CHAPTER 7

Linear models

AFTER DEALING WITH logical models in the preceding chapters we now move on to a quite different kind of model. The models in this chapter and the next are defined in terms of the geometry of instance space. Geometric models most often assume that instances are described by d real-valued features, and thus $\mathscr{X} = \mathbb{R}^d$. For example, we could describe objects by their position on a map in terms of longitude and latitude ($d = 2$), or in the real world by longitude, latitude and altitude ($d = 3$). While most real-valued features are not intrinsically geometric – think of a person's age or an object's temperature – we can still imagine them being plotted in a d-dimensional Cartesian coordinate system. We can then use geometric concepts such as lines and planes to impose structure on this space, for instance in order to build a classification model. Alternatively, we can use the geometric notion of distance to represent similarity, on the basis that if two points are close together they have similar feature values and thus can be expected to behave similarly with respect to the property of interest. Such distance-based models are the subject of the next chapter. In this chapter we will look at models that can be understood in terms of lines and planes, commonly called *linear models*.

Linearity plays a fundamental role in mathematics and related disciplines, and the mathematics of linear models is well-understood (see Background 7.1 for the most important concepts). In machine learning, linear models are of particular interest because of their simplicity (remember our rule of thumb 'everything should be made as

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

If x_1 and x_2 are two scalars or vectors of the same dimension and α and β are arbitrary scalars, then $\alpha x_1 + \beta x_2$ is called a *linear combination* of x_1 and x_2. If f is a *linear function* of x, then

$$f(\alpha x_1 + \beta x_2) = \alpha f(x_1) + \beta f(x_2)$$

In words, the function value of a linear combination of some inputs is a linear combination of their function values. As a special case, if $\beta = 1 - \alpha$ we are taking a weighted average of x_1 and x_2, and the linearity of f then means that the function value of the weighted average is the weighted average of the function values.

Linear functions take particular forms, depending on the domain and codomain of f. If x and $f(x)$ are scalars, it follows that f is of the form $f(x) = a + bx$ for some constants a and b; a is called the *intercept* and b the *slope*. If $\mathbf{x} = (x_1, \ldots, x_d)$ is a vector and $f(\mathbf{x})$ is a scalar, then f is of the form

$$f(\mathbf{x}) = a + b_1 x_1 + \ldots + b_d x_d = a + \mathbf{b} \cdot \mathbf{x} \tag{7.1}$$

with $\mathbf{b} = (b_1, \ldots, b_d)$. The equation $f(\mathbf{x}) = 0$ defines a plane in $\mathbb{R}^d$ perpendicular to the *normal vector* $\mathbf{b}$.

The most general case is where $f(\mathbf{x})$ is a d'-dimensional vector, in which case f is of the form $f(\mathbf{x}) = \mathbf{M}\mathbf{x} + \mathbf{t}$, where $\mathbf{M}$ is a d'-by-d matrix representing a *linear transformation* such as a rotation or a scaling, and $\mathbf{t}$ is a d'-vector representing a translation. In this case f is called an *affine transformation* (the difference between linear and affine transformations is that the former maps the origin to itself; notice that a linear function of the form of Equation 7.1 is a linear transformation only if the intercept is 0).

In all these forms we can avoid representing the intercept a or the translation $\mathbf{t}$ separately by using homogeneous coordinates. For instance, by writing $\mathbf{b}^\circ = (a, b_1, \ldots, b_d)$ and $\mathbf{x}^\circ = (1, x_1, \ldots, x_d)$ in Equation 7.1 we have $f(\mathbf{x}) = \mathbf{b}^\circ \cdot \mathbf{x}^\circ$ (see also Background 1.2 on p.24).

Examples of non-linear functions are the polynomials in x of degree $p > 1$: $g(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_p x^p = \sum_{i=0}^{p} a_i x^i$. Other non-linear functions can be approximated by a polynomial through their Taylor expansion. The *linear approximation* of a function g at x_0 is $g(x_0) + g'(x_0)(x - x_0)$, where $g'(x)$ is the derivative of x. A *piecewise linear* approximation is obtained by combining several linear approximations at different points x_0.

Background 7.1. Linear models.

simple as possible, but not simpler' that we introduced on p.30). Here are a couple of manifestations of this simplicity.

☞ Linear models are *parametric*, meaning that they have a fixed form with a small number of numeric parameters that need to be learned from data. This is

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

different from tree or rule models, where the structure of the model (e.g., which features to use in the tree, and where) is not fixed in advance.

☞ Linear models are stable, which is to say that small variations in the training data have only limited impact on the learned model. Tree models tend to vary more with the training data, as the choice of a different split at the root of the tree typically means that the rest of the tree is different as well.

☞ Linear models are less likely to overfit the training data than some other models, largely because they have relatively few parameters. The flipside of this is that they sometimes lead to *underfitting*: e.g., imagine you are learning where the border runs between two countries from labelled samples, then a linear model is unlikely to give a good approximation.

The last two points can be summarised by saying that linear models have low variance but high bias. Such models are often preferable when you have limited data and want to avoid overfitting. High variance–low bias models such as decision trees are preferable if data is abundant but underfitting is a concern. It is usually a good idea to start with simple, high-bias models such as linear models and only move on to more elaborate models if the simpler ones appear to be underfitting.

Linear models exist for all predictive tasks, including classification, probability estimation and regression. Linear regression, in particular, is a well-studied problem that can be solved by the least-squares method, which is the topic of the next section. We will look at a number of other linear models in this chapter, including least-squares classification (also in Section 7.1), the perceptron in Section 7.2, and the support vector machine in Section 7.3. We will also find out how these models can be turned into probability estimators in Section 7.4. Finally, Section 7.5 briefly discusses how each of these methods could learn non-linear models by means of so-called kernel functions.

7.1 The least-squares method

We start by introducing a method that can be used to learn linear models for classification and regression. Recall that the regression problem is to learn a function estimator $\hat{f} : \mathscr{X} \to \mathbb{R}$ from examples $(x_i, f(x_i))$, where in this chapter we assume $\mathscr{X} = \mathbb{R}^d$. The differences between the actual and estimated function values on the training examples are called *residuals* $\epsilon_i = f(x_i) - \hat{f}(x_i)$. The *least-squares method*, introduced by Carl Friedrich Gauss in the late eighteenth century, consists in finding $\hat{f}$ such that $\sum_{i=1}^{n} \epsilon_i^2$ is minimised. The following example illustrates the method in the simple case of a single feature, which is called *univariate regression*.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

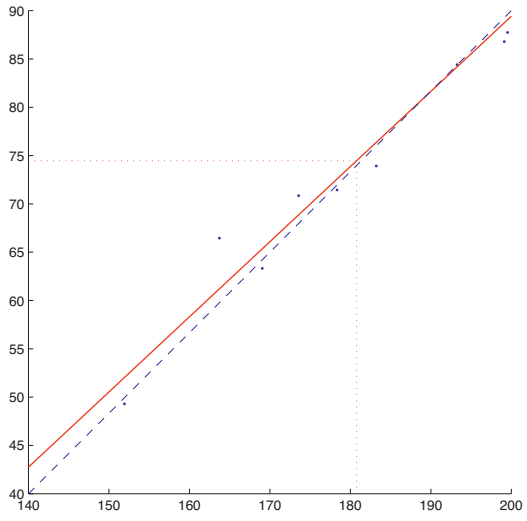

Figure 7.1. The red solid line indicates the result of applying linear regression to 10 measurements of body weight (on the y-axis, in kilograms) against body height (on the x-axis, in centimetres). The orange dotted lines indicate the average height $\overline{h} = 181$ and the average weight $\overline{w} = 74.5$; the regression coefficient $\hat{b} = 0.78$. The measurements were simulated by adding normally distributed noise with mean 0 and variance 5 to the true model indicated by the blue dashed line ($b = 0.83$).

Example 7.1 (Univariate linear regression). Suppose we want to investigate the relationship between people's height and weight. We collect n height and weight measurements $(h_i, w_i), 1 \le i \le n$. Univariate linear regression assumes a linear equation $w = a + bh$, with parameters a and b chosen such that the sum of squared residuals $\sum_{i=1}^{n}(w_i - (a + bh_i))^2$ is minimised. In order to find the parameters we take partial derivatives of this expression, set the partial derivatives to 0 and solve for a and b:

$$\frac{\partial}{\partial a}\sum_{i=1}^{n}(w_i - (a+bh_i))^2 = -2\sum_{i=1}^{n}(w_i - (a+bh_i)) = 0 \qquad \Rightarrow \hat{a} = \overline{w} - \hat{b}\overline{h}$$

$$\frac{\partial}{\partial b}\sum_{i=1}^{n}(w_i - (a+bh_i))^2 = -2\sum_{i=1}^{n}(w_i - (a+bh_i))h_i = 0$$

$$\Rightarrow \hat{b} = \frac{\sum_{i=1}^{n}(h_i - \overline{h})(w_i - \overline{w})}{\sum_{i=1}^{n}(h_i - \overline{h})^2}$$

So the solution found by linear regression is $w = \hat{a} + \hat{b}h = \overline{w} + \hat{b}(h - \overline{h})$; see Figure 7.1 for an example.

---

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

It is worthwhile to note that the expression for the *regression coefficient* or slope $\hat{b}$ derived in this example has n times the covariance between h and w in the enumerator and n times the variance of h in the denominator. This is true in general: for a feature x and a target variable y, the regression coefficient is

$$\hat{b} = n\frac{\sigma_{xy}}{n\sigma_{xx}} = \frac{\sigma_{xy}}{\sigma_{xx}}$$

(Here I use σ_{xx} as an alternative notation for σ_x^2, the variance of variable x.) This can be understood by noting that the covariance is measured in units of x times units of y (e.g., metres times kilograms in Example 7.1) and the variance in units of x squared (e.g., metres squared), so their quotient is measured in units of y per unit of x (e.g., kilograms per metre).

We can notice a few more useful things. The intercept $\hat{a}$ is such that the regression line goes through $(\overline{x}, \overline{y})$. Adding a constant to all x-values (a translation) will affect only the intercept but not the regression coefficient (since it is defined in terms of deviations from the mean, which are unaffected by a translation). So we could *zero-centre* the x-values by subtracting $\overline{x}$, in which case the intercept is equal to $\overline{y}$. We could even subtract $\overline{y}$ from all y-values to achieve a zero intercept, without changing the problem in an essential way.

Furthermore, suppose we replace x_i with $x'_i = x_i/\sigma_{xx}$ and likewise $\overline{x}$ with $\overline{x'} = \overline{x}/\sigma_{xx}$, then we have that $\hat{b} = \frac{1}{n}\sum_{i=1}^{n}(x'_i - \overline{x'})(y_i - \overline{y}) = \sigma_{x'y}$. In other words, if we *normalise* x by dividing all its values by x's variance, we can take the covariance between the normalised feature and the target variable as regression coefficient. In other words, univariate linear regression can be understood as consisting of two steps:

1. normalisation of the feature by dividing its values by the feature's variance;

2. calculating the covariance of the target variable and the normalised feature.

We will see below how these two steps change when dealing with more than one feature.

Another important point to note is that the sum of the residuals of the least-squares solution is zero:

$$\sum_{i=1}^{n}(y_i - (\hat{a} + \hat{b}x_i)) = n(\overline{y} - \hat{a} - \hat{b}\overline{x}) = 0$$

The result follows because $\hat{a} = \overline{y} - \hat{b}\overline{x}$, as derived in Example 7.1. While this property is intuitively appealing, it is worth keeping in mind that it also makes linear regression susceptible to *outliers*: points that are far removed from the regression line, often because of measurement errors.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

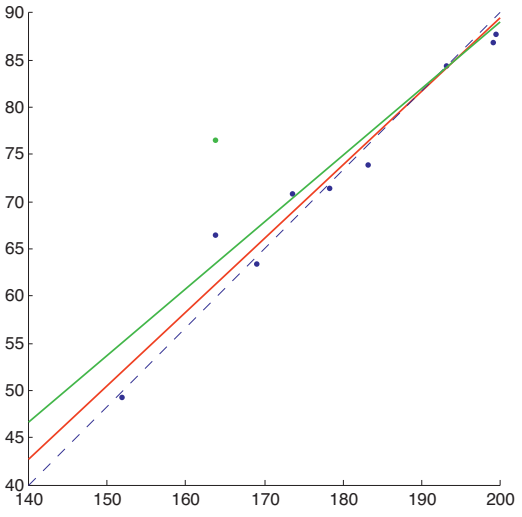

Figure 7.2. The effect of an outlier in univariate regression. One of the blue points got moved up 10 units to the green point, changing the red regression line to the green line.

