CHAPTER *11*

Model ensembles

TWO HEADS ARE BETTER THAN ONE – a well-known proverb suggesting that two minds working together can often achieve better results. If we read 'features' for 'heads' then this is certainly true in machine learning, as we have seen in the preceding chapters. But we can often further improve things by combining not just features but whole models, as will be demonstrated in this chapter. Combinations of models are generally known as *model ensembles*. They are among the most powerful techniques in machine learning, often outperforming other methods. This comes at the cost of increased algorithmic and model complexity.

The topic of model combination has a rich and diverse history, to which we can only partly do justice in this short chapter. The main motivations came from computational learning theory on the one hand, and statistics on the other. It is a well-known statistical intuition that averaging measurements can lead to a more stable and reliable estimate because we reduce the influence of random fluctuations in single measurements. So if we were to build an ensemble of slightly different models from the same training data, we might be able to similarly reduce the influence of random fluctuations in single models. The key question here is how to achieve diversity between these different models. As we shall see, this can often be achieved by training models on random subsets of the data, and even by constructing them from random subsets of the available features.

The motivation from computational learning theory went along the following lines. As we have seen in Section 4.4, learnability of hypothesis languages is studied in the context of a learning model, which determines what we mean by learnability. PAC-learnability requires that a hypothesis be approximately correct most of the time. An alternative learning model called *weak learnability* requires only that a hypothesis is learned that is slightly better than chance. While it appears obvious that PAC-learnability is stricter than weak learnability, it turns out that the two learning models are in fact equivalent: a hypothesis language is PAC-learnable if and only if it is weakly learnable. This was proved constructively by means of an iterative algorithm that repeatedly constructs a hypothesis aimed at correcting the mistakes of the previous hypothesis, thus 'boosting' it. The final model combined the hypotheses learned in each iteration, and therefore establishes an ensemble.

In essence, ensemble methods in machine learning have the following two things in common:

- ☞ they construct multiple, diverse predictive models from adapted versions of the training data (most often reweighted or resampled);
- ☞ they combine the predictions of these models in some way, often by simple averaging or voting (possibly weighted).

It should, however, also be stressed that these commonalities span a very large and diverse space, and that we should correspondingly expect some methods to be practically very different even though superficially similar. For example, it makes a big difference whether the way in which training data is adapted for the next iteration takes the predictions of the previous models into account or not. We will explore this space by means of the two best-known ensemble methods: bagging in Section 11.1 and boosting in Section 11.2. A short discussion of these and related ensemble methods then follows in Section 11.3, before we conclude the chapter in the usual way with a summary and pointers for further reading.

11.1 Bagging and random forests

Bagging, short for 'bootstrap aggregating', is a simple but highly effective ensemble method that creates diverse models on different random samples of the original data set. These samples are taken uniformly with replacement and are known as *bootstrap samples*. Because samples are taken with replacement the bootstrap sample will in general contain duplicates, and hence some of the original data points will be missing even if the bootstrap sample is of the same size as the original data set. This is exactly what we want, as differences between the bootstrap samples will create diversity among the models in the ensemble. To get an idea of how different bootstrap samples

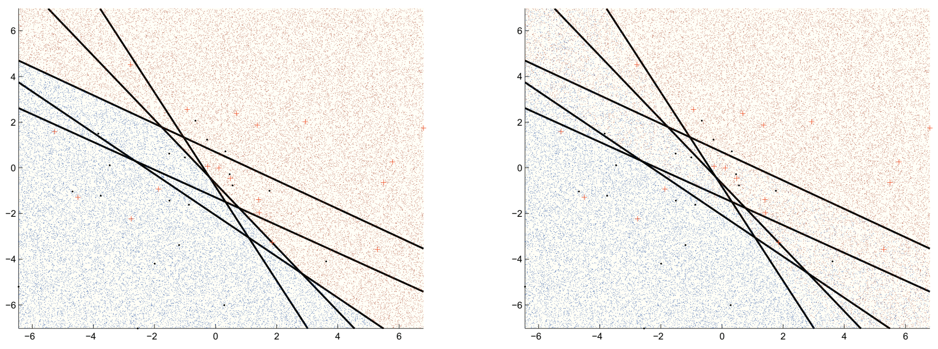

Figure 11.1. (left) An ensemble of five basic linear classifiers built from bootstrap samples with bagging. The decision rule is majority vote, leading to a piecewise linear decision boundary. **(right)** If we turn the votes into probabilities, we see the ensemble is effectively a grouping model: each instance space segment obtains a slightly different probability.

