14.3. Boosting

Boosting is a powerful technique for combining multiple 'base' classifiers to produce a form of committee whose performance can be significantly better than that of any of the base classifiers. Here we describe the most widely used form of boosting algorithm called *AdaBoost*, short for 'adaptive boosting', developed by Freund and Schapire (1996). Boosting can give good results even if the base classifiers have a performance that is only slightly better than random, and hence sometimes the base classifiers are known as *weak learners*. Originally designed for solving classification problems, boosting can also be extended to regression (Friedman, 2001).

The principal difference between boosting and the committee methods such as bagging discussed above, is that the base classifiers are trained in sequence, and each base classifier is trained using a weighted form of the data set in which the weighting coefficient associated with each data point depends on the performance of the previous classifiers. In particular, points that are misclassified by one of the base classifiers are given greater weight when used to train the next classifier in the sequence. Once all the classifiers have been trained, their predictions are then combined through a weighted majority voting scheme, as illustrated schematically in Figure 14.1.

Consider a two-class classification problem, in which the training data comprises input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ along with corresponding binary target variables $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$. Each data point is given an associated weighting parameter w_n, which is initially set $1/N$ for all data points. We shall suppose that we have a procedure available for training a base classifier using weighted data to give a function $y(\mathbf{x}) \in \{-1, 1\}$. At each stage of the algorithm, AdaBoost trains a new classifier using a data set in which the weighting coefficients are adjusted according to the performance of the previously trained classifier so as to give greater weight to the misclassified data points. Finally, when the desired number of base classifiers have been trained, they are combined to form a committee using coefficients that give different weight to different base classifiers. The precise form of the AdaBoost algorithm is given below.

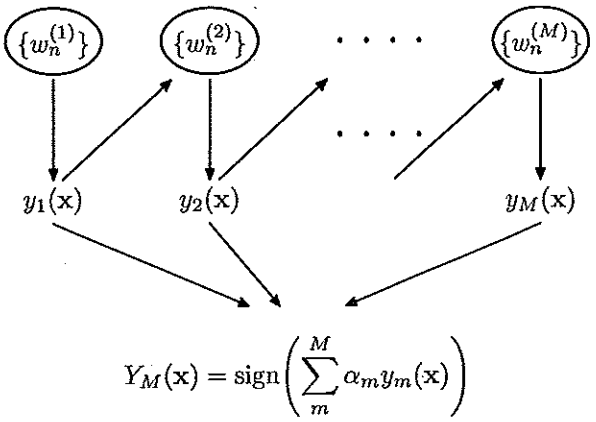

Figure 14.1 Schematic illustration of the boosting framework. Each base classifier $y_m(\mathbf{x})$ is trained on a weighted form of the training set (blue arrows) in which the weights $w_n^{(m)}$ depend on the performance of the previous base classifier $y_{m-1}(\mathbf{x})$ (green arrows). Once all base classifiers have been trained, they are combined to give the final classifier $Y_M(\mathbf{x})$ (red arrows).

AdaBoost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.15}$$

   where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

   (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}} \tag{14.16}$$

   and then use these to evaluate

$$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}. \tag{14.17}$$

   (c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\} \tag{14.18}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right). \tag{14.19}$$

We see that the first base classifier $y_1(\mathbf{x})$ is trained using weighting coefficients $w_n^{(1)}$ that are all equal, which therefore corresponds to the usual procedure for training a single classifier. From (14.18), we see that in subsequent iterations the weighting coefficients $w_n^{(m)}$ are increased for data points that are misclassified and decreased for data points that are correctly classified. Successive classifiers are therefore forced to place greater emphasis on points that have been misclassified by previous classifiers, and data points that continue to be misclassified by successive classifiers receive ever greater weight. The quantities ϵ_m represent weighted measures of the error rates of each of the base classifiers on the data set. We therefore see that the weighting coefficients α_m defined by (14.17) give greater weight to the more accurate classifiers when computing the overall output given by (14.19).