Example 7.2 (The effect of outliers). Suppose that, as the result of a transcription error, one of the weight values in Figure 7.1 is increased by 10 kg. Figure 7.2 shows that this has a considerable effect on the least-squares regression line.

Despite this sensitivity to outliers, the least-squares method usually works surprisingly well for such a simple method. How can it be justified? One way to look at this is to assume that the true function f is indeed linear, but that the observed y-values are contaminated with random noise. That is, our examples are $(x_i, f(x_i) + \epsilon_i)$ rather than $(x_i, f(x_i))$, and we assume that $f(x) = ax + b$ for some a and b. If we knew a and b we could work out exactly what the residuals are, and if we knew σ^2 we could calculate the probability of observing that set of residuals. Since we don't know a and b we have to estimate them, and the estimate we want is the value of a and b that maximises the probability of the residuals. We will see in Chapter 9 that this so-called ☞*maximum-likelihood estimate* is exactly the least-squares solution.

Variants of the least-squares method exist. Here we discussed *ordinary* least squares, which assumes that only the y-values are contaminated with random noise. *Total* least squares generalises this to the situation that both x- and y-values are noisy, but this does not necessarily have a unique solution.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

$\mathbf{X}$ usually denotes an n-by-d data matrix containing n instances in rows described by d features or variables in columns. $\mathbf{X}_{r\cdot}$ denotes the r-th row of $\mathbf{X}$, $\mathbf{X}_{\cdot c}$ denotes the c-th column, and $\mathbf{X}_{rc}$ denotes the entry in the r-th row and c-th column. We also use i and j to range over rows and columns, respectively. The j-th column mean is defined as $\mu_j = \frac{1}{n}\sum_{i=1}^{n}\mathbf{X}_{ij}$; $\boldsymbol{\mu}^{\mathrm{T}}$ is a row vector containing all column means. If $\mathbf{1}$ is an n-vector containing only ones, then $\mathbf{1}\boldsymbol{\mu}^{\mathrm{T}}$ is an n-by-d matrix whose rows are $\boldsymbol{\mu}^{\mathrm{T}}$; hence $\mathbf{X}' = \mathbf{X} - \mathbf{1}\boldsymbol{\mu}^{\mathrm{T}}$ has mean zero in each column and is referred to as the *zero-centred* data matrix.

The *scatter matrix* is the d-by-d matrix $\mathbf{S} = \mathbf{X}'^{\mathrm{T}}\mathbf{X}' = \left(\mathbf{X} - \mathbf{1}\boldsymbol{\mu}^{\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{X} - \mathbf{1}\boldsymbol{\mu}^{\mathrm{T}}\right) = \mathbf{X}^{\mathrm{T}}\mathbf{X} - n\mathbf{M}$, where $\mathbf{M} = \boldsymbol{\mu}\boldsymbol{\mu}^{\mathrm{T}}$ is a d-by-d matrix whose entries are products of column means $\mathbf{M}_{jc} = \mu_j\mu_c$. The *covariance matrix* of $\mathbf{X}$ is $\boldsymbol{\Sigma} = \frac{1}{n}\mathbf{S}$ whose entries are the pairwise covariances $\sigma_{jc} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{X}_{ij} - \mu_j\right)\left(\mathbf{X}_{ic} - \mu_c\right) = \frac{1}{n}\left(\sum_{i=1}^{n}\mathbf{X}_{ij}\mathbf{X}_{ic} - \mu_i\mu_c\right)$. Two uncorrelated features have a covariance close to 0; positively correlated features have a positive covariance, indicating a certain tendency to increase or decrease together; a negative covariance indicates that if one feature increases, the other tends to decrease and vice versa. $\sigma_{jj} = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{X}_{ij} - \mu_j\right)^2 = \frac{1}{n}\left(\sum_{i=1}^{n}\mathbf{X}_{ij}^2 - \mu_j^2\right)$ is the *variance* of column j, also denoted as σ_j^2. The variance is always positive and indicates the spread of the values of a feature around their mean.

A small example clarifies these definitions:

$$\mathbf{X} = \begin{pmatrix} 5 & 0 \\ 3 & 5 \\ 1 & 7 \end{pmatrix} \quad \mathbf{1}\boldsymbol{\mu}^{\mathrm{T}} = \begin{pmatrix} 3 & 4 \\ 3 & 4 \\ 3 & 4 \end{pmatrix} \quad \mathbf{X}' = \begin{pmatrix} 2 & -4 \\ 0 & 1 \\ -2 & 3 \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} 25 & 15 & 5 \\ 15 & 34 & 38 \\ 5 & 38 & 50 \end{pmatrix}$$

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} = \begin{pmatrix} 35 & 22 \\ 22 & 74 \end{pmatrix} \quad \mathbf{M} = \begin{pmatrix} 9 & 12 \\ 12 & 16 \end{pmatrix} \quad \mathbf{S} = \begin{pmatrix} 8 & -14 \\ -14 & 26 \end{pmatrix} \quad \boldsymbol{\Sigma} = \begin{pmatrix} 8/3 & -14/3 \\ -14/3 & 26/3 \end{pmatrix}$$

We see that the two features are negatively correlated and that the second feature has the larger variance. Another way to calculate the scatter matrix is as a sum of outer products, one for each data point: $\mathbf{S} = \sum_{i=1}^{n}\left(\mathbf{X}_{i\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{X}_{i\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right)$. In our example we have

$$\left(\mathbf{X}_{1\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{X}_{1\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right) = \begin{pmatrix} 2 \\ -4 \end{pmatrix}\begin{pmatrix} 2 & -4 \end{pmatrix} = \begin{pmatrix} 4 & -8 \\ -8 & 16 \end{pmatrix}$$

$$\left(\mathbf{X}_{2\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{X}_{2\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\left(\mathbf{X}_{3\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{X}_{3\cdot} - \boldsymbol{\mu}^{\mathrm{T}}\right) = \begin{pmatrix} -2 \\ 3 \end{pmatrix}\begin{pmatrix} -2 & 3 \end{pmatrix} = \begin{pmatrix} 4 & -6 \\ -6 & 9 \end{pmatrix}$$

Background 7.2. Some more matrix notation.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

Multivariate linear regression

In order to deal with an arbitrary number of features it will be useful to employ matrix notation (see Background 7.2). We can write univariate linear regression in matrix form as

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} a + \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} b + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

$$\mathbf{y} = \mathbf{a} + \mathbf{X}\mathbf{b} + \boldsymbol{\epsilon}$$

In the second form of this equation, $\mathbf{y}$, $\mathbf{a}$, $\mathbf{X}$ and $\boldsymbol{\epsilon}$ are n-vectors, and $\mathbf{b}$ is a scalar. In case of d features, all that changes is that $\mathbf{X}$ becomes an n-by-d matrix, and $\mathbf{b}$ becomes a d-vector of regression coefficients.

We can apply the by now familiar trick of using homogeneous coordinates to simplify these equations as follows:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

$$\mathbf{y} = \mathbf{X}^{\circ}\mathbf{w} + \boldsymbol{\epsilon}$$

with $\mathbf{X}^{\circ}$ an n-by-$(d+1)$ matrix whose first column is all 1s and the remaining columns are the columns of $\mathbf{X}$, and $\mathbf{w}$ has the intercept as its first entry and the regression coefficients as the remaining d entries. For convenience we will often blur the distinction between these two formulations and state the regression equation as $\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$ with $\mathbf{X}$ having d columns and $\mathbf{w}$ having d rows – from the context it will be clear whether we are representing the intercept by means of homogeneous coordinates, or have rather zero-centred the target and features to achieve a zero intercept.

In the univariate case we were able to obtain a closed-form solution for $\mathbf{w}$: can we do the same in the multivariate case? First, we are likely to need the covariances between every feature and the target variable. Consider the expression $\mathbf{X}^{\mathrm{T}}\mathbf{y}$, which is an n-vector, the j-th entry of which is the product of the j-th row of $\mathbf{X}^{\mathrm{T}}$ – i.e., the j-th column of $\mathbf{X}$, which is $(x_{1j}, \ldots, x_{nj})$ – with $(y_1, \ldots, y_n)$:

$$(\mathbf{X}^{\mathrm{T}}\mathbf{y})_j = \sum_{i=1}^{n} x_{ij} y_i = \sum_{i=1}^{n} (x_{ij} - \mu_j)(y_i - \overline{y}) + n\mu_j \overline{y} = n(\sigma_{jy} + \mu_j \overline{y})$$

Assuming for the moment that every feature is zero-centred, we have $\mu_j = 0$ and thus $\mathbf{X}^{\mathrm{T}}\mathbf{y}$ is an n-vector holding all the required covariances (times n).

In the univariate case we needed to normalise the features to have unit variance. In the multivariate case we can achieve this by means of a d-by-d scaling matrix: a

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

diagonal matrix with diagonal entries $1/n\sigma_{jj}$. If $\mathbf{S}$ is a diagonal matrix with diagonal entries $n\sigma_{jj}$, we can get the required scaling matrix by simply inverting $\mathbf{S}$. So our first stab at a solution for the *multivariate regression* problem is

$$\hat{\mathbf{w}} = \mathbf{S}^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{7.2}$$

As it turns out, the general case requires a more elaborate matrix instead of $\mathbf{S}$:

$$\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{7.3}$$

Let us try to understand the term $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ a bit better. Assume that the features are uncorrelated (meaning the covariance between every pair of different features is 0) in addition to being zero-centred. In the notation of Background 7.2, the covariance matrix $\mathbf{\Sigma}$ is diagonal with entries σ_{jj}. Since $\mathbf{X}^{\mathrm{T}}\mathbf{X} = n(\mathbf{\Sigma} + \mathbf{M})$, and since the entries of $\mathbf{M}$ are 0 because the columns of $\mathbf{X}$ are zero-centred, this matrix is also diagonal with entries $n\sigma_{jj}$ – in fact, it is the matrix $\mathbf{S}$ referred to above. In other words, assuming zero-centred and uncorrelated features, $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ reduces to our scaling matrix $\mathbf{S}^{-1}$. In the general case we cannot make any assumptions about the features, and $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ *acts as a transformation that decorrelates, centres and normalises the features*.

To make this a bit more concrete, the next example shows how this works out in the bivariate case.

Example 7.3 (Bivariate linear regression in matrix notation). First, we derive the basic expressions.

$$\begin{aligned}
\mathbf{X}^{\mathrm{T}}\mathbf{X} &= \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{pmatrix} = n \begin{pmatrix} \sigma_{11} + \overline{x_1}^2 & \sigma_{12} + \overline{x_1}\,\overline{x_2} \\ \sigma_{12} + \overline{x_1}\,\overline{x_2} & \sigma_{22} + \overline{x_2}^2 \end{pmatrix} \\
(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} &= \frac{1}{nD} \begin{pmatrix} \sigma_{22} + \overline{x_2}^2 & -\sigma_{12} - \overline{x_1}\,\overline{x_2} \\ -\sigma_{12} - \overline{x_1}\,\overline{x_2} & \sigma_{11} + \overline{x_1}^2 \end{pmatrix} \\
D &= (\sigma_{11} + \overline{x_1}^2)(\sigma_{22} + \overline{x_2}^2) - (\sigma_{12} + \overline{x_1}\,\overline{x_2})^2 \\
\mathbf{X}^{\mathrm{T}}\mathbf{y} &= \begin{pmatrix} x_{11} & \cdots & x_{n1} \\ x_{12} & \cdots & x_{n2} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = n \begin{pmatrix} \sigma_{1y} + \overline{x_1}\,\overline{y} \\ \sigma_{2y} + \overline{x_2}\,\overline{y} \end{pmatrix}
\end{aligned}$$

We now consider two special cases. The first is that $\mathbf{X}$ is in homogeneous coordinates, i.e., we are really dealing with a univariate problem. In that case we have

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

$x_{i1} = 1$ for $1 \leq i \leq n$; $\overline{x_1} = 1$; and $\sigma_{11} = \sigma_{12} = \sigma_{1y} = 0$. We then obtain (we write x instead of x_2, σ_{xx} instead of σ_{22} and σ_{xy} instead of σ_{2y}):

$$\begin{aligned}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} &= \frac{1}{n\sigma_{xx}}\begin{pmatrix}\sigma_{xx}+\overline{x}^2 & -\overline{x}\\ -\overline{x} & 1\end{pmatrix}\\ \mathbf{X}^{\mathrm{T}}\mathbf{y} &= n\begin{pmatrix}\overline{y}\\ \sigma_{xy}+\overline{x}\,\overline{y}\end{pmatrix}\\ \hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} &= \frac{1}{\sigma_{xx}}\begin{pmatrix}\sigma_{xx}\overline{y}-\sigma_{xy}\overline{x}\\ \sigma_{xy}\end{pmatrix}\end{aligned}$$

This is the same result as obtained in Example 7.1.