might be, we can calculate the probability that a particular data point is not selected for a bootstrap sample of size n as $(1-1/n)^n$, which for $n = 5$ is about one-third and has limit $1/e = 0.368$ for $n \to \infty$. This means that each bootstrap sample is likely to leave out about a third of the data points.

Algorithm 11.1 gives the basic bagging algorithm, which returns the ensemble as a set of models. We can choose to combine the predictions from the different models by voting – the class predicted by the majority of models wins – or by averaging, which is more appropriate if the base classifiers output scores or probabilities. An illustration is given in Figure 11.1. I trained five basic linear classifiers on bootstrap samples from 20 positive and 20 negative examples. We can clearly see the diversity of the five linear classifiers, which is helped by the fact that the data set is quite small. The

Algorithm 11.1: Bagging($D, T, \mathscr{A}$) – train an ensemble of models from bootstrap samples.

```
Input  : data set $D$; ensemble size $T$; learning algorithm $\mathscr{A}$.
Output : ensemble of models whose predictions are to be combined by voting
         or averaging.
1 for $t = 1$ to $T$ do
2     build a bootstrap sample $D_t$ from $D$ by sampling $|D|$ data points with
      replacement;
3     run $\mathscr{A}$ on $D_t$ to produce a model $M_t$;
4 end
5 return $\{M_t | 1 \le t \le T\}$
```

figure demonstrates the difference between combining predictions through voting (Figure 11.1 (left)) and creating a probabilistic classifier by averaging (Figure 11.1 (right)). With voting we see that bagging creates a piecewise linear decision boundary, something that is impossible with a single linear classifier. If we transform the votes from each model into probability estimates, we see that the different decision boundaries partition the instance space into segments that can potentially each receive a different score.

Bagging is particularly useful in combination with tree models, which are quite sensitive to variations in the training data. When applied to tree models, bagging is often combined with another idea: to build each tree from a different random subset of the features, a process also referred to as *subspace sampling*. This encourages the diversity in the ensemble even more, and has the additional advantage that the training time of each tree is reduced. The resulting ensemble method is called *random forests*, and the algorithm is given in Algorithm 11.2.

Since a decision tree is a grouping model whose leaves partition the instance space, so is a random forest: its corresponding instance space partition is essentially the intersection of the partitions of the individual trees in the ensemble. While the random forest partition is therefore finer than most tree partitions, it can in principle be mapped back to a single tree model (because intersection corresponds to combining the branches of two different trees). This is different from bagging linear classifiers, where the ensemble has a decision boundary that can't be learned by a single base classifier. One could say, therefore, that the random forest algorithm implements an alternative training algorithm for tree models.

Algorithm 11.2: RandomForest(D, T, d) – train an ensemble of tree models from bootstrap samples and random subspaces.

```
   Input  : data set $D$; ensemble size $T$; subspace dimension $d$.
   Output : ensemble of tree models whose predictions are to be combined by
            voting or averaging.
1  for $t = 1$ to $T$ do
2      build a bootstrap sample $D_t$ from $D$ by sampling $|D|$ data points with
       replacement;
3      select $d$ features at random and reduce dimensionality of $D_t$ accordingly;
4      train a tree model $M_t$ on $D_t$ without pruning;
5  end
6  return $\{M_t | 1 \leq t \leq T\}$
```

11.2 Boosting

Boosting is an ensemble technique that is superficially similar to bagging, but uses a more sophisticated technique than bootstrap sampling to create diverse training sets. The basic idea is simple and appealing. Suppose we train a linear classifier on a data set and find that its training error rate is ϵ. We want to add another classifier to the ensemble that does better on the misclassifications of the first classifier. One way to do that is to duplicate the misclassified instances: if our base model is the basic linear classifier, this will shift the class means towards the duplicated instances. A better way to achieve the same thing is to give the misclassified instances a higher weight, and to modify the classifier to take these weights into account (e.g., the basic linear classifier can calculate the class means as a weighted average).