The AdaBoost algorithm is illustrated in Figure 14.2, using a subset of 30 data points taken from the toy classification data set shown in Figure A.7. Here each base learners consists of a threshold on one of the input variables. This simple classifier corresponds to a form of decision tree known as a 'decision stumps', i.e., a decision tree with a single node. Thus each base learner classifies an input according to whether one of the input features exceeds some threshold and therefore simply partitions the space into two regions separated by a linear decision surface that is parallel to one of the axes.

Section 14.4

14.3.1 Minimizing exponential error

Boosting was originally motivated using statistical learning theory, leading to upper bounds on the generalization error. However, these bounds turn out to be too loose to have practical value, and the actual performance of boosting is much better than the bounds alone would suggest. Friedman *et al.* (2000) gave a different and very simple interpretation of boosting in terms of the sequential minimization of an exponential error function.

Consider the exponential error function defined by

$$E = \sum_{n=1}^{N} \exp\{-t_n f_m(\mathbf{x}_n)\} \tag{14.20}$$

where $f_m(\mathbf{x})$ is a classifier defined in terms of a linear combination of base classifiers $y_l(\mathbf{x})$ of the form

$$f_m(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{m} \alpha_l y_l(\mathbf{x}) \tag{14.21}$$

and $t_n \in \{-1, 1\}$ are the training set target values. Our goal is to minimize E with respect to both the weighting coefficients α_l and the parameters of the base classifiers $y_l(\mathbf{x})$.

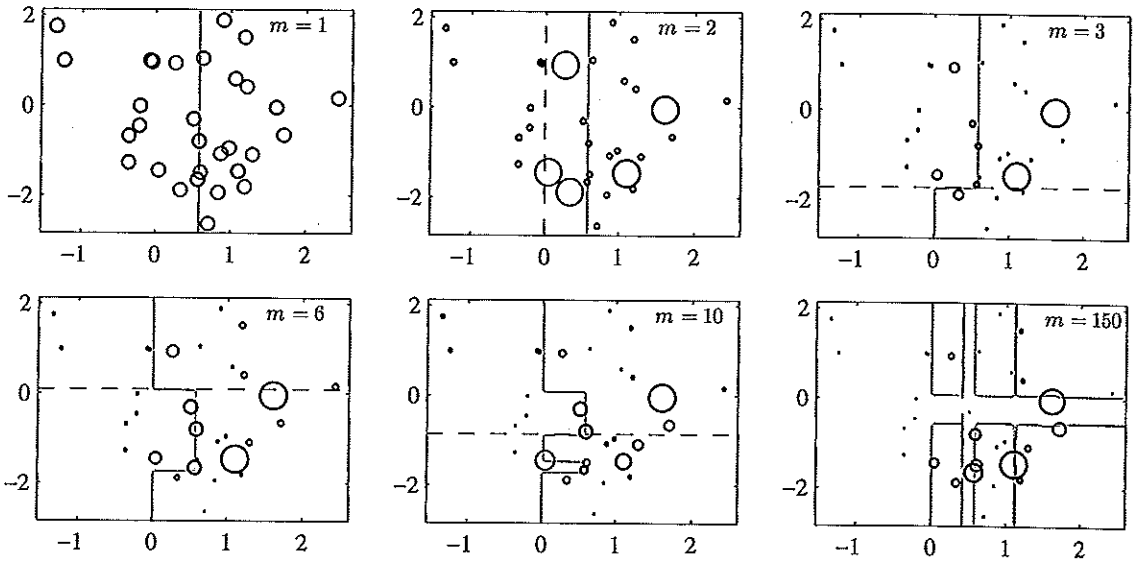

Figure 14.2 Illustration of boosting in which the base learners consist of simple thresholds applied to one or other of the axes. Each figure shows the number m of base learners trained so far, along with the decision boundary of the most recent base learner (dashed black line) and the combined decision boundary of the ensemble (solid green line). Each data point is depicted by a circle whose radius indicates the weight assigned to that data point when training the most recently added base learner. Thus, for instance, we see that points that are misclassified by the $m = 1$ base learner are given greater weight when training the $m = 2$ base learner.