The second special case we consider is where we assume x_1, x_2 and y to be zero-centred, which means that the intercept is zero and $\mathbf{w}$ contains the two regression coefficients. In this case we obtain

$$\begin{aligned}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} &= \frac{1}{n(\sigma_{11}\sigma_{22}-\sigma_{12}^2)}\begin{pmatrix}\sigma_{22} & -\sigma_{12}\\ -\sigma_{12} & \sigma_{11}\end{pmatrix}\\ \mathbf{X}^{\mathrm{T}}\mathbf{y} &= n\begin{pmatrix}\sigma_{1y}\\ \sigma_{2y}\end{pmatrix}\\ \hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} &= \frac{1}{(\sigma_{11}\sigma_{22}-\sigma_{12}^2)}\begin{pmatrix}\sigma_{22}\sigma_{1y}-\sigma_{12}\sigma_{2y}\\ \sigma_{11}\sigma_{2y}-\sigma_{12}\sigma_{1y}\end{pmatrix}\end{aligned}$$

The last expression shows, e.g., that the regression coefficient for x_1 may be non-zero even if x_1 doesn't correlate with the target variable ($\sigma_{1y} = 0$), on account of the correlation between x_1 and x_2 ($\sigma_{12} \neq 0$).

Notice that if we do assume $\sigma_{12} = 0$ then the components of $\hat{\mathbf{w}}$ reduce to σ_{jy}/σ_{jj}, which brings us back to Equation 7.2. *Assuming uncorrelated features effectively decomposes a multivariate regression problem into d univariate problems.* We shall see several other examples of decomposing multivariate learning problems into univariate problems in this book – in fact, we have already seen an example in the form of the ☞*naive Bayes* classifier in Chapter 1. So, you may wonder, why take feature correlation into account at all?

The answer is that ignoring feature correlation can be harmful in certain situations. Consider Figure 7.3: on the left, there is little correlation among the features, and as a result the samples provide a lot of information about the true function. On

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

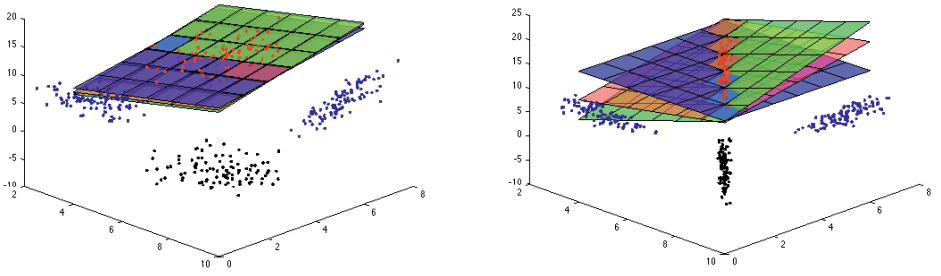

Figure 7.3. (left) Regression functions learned by linear regression. The true function is $y = x_1 + x_2$ (red plane). The red points are noisy samples of this function; the black points show them projected onto the (x_1, x_2)-plane. The green plane indicates the function learned by linear regression; the blue plane is the result of decomposing the problem into two univariate regression problems (blue points). Both are good approximations of the true function. **(right)** The same function, but now x_1 and x_2 are highly (negatively) correlated. The samples now give much less information about the true function: indeed, from the univariate decomposition it appears that the function is constant.

the right, the features are highly negatively correlated in such a way that the sampled values $y = x_1 + x_2 + \epsilon$ appear nearly constant, as any increase in one feature is accompanied by a nearly equal decrease in the other. As a result, decomposing the problem into two univariate regression problems leads to learning a nearly constant function. To be fair, taking the full covariance matrix into account doesn't do so well either in this example. However, although we will not explore the details here, one advantage of the full covariance approach is that it allows us to recognise that we can't place much confidence in our estimates of the regression parameters in this situation. The computational cost of computing the closed-form solution in Equation 7.3 lies in inverting the d-by-d matrix $\mathbf{X}^{\mathrm{T}}\mathbf{X}$, which can be prohibitive in high-dimensional feature spaces.

Regularised regression

We have just seen a situation in which least-squares regression can become *unstable*: i.e., highly dependent on the training data. Instability is a manifestation of a tendency to overfit. *Regularisation* is a general method to avoid such overfitting by applying additional constraints to the weight vector. A common approach is to make sure the weights are, on average, small in magnitude: this is referred to as *shrinkage*. To show how this can be achieved, we first write down the least-squares regression problem as an optimisation problem:

$$\mathbf{w}^* = \operatorname*{arg\,min}_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}} (\mathbf{y} - \mathbf{X}\mathbf{w})$$

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

The right-hand side is just a way to write the sum of squared residuals as a dot product. The regularised version of this optimisation is then as follows:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda ||\mathbf{w}||^2 \tag{7.4}$$

where $||\mathbf{w}||^2 = \sum_i w_i^2$ is the squared norm of the vector $\mathbf{w}$, or, equivalently, the dot product $\mathbf{w}^{\mathrm{T}}\mathbf{w}$; λ is a scalar determining the amount of regularisation. This regularised problem still has a closed-form solution:

$$\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{7.5}$$

where $\mathbf{I}$ denotes the identity matrix with 1s on the diagonal and 0s everywhere else. Comparing this with Equation 7.3 on p.202 we see that regularisation amounts to adding λ to the diagonal of $\mathbf{X}^{\mathrm{T}}\mathbf{X}$, a well-known trick to improve the numerical stability of matrix inversion. This form of least-squares regression is known as *ridge regression*.

An interesting alternative form of regularised regression is provided by the *lasso*, which stands for 'least absolute shrinkage and selection operator'. It replaces the ridge regularisation term $\sum_i w_i^2$ with the sum of absolute weights $\sum_i |w_i|$. (Using terminology that will be introduced in Definition 8.2 on p.235: lasso uses L_1 regularisation where ridge regression uses the L_2 norm.) The result is that some weights are shrunk, but others are set to 0, and so the lasso regression favours *sparse solutions*. It should be added that lasso regression is quite sensitive to the regularisation parameter λ, which is usually set on hold-out data or in cross-validation. Also, there is no closed-form solution and so some numerical optimisation technique must be applied.

Using least-squares regression for classification

So far we have used the least-squares method to construct function approximators. Interestingly, we can also use linear regression to learn a binary classifier by encoding the two classes as real numbers. For instance, we can label the *Pos* positive examples with $y^{\oplus} = +1$ and the *Neg* negative examples with $y^{\ominus} = -1$. It then follows that $\mathbf{X}^{\mathrm{T}}\mathbf{y} = Pos\,\boldsymbol{\mu}^{\oplus} - Neg\,\boldsymbol{\mu}^{\ominus}$, where $\boldsymbol{\mu}^{\oplus}$ and $\boldsymbol{\mu}^{\ominus}$ are d-vectors containing each feature's mean values for the positive and negative examples, respectively.

Example 7.4 (Univariate least-squares classifier). In the univariate case we have $\sum_i x_i y_i = Pos\,\mu^{\oplus} - Neg\,\mu^{\ominus}$; we also know (see Example 7.3) that $\sum_i x_i y_i = n(\sigma_{xy} + \overline{x}\,\overline{y})$, and so $\sigma_{xy} = pos\,\mu^{\oplus} - neg\,\mu^{\ominus} - \overline{x}\,\overline{y}$. Since $\overline{x} = pos\,\mu^{\oplus} + neg\,\mu^{\ominus}$ and $\overline{y} = pos - neg$, we can rewrite the covariance between x and y as $\sigma_{xy} =$

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

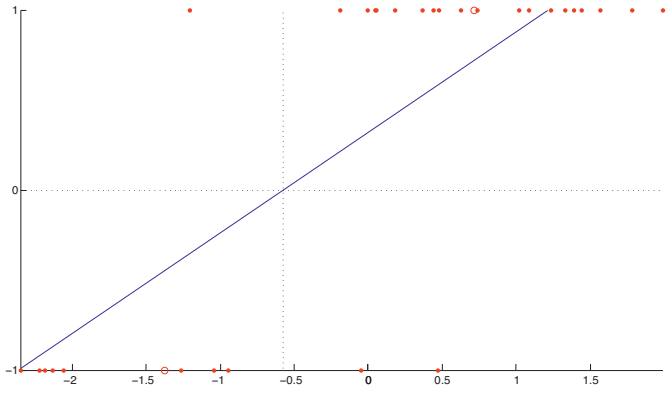

Figure 7.4. Using univariate linear regression to obtain a decision boundary. The 10 negative examples are labelled with $y^{\ominus} = -1$ and the 20 positive examples are labelled $y^{\oplus} = +1$. $\mu^{\ominus}$ and $\mu^{\oplus}$ are indicated by red circles. The blue line is the linear regression line $y = \overline{y} + \hat{b}(x - \overline{x})$, and the crosshair indicates the decision boundary $x_0 = \overline{x} - \overline{y}/\hat{b}$. This results in three examples being misclassified – notice that this is the best that can be achieved with the given data.

$2pos \cdot neg\,(\mu^{\oplus} - \mu^{\ominus})$, and so the slope of the regression line is

$$\hat{b} = 2pos \cdot neg \frac{\mu^{\oplus} - \mu^{\ominus}}{\sigma_{xx}} \tag{7.6}$$

This equation shows that the slope of the regression line increases with the separation between the classes (measured as the distance between the class means in proportion to the feature's variance), but also decreases if the class distribution becomes skewed.

The regression equation $y = \overline{y} + \hat{b}(x - \overline{x})$ can then be used to obtain a decision boundary. We need to determine the point (x_0, y_0) such that y_0 is half-way between $y^{\oplus}$ and $y^{\ominus}$ (i.e., $y_0 = 0$ in our case). We then have

$$x_0 = \overline{x} + \frac{y_0 - \overline{y}}{\hat{b}} = \overline{x} - \frac{pos - neg}{2pos \cdot neg} \frac{\sigma_{xx}}{\mu^{\oplus} - \mu^{\ominus}}$$

That is, if there are equal numbers of positive and negative examples we simply threshold the feature at the feature mean $\overline{x}$; in case of unequal class distribution we shift this threshold to the left or right as appropriate (Figure 7.4).

In the general case, the *least-squares classifier* learns the decision boundary $\mathbf{w} \cdot \mathbf{x} = t$ with

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}(Pos\,\boldsymbol{\mu}^{\oplus} - Neg\,\boldsymbol{\mu}^{\ominus}) \tag{7.7}$$

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

We would hence assign class $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x} - t)$ to instance $\mathbf{x}$, where

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Various simplifying assumptions can be made, including zero-centred features, equal-variance features, uncorrelated features and equal class prevalences. In the simplest case, when all these assumptions are made, Equation 7.7 reduces to $\mathbf{w} = c(\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})$ where c is some scalar that can be incorporated in the decision threshold t. We recognise this as the ☞*basic linear classifier* that was introduced in the Prologue. Equation 7.7 thus tells us how to adapt the basic linear classifier, using the least-squares method, in order to take feature correlation and unequal class distributions into account.

In summary, *a general way of constructing a linear classifier with decision boundary* $\mathbf{w} \cdot \mathbf{x} = t$ *is by constructing* $\mathbf{w}$ *as* $\mathbf{M}^{-1}(n^{\oplus}\boldsymbol{\mu}^{\oplus} - n^{\ominus}\boldsymbol{\mu}^{\ominus})$, with different possible choices of $\mathbf{M}$, $n^{\oplus}$ and $n^{\ominus}$. The full covariance approach with $\mathbf{M} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$ has time complexity $O(n^2 d)$ for construction of $\mathbf{M}$ and $O(d^3)$ for inverting it,[1] so this approach becomes unfeasible with large numbers of features.

7.2 The perceptron

Recall from Chapter 1 that labelled data is called ☞*linearly separable* if there exists a linear decision boundary separating the classes. The least-squares classifier may find a perfectly separating decision boundary if one exists, but this is not guaranteed. To see this, suppose that the basic linear classifier achieves perfect separation for a given training set. Now, move all but one of the positive points away from the negative class. The decision boundary will also move away from the negative class, at some point crossing the one positive that remains fixed. By construction, the modified data is still linearly separable, since the original decision boundary separates it; however, the statistics of the modified data are such that the basic linear classifier will misclassify the one positive outlier.