But how much should the weights change? The idea is that half of the total weight is assigned to the misclassified examples, and the other half to the rest. Since we started with uniform weights that sum to 1, the current weight assigned to the misclassified examples is exactly the error rate ϵ, so we multiply their weights by $1/2\epsilon$. Assuming $\epsilon < 0.5$ this is an increase in weight as desired. The weights of the correctly classified examples get multiplied by $1/2(1-\epsilon)$, so the adjusted weights again sum to 1. In the next round we do exactly the same, except we take the non-uniform weights into account when evaluating the error rate.

Example 11.1 (Weight updates in boosting). Suppose a linear classifier achieves performance as in the contingency table on the left. The error rate is $\epsilon = (9+16)/100 = 0.25$. The weight update for the misclassified examples is a factor $1/2\epsilon = 2$ and for the correctly classified examples $1/2(1-\epsilon) = 2/3$.

	Predicted ⊕	*Predicted* ⊖	
Actual ⊕	**24**	**16**	40
Actual ⊖	**9**	**51**	60
	33	67	100

	⊕	⊖	
⊕	**16**	**32**	48
⊖	**18**	**34**	52
	34	66	100

Taking these updated weights into account leads to the contingency table on the right, which has a (weighted) error rate of 0.5.

We need one more ingredient in our boosting algorithm and that is a confidence factor α for each model in the ensemble, which we will use to form an ensemble prediction that is a weighted average of each individual model. Clearly we want α to increase with decreasing ϵ: a common choice is

$$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t} = \ln\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \tag{11.1}$$

which we will justify in a moment. The basic boosting algorithm is given in Algorithm 11.3. Figure 11.2 (left) illustrates how a boosted ensemble of five basic linear classifiers can achieve zero training error. It is clear that the resulting decision boundary is much more complex than could be achieved by a single basic linear classifier. In contrast, a bagged ensemble of basic linear classifiers has learned five very similar decision boundaries, the reason being that on this data set the bootstrap samples are all very similar.

I will now justify the particular choice for α_t in Equation 11.1. First, I will show that the two weight updates for the misclassified instances and the correctly classified instances can be written as reciprocal terms δ_t and $1/\delta_t$ normalised by some term Z_t:

$$\frac{1}{2\epsilon_t} = \frac{\delta_t}{Z_t} \qquad\qquad \frac{1}{2(1-\epsilon_t)} = \frac{1/\delta_t}{Z_t}$$

The second expression gives $\delta_t = 2(1-\epsilon_t)/Z_t$; substituting this back into the first expression yields

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)} \qquad\qquad \delta_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \exp(\alpha_t) \tag{11.2}$$

Algorithm 11.3: Boosting($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

```
   Input   : data set $D$; ensemble size $T$; learning algorithm $\mathscr{A}$.
   Output  : weighted ensemble of models.
1  $w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$ ;                          // start with uniform weights
2  for $t = 1$ to $T$ do
3      run $\mathscr{A}$ on $D$ with weights $w_{ti}$ to produce a model $M_t$;
4      calculate weighted error $\epsilon_t$;
5      if $\epsilon_t \geq 1/2$ then
6          set $T \leftarrow t-1$ and break
7      end
8      $\alpha_t \leftarrow \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ ;                          // confidence for this model
9      $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$ ;       // increase weight
10     $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$ ;  // decrease weight
11 end
12 return $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$
```