Instead of doing a global error function minimization, however, we shall suppose that the base classifiers $y_1(\mathbf{x}), \ldots, y_{m-1}(\mathbf{x})$ are fixed, as are their coefficients $\alpha_1, \ldots, \alpha_{m-1}$, and so we are minimizing only with respect to α_m and $y_m(\mathbf{x})$. Separating off the contribution from base classifier $y_m(\mathbf{x})$, we can then write the error function in the form

$$\begin{aligned} E &= \sum_{n=1}^{N} \exp\left\{-t_n f_{m-1}(\mathbf{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n)\right\} \\ &= \sum_{n=1}^{N} w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n)\right\} \end{aligned} \tag{14.22}$$

where the coefficients $w_n^{(m)} = \exp\{-t_n f_{m-1}(\mathbf{x}_n)\}$ can be viewed as constants because we are optimizing only α_m and $y_m(\mathbf{x})$. If we denote by $\mathcal{T}_m$ the set of data points that are correctly classified by $y_m(\mathbf{x})$, and if we denote the remaining misclassified points by $\mathcal{M}_m$, then we can in turn rewrite the error function in the

form

$$E = e^{-\alpha_m/2} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{M}_m} w_n^{(m)}$$
$$= (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^{N} w_n^{(m)}. \tag{14.23}$$

When we minimize this with respect to $y_m(\mathbf{x})$, we see that the second term is constant, and so this is equivalent to minimizing (14.15) because the overall multiplicative factor in front of the summation does not affect the location of the minimum. Similarly, minimizing with respect to α_m, we obtain (14.17) in which ϵ_m is defined by (14.16).

Exercise 14.6

From (14.22) we see that, having found α_m and $y_m(\mathbf{x})$, the weights on the data points are updated using

$$w_n^{(m+1)} = w_n^{(m)} \exp\left\{-\frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n)\right\}. \tag{14.24}$$

Making use of the fact that

$$t_n y_m(\mathbf{x}_n) = 1 - 2I(y_m(\mathbf{x}_n) \neq t_n) \tag{14.25}$$

we see that the weights $w_n^{(m)}$ are updated at the next iteration using

$$w_n^{(m+1)} = w_n^{(m)} \exp(-\alpha_m/2) \exp\{\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)\}. \tag{14.26}$$

Because the term $\exp(-\alpha_m/2)$ is independent of n, we see that it weights all data points by the same factor and so can be discarded. Thus we obtain (14.18).

Finally, once all the base classifiers are trained, new data points are classified by evaluating the sign of the combined function defined according to (14.21). Because the factor of $1/2$ does not affect the sign it can be omitted, giving (14.19).

14.3.2 Error functions for boosting

The exponential error function that is minimized by the AdaBoost algorithm differs from those considered in previous chapters. To gain some insight into the nature of the exponential error function, we first consider the expected error given by

$$\mathbb{E}_{\mathbf{x},t}[\exp\{-ty(\mathbf{x})\}] = \sum_t \int \exp\{-ty(\mathbf{x})\} p(t|\mathbf{x}) p(\mathbf{x}) \, \mathrm{d}\mathbf{x}. \tag{14.27}$$

Exercise 14.7

If we perform a variational minimization with respect to all possible functions $y(\mathbf{x})$, we obtain

$$y(\mathbf{x}) = \frac{1}{2} \ln \left\{ \frac{p(t=1|\mathbf{x})}{p(t=-1|\mathbf{x})} \right\} \tag{14.28}$$