A linear classifier that will achieve perfect separation on linearly separable data is the *perceptron*, originally proposed as a simple neural network. The perceptron iterates over the training set, updating the weight vector every time it encounters an incorrectly classified example. For example, let $\mathbf{x}_i$ be a misclassified positive example, then we have $y_i = +1$ and $\mathbf{w} \cdot \mathbf{x}_i < t$. We therefore want to find $\mathbf{w}'$ such that $\mathbf{w}' \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$, which moves the decision boundary towards and hopefully past x_i. This can be achieved by calculating the new weight vector as $\mathbf{w}' = \mathbf{w} + \eta \mathbf{x}_i$, where $0 < \eta \leq 1$ is the *learning rate*. We then have $\mathbf{w}' \cdot \mathbf{x}_i = \mathbf{w} \cdot \mathbf{x}_i + \eta \mathbf{x}_i \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$ as required. Similarly, if $\mathbf{x}_j$ is a misclassified

[1] A more sophisticated algorithm can achieve $O(d^{2.8})$, but this is probably the best we can do.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

negative example, then we have $y_j = -1$ and $\mathbf{w}\cdot\mathbf{x}_j > t$. In this case we calculate the new weight vector as $\mathbf{w}' = \mathbf{w} - \eta\mathbf{x}_j$, and thus $\mathbf{w}'\cdot\mathbf{x}_j = \mathbf{w}\cdot\mathbf{x}_j - \eta\mathbf{x}_j\cdot\mathbf{x}_j < \mathbf{w}\cdot\mathbf{x}_j$. The two cases can be combined in a single update rule:

$$\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i \tag{7.8}$$

The perceptron training algorithm is given in Algorithm 7.1. It iterates through the training examples until all examples are correctly classified. The algorithm can easily be turned into an *online* algorithm that processes a stream of examples, updating the weight vector only if the last received example is misclassified. The perceptron is guaranteed to converge to a solution if the training data is linearly separable, but it won't converge otherwise. Figure 7.5 gives a graphical illustration of the perceptron training algorithm. In this particular example I initialised the weight vector to the basic linear classifier, which means the learning rate does have an effect on how quickly we move away from the initial decision boundary. However, if the weight vector is initialised to the zero vector, it is easy to see that the learning rate is just a constant factor that does not affect convergence. We will set it to 1 in the remainder of this section.

The key point of the perceptron algorithm is that, every time an example $\mathbf{x}_i$ is misclassified, we add $y_i\mathbf{x}_i$ to the weight vector. After training has completed, each example has been misclassified zero or more times – denote this number α_i for example $\mathbf{x}_i$.

Algorithm 7.1: Perceptron(D,η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates;
learning rate η.
Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w}\cdot\mathbf{x})$.

```
w ← 0 ;                          // Other initialisations of the weight vector are possible
converged ← false;
while converged = false do
    converged ← true;
    for i = 1 to |D| do
        if y_i w·x_i ≤ 0          // i.e., ŷ_i ≠ y_i
        then
            w ← w + η y_i x_i;
            converged ← false;    // We changed w so haven't converged yet
        end
    end
end
```

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

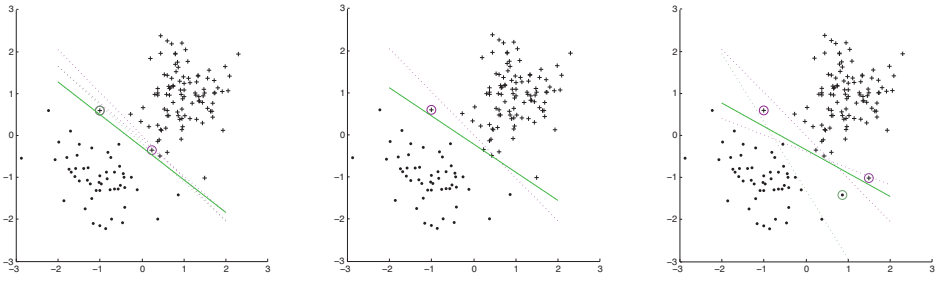

Figure 7.5. (left) A perceptron trained with a small learning rate ($\eta = 0.2$). The circled examples are the ones that trigger the weight update. **(middle)** Increasing the learning rate to $\eta = 0.5$ leads in this case to a rapid convergence. **(right)** Increasing the learning rate further to $\eta = 1$ may lead to too aggressive weight updating, which harms convergence. The starting point in all three cases was the basic linear classifier.

Using this notation the weight vector can be expressed as

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{7.9}$$

In other words, the weight vector is a linear combination of the training instances. The perceptron shares this property with, e.g., the basic linear classifier:

$$\mathbf{w}_{blc} = \boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus} = \frac{1}{Pos} \sum_{\mathbf{x}^{\oplus} \in Tr^{\oplus}} \mathbf{x}^{\oplus} - \frac{1}{Neg} \sum_{\mathbf{x}^{\ominus} \in Tr^{\ominus}} \mathbf{x}^{\ominus} = \sum_{\mathbf{x}^{\oplus} \in Tr^{\oplus}} \alpha^{\oplus} c(\mathbf{x}^{\oplus}) \mathbf{x}^{\oplus} + \sum_{\mathbf{x}^{\ominus} \in Tr^{\ominus}} \alpha^{\ominus} c(\mathbf{x}^{\ominus}) \mathbf{x}^{\ominus} \tag{7.10}$$

Algorithm 7.2: DualPerceptron(D) – perceptron training in dual form.

Input : labelled training data D in homogeneous coordinates.
Output : coefficients α_i defining weight vector $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$.

$\alpha_i \leftarrow 0$ for $1 \leq i \leq |D|$;
converged←false;
while *converged* = false **do**
  converged←true;
  for $i = 1$ to $|D|$ **do**
    if $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j \leq 0$ **then**
      $\alpha_i \leftarrow \alpha_i + 1$;
      converged←false;
    end
  end
end

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

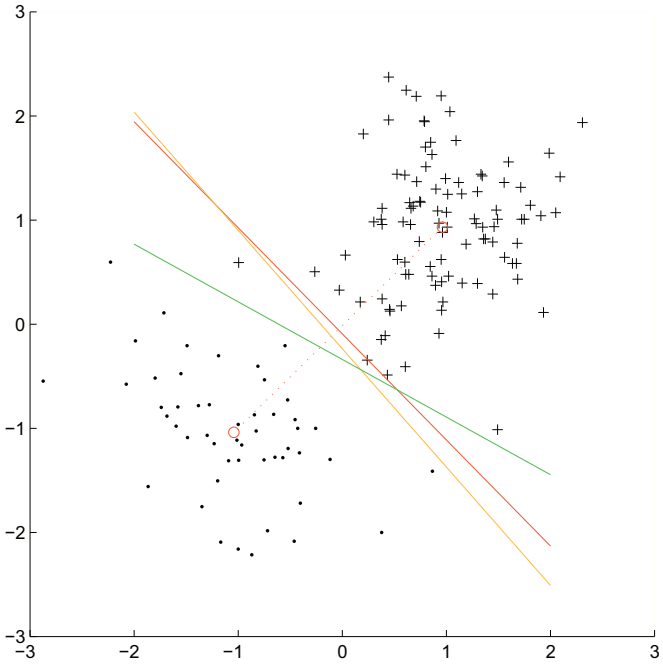

Figure 7.6. Three differently trained linear classifiers on a data set of 100 positives (top-right) and 50 negatives (bottom-left): the basic linear classifier in red, the least-squares classifier in orange and the perceptron in green. Notice that the perceptron perfectly separates the training data, but its heuristic approach may lead to overfitting in certain situations.

where $c(\mathbf{x})$ is the true class of example $\mathbf{x}$ (i.e., $+1$ or -1), $\alpha^{\oplus} = 1/Pos$ and $\alpha^{\ominus} = 1/Neg$. *In the dual, instance-based view of linear classification we are learning instance weights α_i rather than feature weights w_j.* In this dual perspective, an instance $\mathbf{x}$ is classified as $\hat{y} = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x}\right)$. This means that, during training, the only information needed about the training data is all pairwise dot products: the n-by-n matrix $\mathbf{G} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$ containing these dot products is called the *Gram matrix*. Algorithm 7.2 gives the dual form of the perceptron training algorithm. We will encounter this instance-based perspective again when we discuss support vector machines in the next section.

Figure 7.6 demonstrates the difference between the basic linear classifier, the least-squares classifier and the perceptron on some random data. For this particular data set, neither the basic linear classifier nor the least-squares classifier achieves perfect separation, but the perceptron does. One difference with other linear methods is that we cannot derive a closed-form solution for the weight vector learned by the perceptron, so it is a more heuristic approach.

The perceptron can easily be turned into a linear function approximator (Algorithm 7.3). To this end the update rule is changed to $\mathbf{w}' = \mathbf{w} + (y_i - \hat{y}_i)^2 \mathbf{x}_i$, which uses squared

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

residuals. This is unlikely to converge to the exact function, so the algorithm simply runs for a fixed number of training epochs (an epoch is one complete run through the training data). Alternatively, one could run the algorithm until a bound on the sum of squared residuals is reached.

7.3 Support vector machines

Linearly separable data admits infinitely many decision boundaries that separate the classes, but intuitively some of these are better than others. For example, the left and middle decision boundaries in Figure 7.5 seem to be unnecessarily close to some of the positives; while the one on the right leaves a bit more space on either side, it doesn't seem particularly good either. To make this a bit more precise, recall that in Section 2.2 we defined the ☞*margin* of an example assigned by a scoring classifier as $c(x)\hat{s}(x)$, where $c(x)$ is $+1$ for positive examples and -1 for negative examples and $\hat{s}(x)$ is the score of example x. If we take $\hat{s}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - t$, then a true positive $\mathbf{x}_i$ has margin $\mathbf{w} \cdot \mathbf{x}_i - t > 0$ and a true negative $\mathbf{x}_j$ has margin $-(\mathbf{w} \cdot \mathbf{x}_j - t) > 0$. For a given training set and decision boundary, let $m^{\oplus}$ be the smallest margin of any positive, and $m^{\ominus}$ the smallest margin of any negative, then we want the sum of these to be as large as possible. This sum is independent of the decision threshold t, as long as we keep the nearest positives and negatives at the right sides of the decision boundary, and so we re-adjust t such that $m^{\oplus}$ and $m^{\ominus}$ become equal. Figure 7.7 depicts this graphically in a two-dimensional instance space. The training examples nearest to the decision boundary are called *support vectors*: as we shall see, the decision boundary of a support vector machine (SVM) is defined as a linear combination of the support vectors.

The *margin* is thus defined as $m/||\mathbf{w}||$, where m is the distance between the decision boundary and the nearest training instances (at least one of each class) as

Algorithm 7.3: PerceptronRegression(D, T) – train a perceptron for regression.

Input : labelled training data D in homogeneous coordinates;
maximum number of training epochs T.
Output : weight vector $\mathbf{w}$ defining function approximator $\hat{y} = \mathbf{w} \cdot \mathbf{x}$.

```
1 w ←0; t ←0;
2 while t < T do
3     for i = 1 to |D| do
4         w←w + (y_i − ŷ_i)^2 x_i;
5     end
6     t ← t + 1;
7 end
```

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

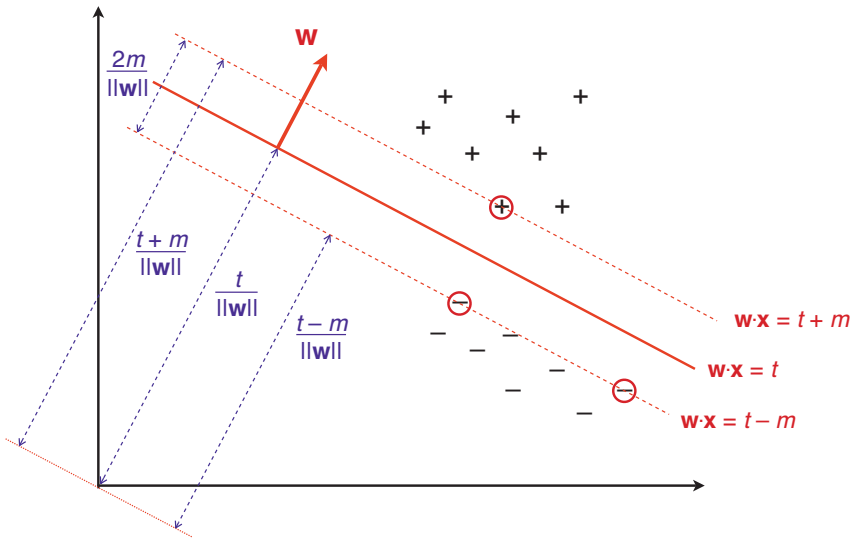

Figure 7.7. The geometry of a support vector classifier. The circled data points are the support vectors, which are the training examples nearest to the decision boundary. The support vector machine finds the decision boundary that maximises the margin $m/||\mathbf{w}||$.

measured along $\mathbf{w}$. Since we are free to rescale t, $||\mathbf{w}||$ and m, it is customary to choose $m = 1$. Maximising the margin then corresponds to minimising $||\mathbf{w}||$ or, more conveniently, $\frac{1}{2}||\mathbf{w}||^2$, provided of course that none of the training points fall inside the margin. This leads to a quadratic, constrained optimisation problem:

$$\mathbf{w}^*, t^* = \arg\min_{\mathbf{w},t} \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1, 1 \leq i \leq n$$

We will approach this using the method of Lagrange multipliers (see Background 7.3). Adding the constraints with multipliers α_i for each training example gives the Lagrange function

$$\begin{aligned}
\Lambda(\mathbf{w}, t, \alpha_1, \ldots, \alpha_n) &= \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i (y_i(\mathbf{w}\cdot\mathbf{x}_i - t) - 1) \\
&= \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i y_i (\mathbf{w}\cdot\mathbf{x}_i) + \sum_{i=1}^{n} \alpha_i y_i t + \sum_{i=1}^{n} \alpha_i \\
&= \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \mathbf{w}\cdot\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right) + t\left(\sum_{i=1}^{n} \alpha_i y_i\right) + \sum_{i=1}^{n} \alpha_i
\end{aligned}$$

While this looks like a formidable formula, some further analysis will allow us to derive the simpler dual form of the Lagrange function.