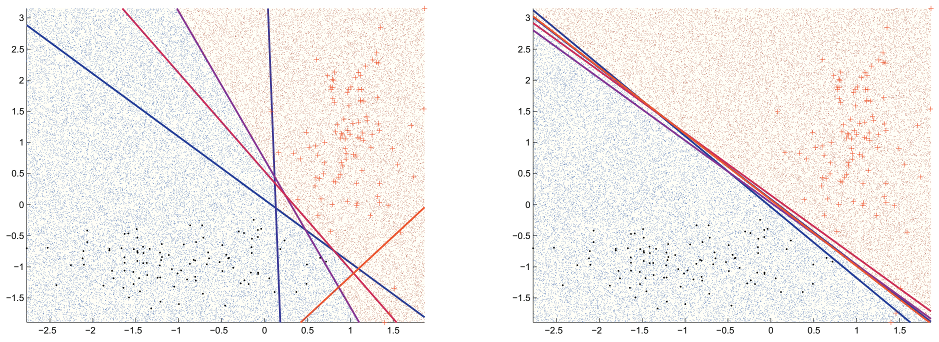

Figure 11.2. (left) An ensemble of five boosted basic linear classifiers with majority vote. The linear classifiers were learned from blue to red; none of them achieves zero training error, but the ensemble does. **(right)** Applying bagging results in a much more homogeneous ensemble, indicating that there is little diversity in the bootstrap samples.

So the weight update for misclassified instances is $\exp(\alpha_t)/Z_t$ and for correctly classified instances $\exp(-\alpha_t)/Z_t$. Using the fact that $y_i M_t(x_i) = +1$ for instances correctly classified by model M_t and -1 otherwise, we can write the weight update as

$$w_{(t+1)i} = w_{ti}\frac{\exp\left(-\alpha_t y_i M_t(x_i)\right)}{Z_t}$$

which is the expression commonly found in the literature.

Let us now step back and pretend that we haven't yet decided what α_t should be in each round. Since the weight updates are multiplicative, we have

$$w_{(T+1)i} = w_{1i}\prod_{t=1}^{T}\frac{\exp\left(-\alpha_t y_i M_t(x_i)\right)}{Z_t} = \frac{1}{|D|}\frac{\exp\left(-y_i M(x_i)\right)}{\prod_{t=1}^{T} Z_t}$$

where $M(x_i) = \sum_{t=1}^{T}\alpha_t M_t(x_i)$ is the model represented by the boosted ensemble. The weights always add up to 1 over the instance space, and so

$$\prod_{t=1}^{T} Z_t = \frac{1}{|D|}\sum_{i=1}^{|D|}\exp\left(-y_i M(x_i)\right)$$

Notice that $\exp\left(-y_i M(x_i)\right)$ is always positive and at least 1 if $-y_i M(x_i)$ is positive, which happens if x_i is misclassified by the ensemble (i.e., $\text{sign}(M(x_i) \neq y_i)$. So the right-hand side of this expression is at least equal to the training error of the boosted ensemble, and $\prod_{t=1}^{T} Z_t$ is an upper bound on that training error. A simple heuristic would therefore be to greedily minimise

$$Z_t = \sum_{i=1}^{|D|} w_{ti}\exp\left(-\alpha_t y_i M_t(x_i)\right) \tag{11.3}$$

in each boosting round. Now, the sum of the weights of instances incorrectly classified by M_t is ϵ_t, and so

$$Z_t = \epsilon_t \exp(\alpha_t) + (1-\epsilon_t)\exp(-\alpha_t)$$

Taking the derivative with respect to α_t, setting it to zero and solving for α_t yields α_t as given in Equation 11.1 and Z_t as given in Equation 11.2.

Notice that Equation 11.3 demonstrates that the loss function minimised by boosting is ☞*exponential loss* $\exp\big(-y\hat{s}(x)\big)$ which we already encountered in Figure 2.6 on p.63. Notice, furthermore, that minimising Z_t means minimising $2\sqrt{\epsilon_t(1-\epsilon_t)}$ according to Equation 11.2. You may recognise this as the ☞$\sqrt{\text{Gini}}$ *impurity measure* we investigated in Chapter 5. There, we saw that this splitting criterion is insensitive to changes in the class distribution (see Figure 5.7 on p.146). Here, it arises essentially because of the way weight updates are implemented in the boosting algorithm.

