi(\mathbf{x})$ will typically include a bias component $\phi_0(\mathbf{x}) = 1$.

For the perceptron it is more convenient to use target values t = +1 for class C_1 and t = −1 for class C_2, which matches the choice of activation function. The algorithm used to determine the parameters $\mathbf{w}$ of the perceptron can most easily be motivated by error function minimization.

A natural choice of error func□tion would be the total number of misclassified patterns.

However, this does not lead to a simple learning algorithm because the error is a piecewise constant function of$\mathbf{w}$, with discontinuities wherever a change in $\mathbf{w}$ causes the decision boundary to move across one of the data points. Methods based on changing $\mathbf{w}$ using the gradi□ent of the error function cannot then be applied, because the gradient is zero almost everywhere. We therefore consider an alternative error function known as the perceptron cri□terion : $E_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\phi_n t_n$ where $\mathcal{M}$ denotes the set of all misclassified patterns.

The perceptron criterion associates zero error with any pattern that is correctly classified, whereas for a mis□classified pattern $\mathbf{x}_n$ it tries to minimize the quantity $-\mathbf{w}^T\phi(x_n)t_n$. The total error function is piecewise linear.

We now apply the stochastic gradient descent algorithm to this error function. The change in the weight vector $\mathbf{w}$ is then given by

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} = -\eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \phi_n t_n$$

where η is the learning rate parameter and τ is an integer that indexes the steps of the algorithm. Because the perceptron function $y(\mathbf{x}, \mathbf{w})$ is unchanged if we multiply $\mathbf{w}$ by a constant, we can set the learning rate parameter η equal to 1 without of generality. Note that, as the weight vector evolves during training, the set of patterns that are misclassified will change. The perceptron learning rule is not guaranteed to reduce the total error function at each stage. However, the **perceptron convergence theorem** states that if there exists an ex□act solution (in other words, if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite num□ber of steps. Note, however, that the number of steps required to achieve con□vergence could still be substantial, and in practice, until convergence is achieved, we will not be able to distinguish between a nonseparable problem and one that is simply slow to converge.
Even when the data set is linearly separable, there may be many solutions, and which one is found will depend on the initialization of the parameters and on the or□der of presentation of the data points. Furthermore, for data sets that are not linearly separable, the perceptron learning algorithm will never converge.