By taking the partial derivative of the Lagrange function with respect to t and setting it to 0 we find that for the optimal threshold t we have $\sum_{i=1}^{n} \alpha_i y_i = 0$. Similarly, by

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

Optimisation is a broad term denoting the problem of finding the best item or value among a set of alternatives. We have already seen a very simple, unconstrained form of optimisation in Example 7.1 on p.197, where we found the values of a and b minimising the sum of squared residuals $f(a,b) = \sum_{i=1}^{n}(w_i - (a + bh_i))^2$; this can be denoted as

$$a^*, b^* = \underset{a,b}{\arg\min}\, f(a,b)$$

f is called the *objective function*; it can be linear, quadratic (as in this case), or more complex. We found the minimum of f by setting the partial derivatives of f with respect to a and b to 0, and solving for a and b; the vector of these partial derivatives is called the *gradient* and denoted ∇f, so a succinct way of defining the unconstrained optimisation problem is: find a and b such that $\nabla f(a,b) = \mathbf{0}$. In this particular case the objective function is *convex*, which essentially means that there is a unique global minimum. This is, however, not always the case.

A *constrained optimisation* problem is one where the alternatives are subject to constraints, for instance

$$a^*, b^* = \underset{a,b}{\arg\min}\, f(a,b) \qquad \text{subject to } g(a,b) = c$$

If the relationship expressed by the constraint is linear, say $a - b = 0$, we can of course eliminate one of the variables and solve the simpler, unconstrained problem. However, this may not be possible if the constraints are non-linear. *Lagrange multipliers* are a powerful way of dealing with the general case. We form the Lagrange function defined by

$$\Lambda(a,b,\lambda) = f(a,b) - \lambda(g(a,b) - c)$$

where λ is the Lagrange multiplier, and solve the unconstrained problem $\nabla\Lambda(a,b,\lambda) = \mathbf{0}$. Since $\nabla_{a,b}\Lambda(a,b,\lambda) = \nabla f(a,b) - \lambda\nabla g(a,b)$ and $\nabla_{\lambda}\Lambda(a,b,\lambda) = g(a,b) - c$, this is a succinct way of requiring (*i*) that the gradients of f and g point in the same direction, and (*ii*) that the constraint is satisfied. We can include multiple equality constraints and also inequality constraints, each with their own Lagrange multiplier.

From the Lagrange function it is possible to derive a *dual* optimisation problem where we find the optimal values of the Lagrange multipliers. In general, the solution to the dual problem is only a lower bound on the solution to the *primal* problem, but under a set of conditions known as the *Karush–Kuhn–Tucker conditions* (*KKT*) the two solutions become equal. The quadratic optimisation problem posed by support vector machines is usually solved in its dual form.

Background 7.3. Basic concepts and terminology in mathematical optimisation.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

taking the partial derivative of the Lagrange function with respect to $\mathbf{w}$ we see that the Lagrange multipliers define the weight vector as a linear combination of the training examples:

$$\frac{\partial}{\partial \mathbf{w}}\Lambda(\mathbf{w}, t, \alpha_1, \ldots, \alpha_n) = \frac{\partial}{\partial \mathbf{w}}\frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \frac{\partial}{\partial \mathbf{w}}\mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i\right) = \mathbf{w} - \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i$$

Since this partial derivative is 0 for an optimal weight vector we conclude $\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i$ – the same expression as we derived for the perceptron in Equation 7.9 on p.209. For the perceptron, the instance weights α_i are non-negative integers denoting the number of times an example has been misclassified in training. For a support vector machine, the α_i are non-negative reals. What they have in common is that, if $\alpha_i = 0$ for a particular example $\mathbf{x}_i$, that example could be removed from the training set without affecting the learned decision boundary. In the case of support vector machines this means that $\alpha_i > 0$ only for the support vectors: the training examples nearest to the decision boundary.

Now, by plugging the expressions $\sum_{i=1}^{n}\alpha_i y_i = 0$ and $\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i$ back into the Lagrangian we are able to eliminate $\mathbf{w}$ and t, and hence obtain the dual optimisation problem, which is entirely formulated in terms of the Lagrange multipliers:

$$\begin{aligned}\Lambda(\alpha_1, \ldots, \alpha_n) &= -\frac{1}{2}\left(\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i\right)\cdot\left(\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i\right) + \sum_{i=1}^{n}\alpha_i \\ &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n}\alpha_i\end{aligned}$$

The dual problem is to maximise this function under positivity constraints and one equality constraint:

$$\alpha_1^*, \ldots, \alpha_n^* = \operatorname*{arg\,max}_{\alpha_1,\ldots,\alpha_n} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$
$$\text{subject to } \alpha_i \geq 0, 1 \leq i \leq n \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

The dual form of the optimisation problem for support vector machines illustrates two important points. First, it shows that searching for the maximum-margin decision boundary is equivalent to searching for the support vectors: they are the training examples with non-zero Lagrange multipliers, and through $\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i$ they completely determine the decision boundary. Secondly, it shows that the optimisation problem is entirely defined by pairwise dot products between training instances: the entries of the Gram matrix. As we shall see in Section 7.5, this paves the way for a powerful adaptation of support vector machines that allows them to learn non-linear decision boundaries.

The following example makes these issues a bit more concrete by showing detailed calculations on some toy data.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

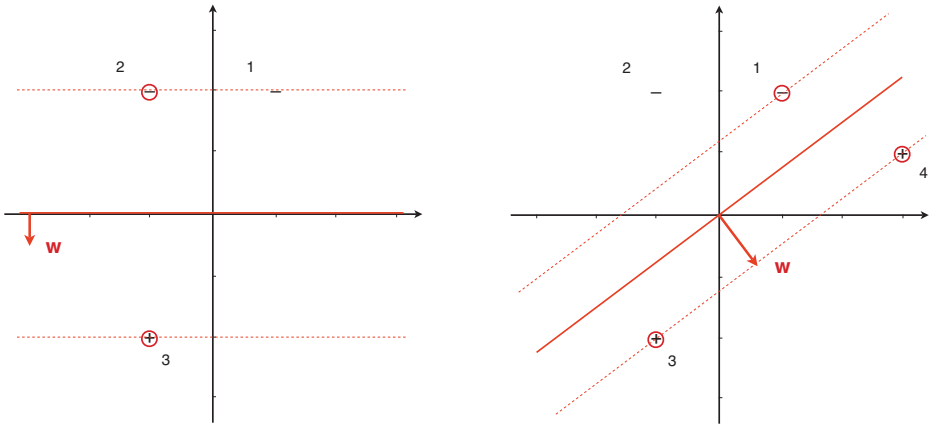

Figure 7.8. **(left)** A maximum-margin classifier built from three examples, with $\mathbf{w} = (0, -1/2)$ and margin 2. The circled examples are the support vectors: they receive non-zero Lagrange multipliers and define the decision boundary. **(right)** By adding a second positive the decision boundary is rotated to $\mathbf{w} = (3/5, -4/5)$ and the margin decreases to 1.

Example 7.5 (Two maximum-margin classifiers and their support vectors). Let the data points and labels be as follows (see Figure 7.8 (left)):

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \qquad \mathbf{X}' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

The matrix $\mathbf{X}'$ on the right incorporates the class labels; i.e., the rows are $y_i \mathbf{x}_i$. The Gram matrix is (without and with class labels):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & -5 \\ 3 & 5 & -3 \\ -5 & -3 & 5 \end{pmatrix} \qquad \mathbf{X}'\mathbf{X}'^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{pmatrix}$$

The dual optimisation problem is thus

$$\begin{aligned} &\operatorname*{arg\,max}_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(5\alpha_1^2 + 3\alpha_1\alpha_2 + 5\alpha_1\alpha_3 + 3\alpha_2\alpha_1 + 5\alpha_2^2 + 3\alpha_2\alpha_3 + 5\alpha_3\alpha_1 \right. \\ &\qquad\qquad \left. + 3\alpha_3\alpha_2 + 5\alpha_3^2\right) + \alpha_1 + \alpha_2 + \alpha_3 \\ &= \operatorname*{arg\,max}_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(5\alpha_1^2 + 6\alpha_1\alpha_2 + 10\alpha_1\alpha_3 + 5\alpha_2^2 + 6\alpha_2\alpha_3 + 5\alpha_3^2\right) + \alpha_1 + \alpha_2 + \alpha_3 \end{aligned}$$

subject to $\alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0$ and $-\alpha_1 - \alpha_2 + \alpha_3 = 0$. While in practice such problems are solved by dedicated quadratic optimisation solvers, here we will show how to solve this toy problem by hand.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

Using the equality constraint we can eliminate one of the variables, say α_3, and simplify the objective function to

$$\arg\max_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(5\alpha_1^2+6\alpha_1\alpha_2+10\alpha_1(\alpha_1+\alpha_2)+5\alpha_2^2+6\alpha_2(\alpha_1+\alpha_2)+5(\alpha_1+\alpha_2)^2\right) +2\alpha_1+2\alpha_2$$

$$= \arg\max_{\alpha_1,\alpha_2,\alpha_3} -\frac{1}{2}\left(20\alpha_1^2+32\alpha_1\alpha_2+16\alpha_2^2\right)+2\alpha_1+2\alpha_2$$

Setting partial derivatives to 0 we obtain $-20\alpha_1-16\alpha_2+2=0$ and $-16\alpha_1-16\alpha_2+2=0$ (notice that, because the objective function is quadratic, these equations are guaranteed to be linear). We therefore obtain the solution $\alpha_1=0$ and $\alpha_2=\alpha_3=1/8$. We then have $\mathbf{w}=1/8(\mathbf{x}_3-\mathbf{x}_2)=\begin{pmatrix}0\\-1/2\end{pmatrix}$, resulting in a margin of $1/||\mathbf{w}||=2$. Finally, t can be obtained from any support vector, say $\mathbf{x}_2$, since $y_2(\mathbf{w}\cdot\mathbf{x}_2-t)=1$; this gives $-1\cdot(-1-t)=1$, hence $t=0$. The resulting maximum-margin classifier is depicted in Figure 7.8 (left). Notice that the first example $\mathbf{x}_1$ is not a support vector, even though it is on the margin: this is because removing it will not affect the decision boundary.

We now add an additional positive at $(3,1)$. This gives the following data matrices:

$$\mathbf{X}'=\begin{pmatrix}-1&-2\\1&-2\\-1&-2\\3&1\end{pmatrix} \qquad \mathbf{X}'\mathbf{X}'^{\mathrm{T}}=\begin{pmatrix}5&3&5&-5\\3&5&3&1\\5&3&5&-5\\-5&1&-5&10\end{pmatrix}$$

It can be verified by similar calculations to those above that the margin decreases to 1 and the decision boundary rotates to $\mathbf{w}=\begin{pmatrix}3/5\\-4/5\end{pmatrix}$ (Figure 7.8 (right)). The Lagrange multipliers now are $\alpha_1=1/2$, $\alpha_2=0$, $\alpha_3=1/10$ and $\alpha_4=2/5$. Thus, only $\mathbf{x}_3$ is a support vector in both the original and the extended data set.