Boosted rule learning

An interesting variant of boosting arises when our base models are partial classifiers that sometimes abstain from making a prediction. For example, suppose that our base classifiers are conjunctive rules whose head is fixed to predicting the positive class. An individual rule therefore either predicts the positive class for those instances it covers, or otherwise abstains from making a prediction. We can use boosting to learn an ensemble of such rules that takes a weighted vote among its members.

We need to make some small adjustments to the boosting equations, as follows. Notice that ϵ_t is the weighted error of the t-th base classifier. Since our rules always predict positive for covered instances, these errors only concern covered negatives, which we will indicate by $\epsilon_t^{\ominus}$. Similarly, we indicate the weighted sum of covered positives as $\epsilon_t^{\oplus}$, which will play the role of $1-\epsilon_t$. However, with abstaining rules there is a third component, indicated as ϵ_t^0, which is the weighted sum of instances which the rule doesn't cover ($\epsilon_t^0 + \epsilon_t^{\oplus} + \epsilon_t^{\ominus} = 1$). We then have

$$Z_t = \epsilon_t^0 + \epsilon_t^{\ominus}\exp(\alpha_t) + \epsilon_t^{\oplus}\exp(-\alpha_t)$$

The value of α_t which maximises this is

$$\alpha_t = \frac{1}{2}\ln\frac{\epsilon_t^{\oplus}}{\epsilon_t^{\ominus}} = \ln\sqrt{\frac{\epsilon_t^{\oplus}}{\epsilon_t^{\ominus}}} \tag{11.4}$$

which gives

$$Z_t = \epsilon_t^0 + 2\sqrt{\epsilon_t^{\oplus}\epsilon_t^{\ominus}} = 1 - \epsilon_t^{\oplus} - \epsilon_t^{\ominus} + 2\sqrt{\epsilon_t^{\oplus}\epsilon_t^{\ominus}} = 1 - \left(\sqrt{\epsilon_t^{\oplus}} - \sqrt{\epsilon_t^{\ominus}}\right)^2$$

This means that in each boosting round we construct a rule that maximises $\left|\sqrt{\epsilon_t^{\oplus}} - \sqrt{\epsilon_t^{\ominus}}\right|$, and set its confidence factor to α_t as in Equation 11.4. In order to obtain a prediction from the ensemble, we add up the confidence factors of all rules covering it. Note that these confidence factors are negative if $\epsilon_t^{\oplus} < \epsilon_t^{\ominus}$, which indicates that the rule correlates with the negative class; this is not a problem as such, but can be avoided by changing the objective function for individual rules to $\sqrt{\epsilon_t^{\oplus}} - \sqrt{\epsilon_t^{\ominus}}$.

The weight updates after each iteration of boosting are the same as previously, except that the weights of examples not covered by the rule do not change. Boosted rule learning is therefore similar to the ☞*weighted covering* (Algorithm 6.5 on p.182) algorithm for subgroup discovery. The difference is that there we wanted to promote rule overlap without reference to the class and hence decrease the weight of all covered examples, whereas here we decrease the weight of covered positives and increase the weight of covered negatives.

11.3 Mapping the ensemble landscape

Now that we have looked at two often-used ensemble methods in somewhat more detail, we consider how their differences in performance might be explained, before turning attention to some of the many other ensemble methods in the literature.

Bias, variance and margins

Ensemble methods are a good vehicle to further understand the ☞*bias–variance dilemma* we discussed in the context of regression in Section 3.2. Broadly speaking, there are three reasons why a model may misclassify a test instance. First, it may simply be unavoidable in the given feature space if instances from different classes are described by the same feature vectors. In a probabilistic context this happens when the per-class distributions $P(X|Y)$ overlap, so that the same instance has non-zero likelihoods for several classes. In such a situation, the best we can hope to do is to approximate the target concept.