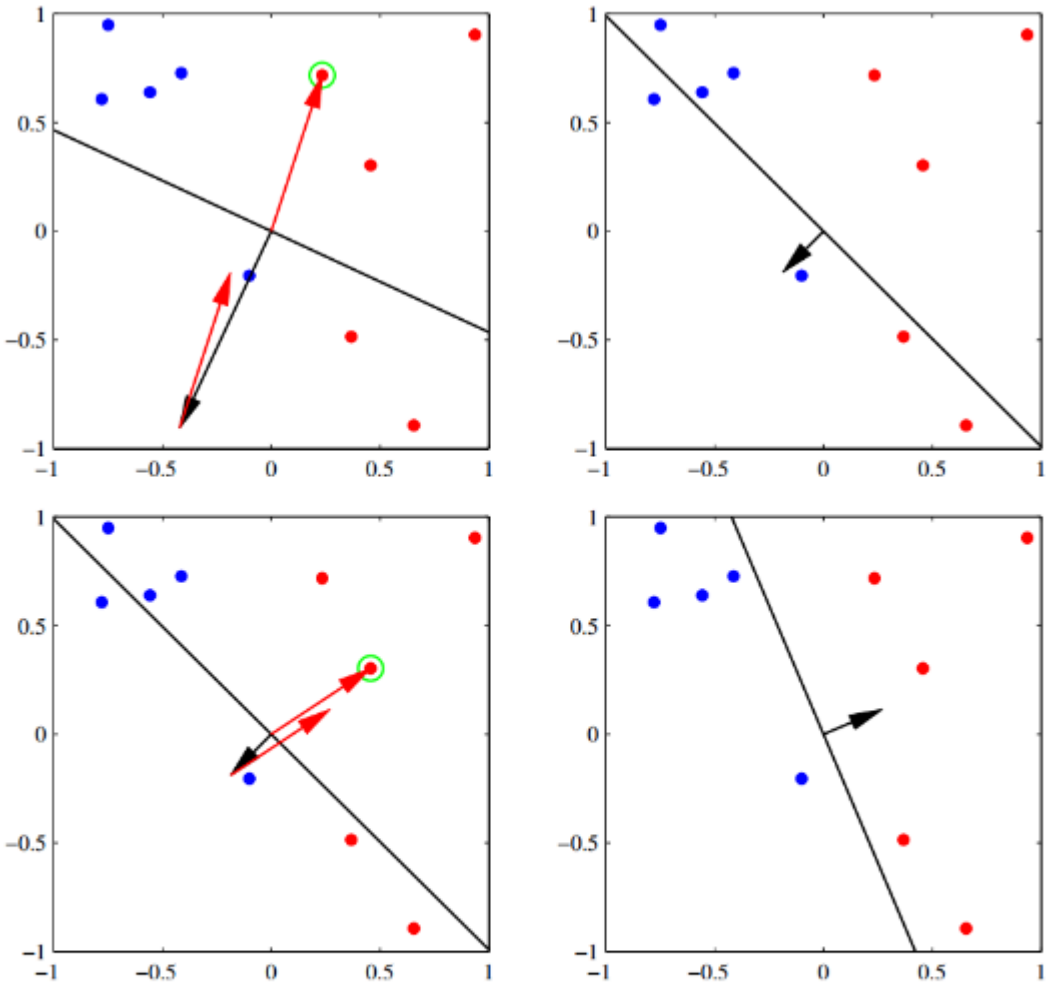


Figure 4.7 Illustration of the convergence of the perceptron learning algorithm, showing data points from two classes (red and blue) in a two-dimensional feature space (ϕ_1, ϕ_2). The top left plot shows the initial parameter vector $\mathbf{w}$ shown as a black arrow together with the corresponding decision boundary (black line), in which the arrow points towards the decision region which classified as belonging to the red class. The data point circled in green is misclassified and so its feature vector is added to the current weight vector, giving the new decision boundary shown in the top right plot. The bottom left plot shows the next misclassified point to be considered, indicated by the green circle, and its feature vector is again added to the weight vector giving the decision boundary shown in the bottom right plot for which all data points are correctly classified.

Fixed basis functions

All of the algorithms are equally applicable if we first make a fixed nonlinear transformation of the inputs using a vector of basis functions $\phi(\mathbf{x})$. The resulting decision boundaries will be linear in the feature space ϕ, and these correspond to nonlinear decision boundaries in the original x space. Classes that are linearly separable in the feature space $\phi(\mathbf{x})$ need not be linearly separable in the original observation space $\mathbf{x}$.
This corresponds to posterior probabilities $p(C_k|\mathbf{x})$, which, for at least some values of x, are not 0 or 1. In such cases, the opti□mal solution is obtained by modelling the posterior probabilities accurately and then applying standard decision theory.
Note that nonlinear transformations $\phi(\mathbf{x})$ cannot remove such class overlap. Indeed, they can increase the level of overlap, or create overlap where none existed in the original observation space. However, suitable choices of nonlinearity can make the process of modelling the posterior probabilities easier.

Logistic regression

The logic regression is $p(C_!|\phi) = y(\phi) = \sigma(\mathbf{w}^T\phi), p(C_2|\phi) = 1 - p(C_1|\phi)$.
For an M-dimensional feature space ϕ, this model has M adjustable parameters.
This model is useful when you are working with a large values of M.
For a data set $\{\phi_n, t_n\}$, where $t_n \in \{0, 1\}$ and $\phi_n = \phi(\mathbf{x}_n)$, with n = 1,...,N, the likelihood function can be written as $p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}\{1 - y_n\}^{1-t_n}$ where $\mathbf{t} = (t_1, \ldots, t_N)^T$ and $y_n = p(C_1|\phi_n)$. As usual, we can define an error function by taking the negative logarithm of the likelihood, which gives the cross□entropy error function in the form

$$E(\mathbf{w}) = -ln(p(\mathbf{t}|\mathbf{w})) = -\sum_{n=1}^{N}\{t_n ln(y_n) + (1 - t_n)ln(1 - y_n)\}$$

where $y_n = \sigma(a_n)$ and $a_n = \mathbf{w}^T\phi_n$. Taking the gradient of the error function with
Exercise 4.13 respect to w, we obtain:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi_n$$

It is worth noting that maximum likelihood can exhibit severe over-fitting for data sets that are linearly separable. This arises because the maximum likelihood so□lution occurs when the hyperplane corresponding to σ = 0.5, equivalent to $\mathbf{w}^T\phi = 0$, separates the two classes and the magnitude of $\mathbf{w}$ goes to infinity. In this case, the logistic sigmoid function becomes infinitely steep in feature space, corresponding to a Heaviside step function, so that every training point from each class k is assigned a posterior probability $p(C_k|\mathbf{x}) = 1$.
Note that the problem will arise even if the number of data points is large compared with the number of parameters in the model, so long as the training data set is linearly separable. The

singularity can be avoided by inclusion of a prior and finding a $\mathbf{MAP}$ solution for $\mathbf{w}$, or equivalently by adding a regularization term to the error function.