Soft margin SVM

If the data is not linearly separable, then the constraints $\mathbf{w}\cdot\mathbf{x}_i-t\geq 1$ posed by the examples are not jointly satisfiable. However, there is a very elegant way of adapting the optimisation problem such that it admits a solution even in this case. The idea is to introduce *slack variables* ξ_i, one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary – we will call these *margin errors*. Thus, we change the constraints to $\mathbf{w}\cdot\mathbf{x}_i-t\geq 1-\xi_i$ and add the sum of

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

all slack variables to the objective function to be minimised, resulting in the following *soft margin* optimisation problem:

$$\mathbf{w}^*, t^*, \xi_i^* = \arg\min_{\mathbf{w},t,\xi_i} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1-\xi_i \text{ and } \xi_i \geq 0, 1 \leq i \leq n \tag{7.11}$$

C is a user-defined parameter trading off margin maximisation against slack variable minimisation: a high value of C means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin. If we allow more margin errors we need fewer support vectors, hence C controls to some extent the 'complexity' of the SVM and hence is often referred to as the *complexity parameter*. It can be seen as a form of regularisation similar to that discussed in the context of least-squares regression.

The Lagrange function is then as follows:

$$\begin{aligned}\Lambda(\mathbf{w}, t, \xi_i, \alpha_i, \beta_i) &= \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}\cdot\mathbf{x}_i - t) - (1-\xi_i)) - \sum_{i=1}^{n}\beta_i\xi_i \\ &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \mathbf{w}\cdot\left(\sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i\right) + t\left(\sum_{i=1}^{n}\alpha_i y_i\right) + \sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}(C-\alpha_i-\beta_i)\xi_i \\ &= \Lambda(\mathbf{w}, t, \alpha_i) + \sum_{i=1}^{n}(C-\alpha_i-\beta_i)\xi_i\end{aligned}$$

For an optimal solution every partial derivative with respect to ξ_i should be 0, from which it follows that $C - \alpha_i - \beta_i = 0$ for all i, and hence the added term vanishes from the dual problem. Furthermore, since both α_i and β_i are positive, this means that α_i cannot be larger than C, which manifests itself as an additional upper bound on α_i in the dual problem:

$$\alpha_1^*, \ldots, \alpha_n^* = \arg\max_{\alpha_1,\ldots,\alpha_n} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\cdot\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$
$$\text{subject to } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{7.12}$$

This is a remarkable and beautiful result. It follows from the particular way that slack variables were added to the optimisation problem in Equation 7.11. By restricting the slack variables to be positive and adding them to the objective function to be minimised, they function as penalty terms, measuring deviations on the wrong side of the margin only. Furthermore, the fact that the β_i multipliers do not appear in the dual objective follows from the fact that the penalty term in the primal objective is linear in ξ_i. In effect, these slack variables implement what was called *hinge loss* in Figure 2.6 on p.63: a margin $z > 1$ incurs no penalty, and a margin $z = 1 - \xi \leq 1$ incurs a penalty $\xi = 1 - z$.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

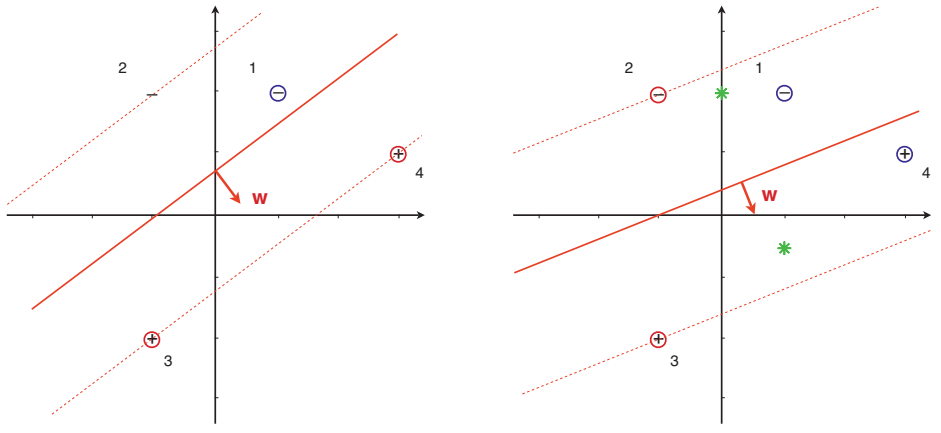

Figure 7.9. (left) The soft margin classifier learned with $C = 5/16$, at which point $\mathbf{x}_2$ is about to become a support vector. **(right)** The soft margin classifier learned with $C = 1/10$: all examples contribute equally to the weight vector. The asterisks denote the class means, and the decision boundary is parallel to the one learned by the basic linear classifier.

What is the significance of the upper bound C on the α_i multipliers? Since $C - \alpha_i - \beta_i = 0$ for all i, $\alpha_i = C$ implies $\beta_i = 0$. The β_i multipliers come from the $\xi_i \geq 0$ constraint, and a multiplier of 0 means that the lower bound is not reached, i.e., $\xi_i > 0$ (analogous to the fact that $\alpha_j = 0$ means that $\mathbf{x}_j$ is not a support vector and hence $\mathbf{w} \cdot \mathbf{x}_j - t > 1$). In other words, a solution to the soft margin optimisation problem in dual form divides the training examples into three cases:

$\alpha_i = 0$	these are outside or on the margin;
$0 < \alpha_i < C$	these are the support vectors on the margin;
$\alpha_i = C$	these are on or inside the margin.

Notice that we still have $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$, and so both second and third case examples participate in spanning the decision boundary.

Example 7.6 (Soft margins). We continue Example 7.5, where we saw that adding the positive example $\mathbf{x}_4 = (3, 1)$ to the first three examples significantly reduced the margin from 2 to 1. We will now show that soft margin classifiers with larger margins are learned with sufficiently large complexity parameter C.

Recall that the Lagrange multipliers for the classifier in Figure 7.8 (right) are $\alpha_1 = 1/2$, $\alpha_2 = 0$, $\alpha_3 = 1/10$ and $\alpha_4 = 2/5$. So α_1 is the largest multiplier, and as long as $C > \alpha_1 = 1/2$ no margin errors are tolerated. For $C = 1/2$ we have $\alpha_1 = C$,

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

and hence for $C < 1/2$ we have that $\mathbf{x}_1$ becomes a margin error and the optimal classifier is a soft margin classifier. Effectively, with decreasing C the decision boundary and the upper margin move upward, while the lower margin stays the same.

The upper margin reaches $\mathbf{x}_2$ for $C = 5/16$ (Figure 7.9 (left)), at which point we have $\mathbf{w} = \begin{pmatrix} 3/8 \\ -1/2 \end{pmatrix}$, $t = 3/8$ and the margin has increased to 1.6. Furthermore, we have $\xi_1 = 6/8, \alpha_1 = C = 5/16, \alpha_2 = 0, \alpha_3 = 1/16$ and $\alpha_4 = 1/4$.

If we now decrease C further, the decision boundary starts to rotate clockwise, so that $\mathbf{x}_4$ becomes a margin error as well, and only $\mathbf{x}_2$ and $\mathbf{x}_3$ are support vectors. The boundary rotates until $C = 1/10$, at which point we have $\mathbf{w} = \begin{pmatrix} 1/5 \\ -1/2 \end{pmatrix}$, $t = 1/5$ and the margin has increased to 1.86. Furthermore, we have $\xi_1 = 4/10$ and $\xi_4 = 7/10$, and all multipliers have become equal to C (Figure 7.9 (right)).

Finally, when C decreases further the decision boundary stays where it is, but the norm of the weight vector gradually decreases and all points become margin errors.

Example 7.6 illustrates an important point: for low enough C, all examples receive the same multiplier C, and hence we have $\mathbf{w} = C\sum_{i=1}^{n} y_i\mathbf{x}_i = C(Pos \cdot \boldsymbol{\mu}^{\oplus} - Neg \cdot \boldsymbol{\mu}^{\ominus})$, where $\boldsymbol{\mu}^{\oplus}$ and $\boldsymbol{\mu}^{\ominus}$ are the means of the positive and negative examples, respectively. In other words, *a minimal-complexity soft margin classifier summarises the classes by their class means in a way very similar to the basic linear classifier*. For intermediate values of C the decision boundary is spanned by the support vectors and the per-class means of the margin errors.

In summary, support vector machines are linear classifiers that construct the unique decision boundary that maximises the distance to the nearest training examples (the support vectors). The complexity parameter C can be used to adjust the number and severity of allowed margin violations. Training an SVM involves solving a large quadratic optimisation problem and is usually best left to a dedicated numerical solver.

7.4 Obtaining probabilities from linear classifiers

As we have seen, a linear classifier produces scores $\hat{s}(\mathbf{x}_i) = \mathbf{w}\cdot\mathbf{x}_i - t$ that are thresholded at 0 in order to classify examples. Owing to the geometric nature of linear classifiers, such scores can be used to obtain the (signed) distance of $\mathbf{x}_i$ from the decision boundary. To see this, notice that the length of the projection of $\mathbf{x}_i$ onto $\mathbf{w}$ is $||\mathbf{x}_i|| \cos\theta$, where

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

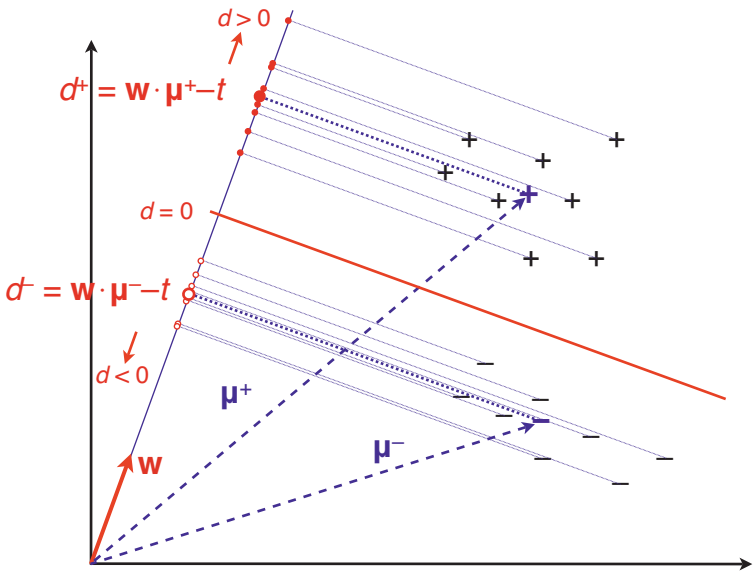

Figure 7.10. We can think of a linear classifier as a projection onto the direction given by $\mathbf{w}$, here assumed to be a unit vector. $\mathbf{w}\cdot\mathbf{x}-t$ gives the signed distance from the decision boundary on the projection line. Also indicated are the class means $\boldsymbol{\mu}^{\oplus}$ and $\boldsymbol{\mu}^{\ominus}$, and the corresponding mean distances $d^{\oplus}$ and $d^{\ominus}$.

θ is the angle between $\mathbf{x}_i$ and $\mathbf{w}$. Since $\mathbf{w}\cdot\mathbf{x}_i = ||\mathbf{w}||\,||\mathbf{x}_i||\cos\theta$, we can write this length as $(\mathbf{w}\cdot\mathbf{x}_i)/||\mathbf{w}||$. This gives the following signed distance:

$$d(\mathbf{x}_i) = \frac{\hat{s}(\mathbf{x}_i)}{||\mathbf{w}||} = \frac{\mathbf{w}\cdot\mathbf{x}_i - t}{||\mathbf{w}||} = \mathbf{w}'\cdot\mathbf{x}_i - t'$$

with $\mathbf{w}' = \mathbf{w}/||\mathbf{w}||$ rescaled to unit length and $t' = t/||\mathbf{w}||$ the corresponding rescaled intercept. The sign of this quantity tells us which side of the decision boundary we are on: positive distances for points on the 'positive' side of the decision boundary (the direction in which $\mathbf{w}$ points) and negative distances on the other side (Figure 7.10).