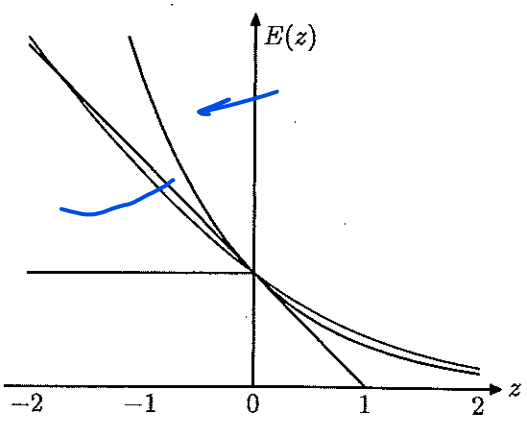

Figure 14.3 Plot of the exponential (green) and rescaled cross-entropy (red) error functions along with the hinge error (blue) used in support vector machines, and the misclassification error (black). Note that for large negative values of $z = ty(\mathbf{x})$, the cross-entropy gives a linearly increasing penalty, whereas the exponential loss gives an exponentially increasing penalty.

which is half the log-odds. Thus the AdaBoost algorithm is seeking the best approximation to the log odds ratio, within the space of functions represented by the linear combination of base classifiers, subject to the constrained minimization resulting from the sequential optimization strategy. This result motivates the use of the sign function in (14.19) to arrive at the final classification decision.

We have already seen that the minimizer $y(\mathbf{x})$ of the cross-entropy error (4.90) for two-class classification is given by the posterior class probability. In the case of a target variable $t \in \{-1, 1\}$, we have seen that the error function is given by $\ln(1 + \exp(-yt))$. This is compared with the exponential error function in Figure 14.3, where we have divided the cross-entropy error by a constant factor $\ln(2)$ so that it passes through the point $(0, 1)$ for ease of comparison. We see that both can be seen as continuous approximations to the ideal misclassification error function. An advantage of the exponential error is that its sequential minimization leads to the simple AdaBoost scheme. One drawback, however, is that it penalizes large negative values of $ty(\mathbf{x})$ much more strongly than cross-entropy. In particular, we see that for large negative values of ty, the cross-entropy grows linearly with $|ty|$, whereas the exponential error function grows exponentially with $|ty|$. Thus the exponential error function will be much less robust to outliers or misclassified data points. Another important difference between cross-entropy and the exponential error function is that the latter cannot be interpreted as the log likelihood function of any well-defined probabilistic model. Furthermore, the exponential error does not generalize to classification problems having $K > 2$ classes, again in contrast to the cross-entropy for a probabilistic model, which is easily generalized to give (4.108).

Section 7.1.2

Exercise 14.8

Section 4.3.4

The interpretation of boosting as the sequential optimization of an additive model under an exponential error (Friedman *et al.*, 2000) opens the door to a wide range of boosting-like algorithms, including multiclass extensions, by altering the choice of error function. It also motivates the extension to regression problems (Friedman, 2001). If we consider a sum-of-squares error function for regression, then sequential minimization of an additive model of the form (14.21) simply involves fitting each new base classifier to the residual errors $t_n - f_{m-1}(\mathbf{x}_n)$ from the previous model. As we have noted, however, the sum-of-squares error is not robust to outliers, and this

Exercise 14.9

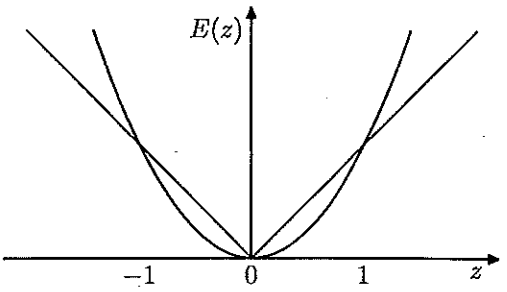

Figure 14.4 Comparison of the squared error (green) with the absolute error (red) showing how the latter places much less emphasis on large errors and hence is more robust to outliers and mislabelled data points.

can be addressed by basing the boosting algorithm on the absolute deviation $|y - t|$ instead. These two error functions are compared in Figure 14.4.