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.

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
```

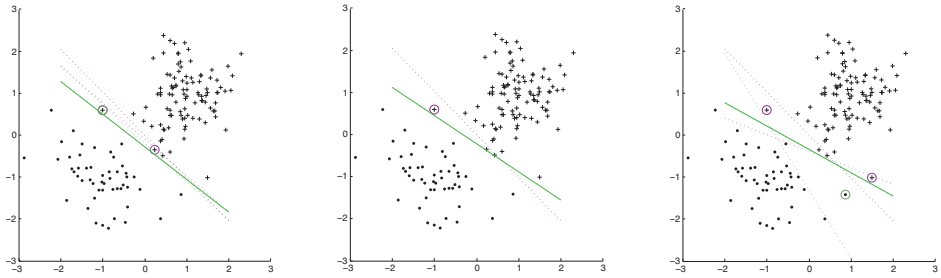

Figure 7.5. (left) A perceptron trained with a small learning rate ($\eta = 0.2$). The circled examples are the ones that trigger the weight update. **(middle)** Increasing the learning rate to $\eta = 0.5$ leads in this case to a rapid convergence. **(right)** Increasing the learning rate further to $\eta = 1$ 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 $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$.

```
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 x_i · x_j ≤ 0 then
7              α_i ← α_i + 1;
8              converged←false;
9          end
10     end
11 end
```

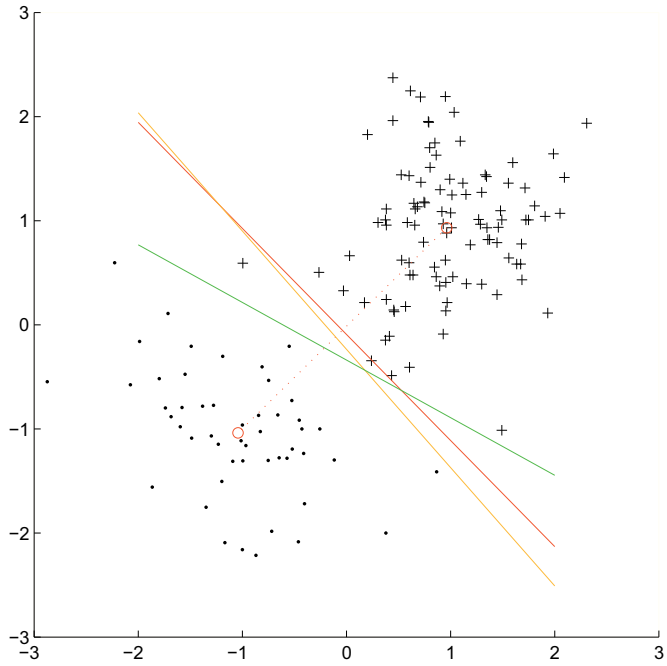

Figure 7.6. Three differently trained linear classifiers on a data set of 100 positives (top-right) and 50 negatives (bottom-left): the basic linear classifier in red, the least-squares classifier in orange and the perceptron in green. Notice that the perceptron perfectly separates the training data,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

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 $\hat{y} = \mathbf{w}\cdot\mathbf{x}$.

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