This geometric interpretation of the scores produced by linear classifiers offers an interesting possibility for turning them into probabilities, a process that was called ☞*calibration* in Section 2.3. Let $\overline{d}^{\oplus}$ denote the mean distance of the positive examples to the decision boundary: i.e., $\overline{d}^{\oplus} = \mathbf{w}\cdot\boldsymbol{\mu}^{\oplus} - t$, where $\boldsymbol{\mu}^{\oplus}$ is the mean of the positive examples and $\mathbf{w}$ is unit length (although the latter assumption is not strictly necessary, as it will turn out that the weight vector will be rescaled). It would not be unreasonable to expect that the distance of positive examples to the decision boundary is normally distributed around this mean:[2] that is, when plotting a histogram of these distances,

[2]For instance, with sufficiently many examples this could be justified by the *central limit theorem*: the sum of a large number of identically distributed independent random variables is approximately normally distributed.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

we would expect the familiar bell curve to appear. Under this assumption, the probability density function of d is $P(d|\oplus) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(d-\overline{d}^{\oplus})^2}{2\sigma^2}\right)$ (see Background 9.1 on p.267 if you need to remind yourself about the normal distribution). Similarly, the distances of negative examples to the decision boundary can be expected to be normally distributed around $\overline{d}^{\ominus} = \mathbf{w}\cdot\boldsymbol{\mu}^{\ominus} - t$, with $\overline{d}^{\ominus} < 0 < \overline{d}^{\oplus}$. We will assume that both normal distributions have the same variance σ^2.

Suppose we now observe a point $\mathbf{x}$ with distance $d(\mathbf{x})$. We classify this point as positive if $d(\mathbf{x}) > 0$ and as negative if $d(\mathbf{x}) < 0$, but we want to attach a probability $\hat{p}(\mathbf{x}) = P(\oplus|d(\mathbf{x}))$ to these predictions. Using Bayes' rule we obtain

$$P(\oplus|d(\mathbf{x})) = \frac{P(d(\mathbf{x})|\oplus)P(\oplus)}{P(d(\mathbf{x})|\oplus)P(\oplus) + P(d(\mathbf{x})|\ominus)P(\ominus)} = \frac{LR}{LR + 1/clr}$$

where LR is the likelihood ratio obtained from the normal score distributions, and clr is the class ratio. We will assume for simplicity that $clr = 1$ in the derivation below. Furthermore, assume for now that $\sigma^2 = 1$ and $\overline{d}^{\oplus} = -\overline{d}^{\ominus} = 1/2$ (we will relax this in a moment). We then have

$$\begin{aligned} LR &= \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \frac{\exp\left(-(d(\mathbf{x}) - 1/2)^2/2\right)}{\exp\left(-(d(\mathbf{x}) + 1/2)^2/2\right)} \\ &= \exp\left(-(d(\mathbf{x}) - 1/2)^2/2 + (d(\mathbf{x}) + 1/2)^2/2\right) = \exp\left(d(\mathbf{x})\right) \end{aligned}$$

and so

$$P(\oplus|d(\mathbf{x})) = \frac{\exp\left(d(\mathbf{x})\right)}{\exp\left(d(\mathbf{x})\right) + 1} = \frac{\exp\left(\mathbf{w}\cdot\mathbf{x} - t\right)}{\exp\left(\mathbf{w}\cdot\mathbf{x} - t\right) + 1}$$

So, in order to obtain probability estimates from a linear classifier outputting distance scores d, we convert d into a probability by means of the mapping $d \mapsto \frac{\exp(d)}{\exp(d)+1}$ (or, equivalently, $d \mapsto \frac{1}{1+\exp(-d)}$). This S-shaped or *sigmoid* function is called the *logistic function*; it finds applications in a wide range of areas (Figure 7.11).

Suppose now that $\overline{d}^{\oplus} = -\overline{d}^{\ominus}$ as before, but we do not assume anything about the magnitude of these mean distances or of σ^2. In this case we have

$$\begin{aligned} LR &= \exp\left(\frac{-(d(\mathbf{x}) - \overline{d}^{\oplus})^2 + (d(\mathbf{x}) - \overline{d}^{\ominus})^2}{2\sigma^2}\right) \\ &= \exp\left(\frac{2\overline{d}^{\oplus}d(\mathbf{x}) - \left(\overline{d}^{\oplus}\right)^2 - 2\overline{d}^{\ominus}d(\mathbf{x}) + \left(\overline{d}^{\ominus}\right)^2}{2\sigma^2}\right) = \exp\left(\gamma d(\mathbf{x})\right) \end{aligned}$$

with $a = (\overline{d}^{\oplus} - \overline{d}^{\ominus})/\sigma^2$ a scaling factor that rescales the weight vector so that the mean distances per class are one unit of variance apart. In other words, by taking the scaling factor γ into account, we can drop our assumption that $\mathbf{w}$ is a unit vector.

If we also drop the assumption that $\overline{d}^{\oplus}$ and $\overline{d}^{\ominus}$ are symmetric around the decision

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

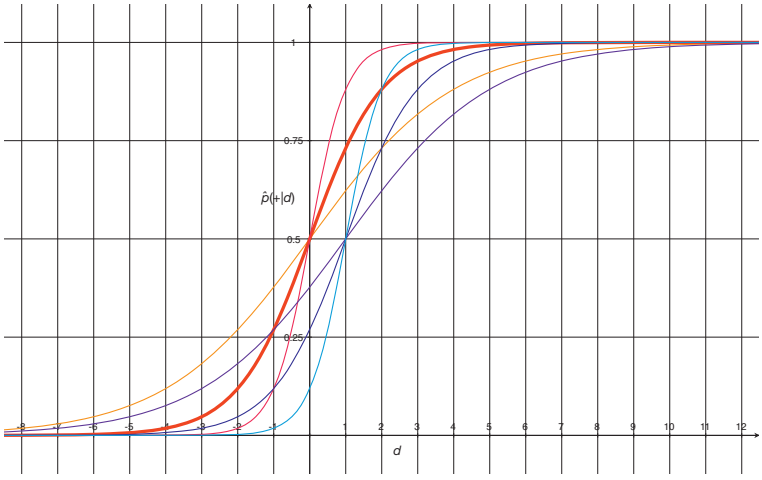

Figure 7.11. The logistic function, a useful function for mapping distances from a linear decision boundary into an estimate of the positive posterior probability. The **fat** red line indicates the standard logistic function $\hat{p}(d) = \frac{1}{1+\exp(-d)}$; this function can be used to obtain probability estimates if the two classes are equally prevalent and the class means are equidistant from the decision boundary and one unit of variance apart. The steeper and flatter red lines show how the function changes if the class means are 2 and 1/2 units of variance apart, respectively. The three blue lines show how these curves change if $d_0 = 1$, which means that the positives are on average further away from the decision boundary.

boundary, then we obtain the most general form

$$LR = \frac{P(d(\mathbf{x})|\oplus)}{P(d(\mathbf{x})|\ominus)} = \exp\big(\gamma(d(\mathbf{x}) - d_0)\big) \tag{7.13}$$

$$\gamma = \frac{\overline{d}^{\oplus} - \overline{d}^{\ominus}}{\sigma^2} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{\sigma^2}, \quad d_0 = \frac{\overline{d}^{\oplus} + \overline{d}^{\ominus}}{2} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})}{2} - t$$

d_0 has the effect of moving the decision boundary from $\mathbf{w}\cdot\mathbf{x} = t$ to $\mathbf{x} = (\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$, that is, halfway between the two class means. The logistic mapping thus becomes $d \mapsto \frac{1}{1+\exp(-\gamma(d-d_0))}$, and the effect of the two parameters is visualised in Figure 7.11.

Example 7.7 (Logistic calibration of a linear classifier). Logistic calibration has a particularly simple form for the basic linear classifier, which has $\mathbf{w} = \boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}$. It follows that

$$\overline{d}^{\oplus} - \overline{d}^{\ominus} = \frac{\mathbf{w}\cdot(\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus})}{||\mathbf{w}||} = \frac{||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||^2}{||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||} = ||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||$$

and hence $\gamma = ||\boldsymbol{\mu}^{\oplus} - \boldsymbol{\mu}^{\ominus}||/\sigma^2$. Furthermore, $d_0 = 0$ as $(\boldsymbol{\mu}^{\oplus} + \boldsymbol{\mu}^{\ominus})/2$ is already on the decision boundary. So in this case logistic calibration does not move the

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

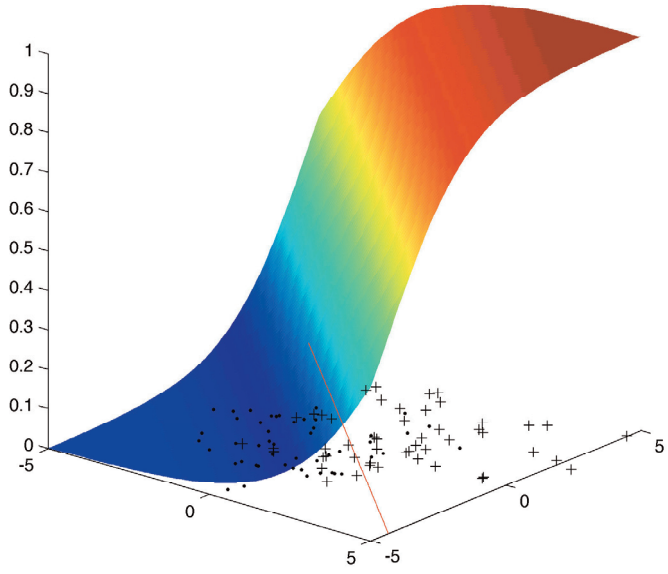

Figure 7.12. The surface shows the sigmoidal probability estimates resulting from logistic calibration of the basic linear classifier on random data satisfying the assumptions of logistic calibration.

decision boundary, and only adjusts the steepness of the sigmoid according to the separation of the classes. Figure 7.12 illustrates this for some data sampled from two normal distributions with the same diagonal covariance matrix.

To summarise: in order to get calibrated probability estimates out of a linear classifier, we first calculate the mean distances $\overline{d}^{\oplus}$ and $\overline{d}^{\ominus}$ and the variance σ^2, and from those the location parameter d_0 and the scaling parameter γ. The likelihood ratio is then $LR = \exp\big(\gamma(d(\mathbf{x}) - d_0)\big) = \exp\big(\gamma(\mathbf{w}\cdot\mathbf{x} - t - d_0)\big)$. Since the logarithm of the likelihood ratio is linear in $\mathbf{x}$, such models are called *log-linear models*. Notice that $\gamma(\mathbf{w}\cdot\mathbf{x} - t - d_0) = \mathbf{w}'\cdot\mathbf{x} - t'$ with $\mathbf{w}' = \gamma\mathbf{w}$ and $t' = \gamma(t + d_0)$. This means that the logistic calibration procedure can change the location of the decision boundary but not its direction. However, there may be an alternative weight vector with a different direction that assigns a higher likelihood to the data. Finding the maximum-likelihood linear classifier using the logistic model is called ☞*logistic regression*, and will be discussed in Section 9.3.

As an alternative to logistic calibration, we can also use the isotonic calibration

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

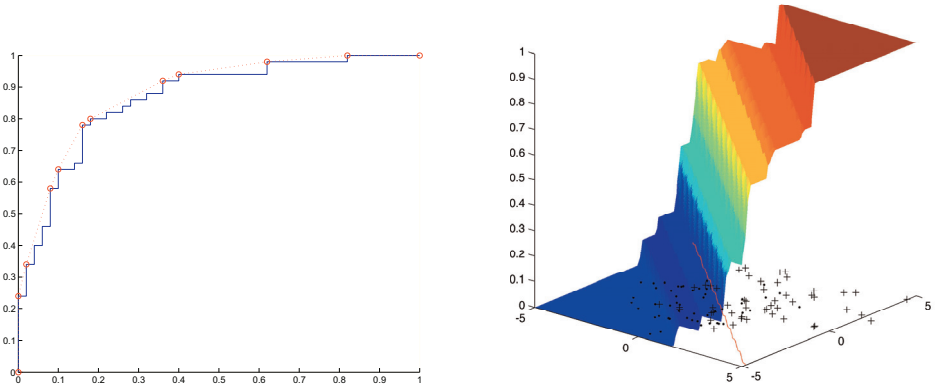

Figure 7.13. (left) ROC curve and convex hull of the same model and data as in Figure 7.12. **(right)** The convex hull can be used as a non-parametric calibration method. Each segment of the convex hull corresponds to a plateau of the probability surface.

method discussed in Section 2.3. Figure 7.13 (left) shows the ROC curve of the basic linear classifier on the data in Figure 7.12 as well as its convex hull. We can then construct a piecewise linear calibration function with plateaus corresponding to the convex hull segments, as shown in Figure 7.13 (right). In contrast with the logistic method this calibration method is non-parametric and hence does not make any assumptions about the data. In order to avoid overfitting, non-parametric methods typically need more data than parametric methods. It is interesting to note that no grading takes place on the plateaus, which are rather similar to the segments of a grouping model. In other words, convex hull calibration can potentially produce a hybrid between grouping and grading models.

7.5 Going beyond linearity with kernel methods

In this chapter we have looked at linear methods for classification and regression. Starting with the least-squares method for regression, we have seen how to adapt it to binary classification, resulting in a version of the basic linear classifier that takes feature correlation into account by constructing the matrix $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ and is sensitive to unequal class distributions. We then looked at the heuristic perceptron algorithm for linearly separable data, and the support vector machine which finds the unique decision boundary with maximum margin and which can be adapted to non-separable data. In this section we show that these techniques can be adapted to learn non-linear decision boundaries. The main idea is simple (and was already explored in Example 1.9 on p.43): to transform the data non-linearly to a feature space in which linear classification can be applied.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

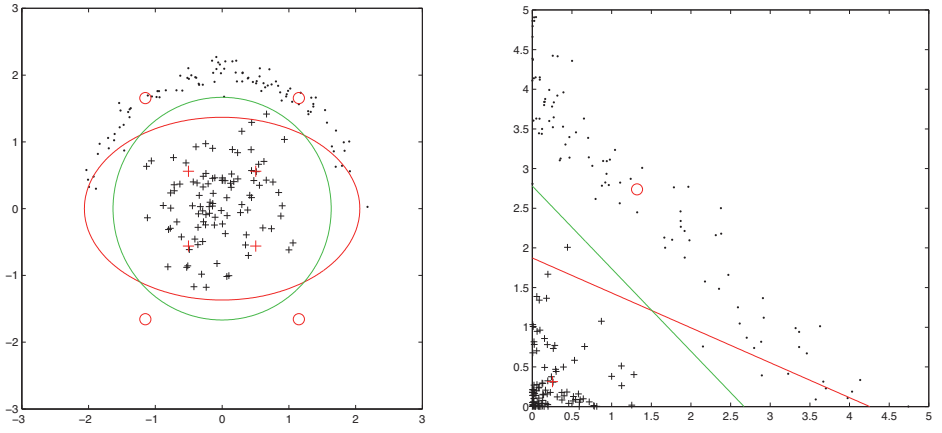

Figure 7.14. (left) Decision boundaries learned by the basic linear classifier and the perceptron using the square of the features. **(right)** Data and decision boundaries in the transformed feature space.

Example 7.8 (Learning a quadratic decision boundary). The data in Figure 7.14 (left) is not linearly separable, but both classes have a clear circular shape. Figure 7.14 (right) shows the same data with the feature values squared. In this transformed feature space the data has become linearly separable, and the perceptron is able to separate the classes. The resulting decision boundary in the original space is a near-circle. Also shown is the decision boundary learned by the basic linear classifier in the quadratic feature space, corresponding to an ellipse in the original space.

In general, mapping points back from the feature space to the instance space is non-trivial. E.g., in this example each class mean in feature space maps back to four points in the original space, owing to the quadratic mapping.

It is customary to call the transformed space the *feature space* and the original space the *input space*. The approach thus appears to be to transform the training data to feature space and learn a model there. In order to classify new data we transform that to feature space as well and apply the model. However, the remarkable thing is that in many cases the feature space does not have to be explicitly constructed, as we can perform all necessary operations in input space.

Take the perceptron algorithm in dual form, for example (Algorithm 7.2 on p.209). The algorithm is a simple counting algorithm – the only operation that is somewhat involved is testing whether example $\mathbf{x}_i$ is correctly classified by evaluating $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot$

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

$\mathbf{x}_j$. The key component of this calculation is the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$. Assuming bivariate examples $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}_j = (x_j, y_j)$ for notational simplicity, the dot product can be written as $\mathbf{x}_i \cdot \mathbf{x}_j = x_i x_j + y_i y_j$. The corresponding instances in the quadratic feature space are (x_i^2, y_i^2) and $\left(x_j^2, y_j^2\right)$, and their dot product is

$$(x_i^2, y_i^2) \cdot \left(x_j^2, y_j^2\right) = x_i^2 x_j^2 + y_i^2 y_j^2$$

This is almost equal to

$$(\mathbf{x}_i \cdot \mathbf{x}_j)^2 = (x_i x_j + y_i y_j)^2 = (x_i x_j)^2 + (y_i y_j)^2 + 2 x_i x_j y_i y_j$$

but not quite because of the third term of cross-products. We can capture this term by extending the feature vector with a third feature $\sqrt{2}xy$. This gives the following feature space:

$$\phi(\mathbf{x}_i) = \left(x_i^2, y_i^2, \sqrt{2} x_i y_i\right) \qquad \phi(\mathbf{x}_j) = \left(x_j^2, y_j^2, \sqrt{2} x_j y_j\right)$$
$$\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = x_i^2 x_j^2 + y_i^2 y_j^2 + 2 x_i x_j y_i y_j = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

We now define $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$, and replace $\mathbf{x}_i \cdot \mathbf{x}_j$ with $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ in the dual perceptron algorithm to obtain the *kernel perceptron* (Algorithm 7.4), which is able to learn the kind of non-linear decision boundaries illustrated in Example 7.8.

The introduction of kernels opens up a whole range of possibilities. Clearly we can define a polynomial kernel of any degree p as $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p$. This transforms

Algorithm 7.4: KernelPerceptron(D, κ) – perceptron training algorithm using a kernel.

Input : labelled training data D in homogeneous coordinates;
kernel function κ.

Output : coefficients α_i defining non-linear decision boundary.

```
1  α_i ← 0 for 1 ≤ i ≤ |D|;
2  converged←false;
3  while converged = false do
4      converged←true;
5      for i = 1 to |D| do
6          if y_i Σ_{j=1}^{|D|} α_j y_j κ(x_i, x_j) ≤ 0 then
7              α_i ← α_i + 1;
8              converged←false;
9          end
10     end
11 end
```

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

a d-dimensional input space into a high-dimensional feature space, such that each new feature is a product of p terms (possibly repeated). If we include a constant, say $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^p$, we would get all lower-order terms as well. So, for example, in a bivariate input space and setting $p = 2$ the resulting feature space is

$$\phi(\mathbf{x}) = \left(x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1\right)$$

with linear as well as quadratic features.

But we are not restricted to polynomial kernels. An often-used kernel is the *Gaussian kernel*, defined as

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{7.14}$$

where σ is a parameter known as the *bandwidth*. To understand the Gaussian kernel a bit better, notice that $\kappa(\mathbf{x}, \mathbf{x}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}) = ||\phi(\mathbf{x})||^2$ for any kernel obeying a number of standard properties referred to as 'positive semi-definiteness'. In this case we have $\kappa(\mathbf{x}, \mathbf{x}) = 1$, which means that all points $\phi(\mathbf{x})$ lie on a hypersphere around the feature space origin – which is however of infinite dimension, so geometric considerations don't help us much here. It is more helpful to think of a Gaussian kernel as imposing a Gaussian (i.e., multivariate normal, see Background 9.1 on p.267) surface on each support vector in instance space, so that the decision boundary is defined in terms of those Gaussian surfaces.

Kernel methods are best known in combination with support vector machines. Notice that the soft margin optimisation problem (Equation 7.12 on p.217) is defined in terms of dot products between training instances and hence the 'kernel trick' can be applied:

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1, \ldots, \alpha_n}{\arg\max} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n}\alpha_i$$
$$\text{subject to } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0$$

One thing to keep in mind is that the decision boundary learned with a non-linear kernel cannot be represented by a simple weight vector in input space. Thus, in order to classify a new example $\mathbf{x}$ we need to evaluate $y_i \sum_{j=1}^{n} \alpha_j y_j \kappa(\mathbf{x}, \mathbf{x}_j)$ which is an $O(n)$ computation involving all training examples, or at least the ones with non-zero multipliers α_j. This is why support vector machines are a popular choice as a kernel method, since they naturally promote sparsity in the support vectors. Although we have restricted attention to numerical features here, it is worth stressing that kernels can be defined over discrete structures, including trees, graphs, and logical formulae, and thus open the way to extending geometric models to non-numerical data.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

7.6 Linear models: Summary and further reading

After considering logical models in the previous three chapters we had a good look at linear models in this chapter. Logical models are inherently non-numerical, and so deal with numerical features by using thresholds to convert them into two or more intervals. Linear models are almost diametrically opposite in that they can deal with numerical features directly but need to pre-process non-numerical features.[3] Geometrically, linear models use lines and planes to build the model, which essentially means that a certain increase or decrease in one of the features has the same effect, regardless of that feature's value or any of the other features. They are simple and robust to variations in the training data, but sometimes suffer from underfitting as a consequence.

☞ In Section 7.1 we considered the least-squares method that was originally conceived to solve a regression problem. This classical method, which derives its name from minimising the sum of squared residuals between predicted and actual function values, is described in innumerable introductory mathematics and engineering texts (and was one of the example programs I remember running on my father's Texas Instruments TI-58 programmable calculator). We first had a look at the problem in univariate form, and then derived the general solution as $\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$, where $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ is a transformation that decorrelates, centres and normalises the features. We then discussed regularised versions of linear regression: ridge regression was introduced by Hoerl and Kennard (1970), and the lasso which naturally leads to sparse solutions was introduced by Tibshirani (1996). We saw how the least-squares method could be applied to binary classification by encoding the classes by +1 and −1, leading to the solution $\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}(\mathit{Pos}\,\boldsymbol{\mu}^{\oplus} - \mathit{Neg}\,\boldsymbol{\mu}^{\ominus})$. This generalises the basic linear classifier by taking feature correlation and unequal class prevalence into account, but at a considerably increased computational cost (quadratic in the number of instances and cubic in the number of features).

☞ Section 7.2 presented another classical linear model, the perceptron. Unlike the least-squares method, which always finds the optimal solution in terms of sum of squared residuals, the perceptron is a heuristic algorithm that depends, for one thing, on the order in which the examples are presented. Invented by Rosenblatt (1958), its convergence for linearly separable data was proved by Novikoff (1962), who gave an upper bound on the number of mistakes made before the perceptron converged. Minsky and Papert (1969) proved further formal properties of the perceptron, but also demonstrated the limitations of a linear classifier. These were overcome with the development, over an extended period of time and with contributions from many people, of the multilayer perceptron and its

[3] Ways to pre-process non-numerical features for use in linear models are discussed in Chapter 10.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

back-propagation training algorithm (Rumelhart, Hinton and Williams, 1986). In this section we also learned about the dual, instance-based view of linear classification in which we are learning instance weights rather than feature weights. For the perceptron these weights are the number of times the example has been misclassified during training.

☞ Maximum-margin classification with support vector machines was the topic of Section 7.3. The approach was proposed by Boser, Guyon and Vapnik (1992). Using the dual formulation, the instance weights are non-zero only for the support vectors, which are the training instances on the margin. The soft-margin generalisation is due to Cortes and Vapnik (1995). Margin errors are allowed, but the total margin error is added as a regularisation term to the objective function to be minimised, weighted by the complexity parameter C; all instances inside the margin receive instance weight C. As we have seen, by making C sufficiently small the support vector machine summarises the classes by their unweighted class means and hence is very similar to the basic linear classifier. A general introduction to SVMs is provided by Cristianini and Shawe-Taylor (2000). The sequential minimal optimisation algorithm is an often-used solver which iteratively selects pairs of multipliers to optimise analytically and is due to Platt (1998).

☞ In Section 7.4 we considered two methods to turn linear classifiers into probability estimators by converting the signed distance from the decision boundary into class probabilities. One well-known method is to use the logistic function, either straight out of the box or by fitting location and spread parameters to the data. Although this is often presented as a simple trick, we saw how it can be justified by assuming that the distances per class are normally distributed with the same variance; this latter assumption is needed to make the transformation monotonic. A non-parametric alternative is to use the ROC convex hull to obtain calibrated probability estimates. As was already mentioned in the summary of Chapter 2, the approach has its roots in isotonic regression (Best and Chakravarti, 1990) and was introduced to the machine learning community by Zadrozny and Elkan (2002). Fawcett and Niculescu-Mizil (2007) and Flach and Matsubara (2007) show its equivalence to calibration by means of the ROC convex hull.

☞ Finally, Section 7.5 discussed briefly how to go beyond linearity with kernel methods. The 'kernel trick' can be applied to any learning algorithm that can be entirely described in terms of dot products, which includes most approaches discussed in this chapter. The beauty is that we are implicitly classifying in a high-dimensional feature space, without having to construct the space explicitly. I

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press

gave the kernel perceptron as a simple example of a kernelised algorithm; in the next chapter we will see another example. Shawe-Taylor and Cristianini (2004) provide an excellent reference bringing together a wealth of material on the use of kernels in machine learning, and Gärtner (2009) discusses how kernel methods can be applied to structured, non-numerical data.

https://doi.org/10.1017/CBO9780511973000.009 Published online by Cambridge University Press