The second reason for classification errors is that the model lacks expressivity to exactly represent the target concept. For example, if the data is not linearly separable then even the best linear classifier will make mistakes. This is the bias of a classifier, and it is inversely related to its expressivity. Although there is no generally agreed way to measure expressivity or bias of a classifier[1] it is intuitively clear that, say, a hyperbolic decision boundary has lower bias than a linear one. It is also clear that tree models have the lowest possible bias, as their leaves can be made arbitrarily small to cover singleton instances.

It may seem that low-bias models are generally preferable. However, a practical rule of thumb in machine learning is that *low-bias models tend to have high variance, and*

[1]While squared loss nicely decomposes into squared bias and variance as shown in Equation 3.2 on p.93, loss functions used in classification such as 0–1 loss can be decomposed in several ways.

vice versa. Variance is the third source of classification errors. A model has high variance if its decision boundary is highly dependent on the training data. For example, the nearest-neighbour classifier's instance space segments are determined by a single training point, so if I move a training point in a segment bordering on the decision boundary, that boundary will change. Tree models have high variance for a different reason: if I change the training data sufficiently for another feature to be selected at the root of the tree, then the rest of the tree is likely to be different as well. An example of a low-variance model is the basic linear classifier, because it averages over all the points in a class.

Now look back at Figure 11.1 on p.332. The bagged ensemble of basic linear classifiers has learned a piecewise linear decision boundary that exceeds the expressivity of a single linear classifier. This illustrates that bagging, like any ensemble method, is capable of reducing the bias of a high-bias base model such as a linear classifier. However, if we compare this with boosting in Figure 11.2 on p.336, we see that the reduction in bias resulting from bagging is much smaller than that of boosting. In fact, *bagging is predominantly a variance-reduction technique, while boosting is primarily a bias-reduction technique*. This explains why bagging is often used in combination with high-variance models such as tree models (☞*random forests* in Algorithm 11.2), whereas boosting is typically used with high-bias models such as linear classifiers or univariate decision trees (also called *decision stumps*).

Another way to understand boosting is in terms of margins. Intuitively, the margin is the signed distance from the decision boundary, with the sign indicating whether we are on the correct or the wrong side. It has been observed in experiments that boosting is effective in increasing the margins of examples, even if they are already on the correct side of the decision boundary. The effect is that boosting may continue to improve performance on the test set even after the training error has been reduced to zero. Given that boosting was originally conceived in a PAC-learning framework, which is not specifically aimed at increasing margins, this was a surprising result.

Other ensemble methods

There are many other ensemble methods beyond bagging and boosting. The main variation lies in the way predictions of the base models are combined. Notice that this could itself be defined as a learning problem: given the predictions of some base classifiers as features, learn a *meta-model* that best combines their predictions. For example, in boosting we could learn the weights α_t rather than deriving them from each base model's error rate. Learning a linear meta-model is known as *stacking*. Several variations on this theme exist: e.g., decision trees have been used as the meta-model.

It is also possible to combine different base models into a heterogeneous ensemble: in this way the base model diversity derives from the fact that base models are trained by different learning algorithms, and so they can all use the same training set. Some of these base models might employ different settings of a parameter: for example, the ensemble might include several support vector machines with different values of the complexity parameter which regulates the extent to which margin errors are tolerated.

Generally speaking, then, model ensembles consist of a set of base models and a meta-model that is trained to decide how base model predictions should be combined. Implicitly, training a meta-model involves an assessment of the quality of each base model: for instanceif the meta-model is linear as in stacking, a weight close to zero means that the corresponding base classifier does not contribute much to the ensemble. It is even conceivable that a base classifier obtains a negative weight, meaning that in the context of the other base models its predictions are best inverted. We could go one step further and try to *predict* how well a base model is expected to perform, even before we train it! By formulating this as a learning problem at the meta-level, we arrive at the field of meta-learning.

Meta-learning

Meta-learning first involves training a variety of models on a large collection of data sets. The aim is then to construct a model that can help us answer questions such as the following:

- ☞ In which situations is a decision tree likely to outperform a support vector machine?
- ☞ When can a linear classifier be expected to perform poorly?
- ☞ Can the data be used to give suggestions for setting particular parameters?

The key question in meta-learning is how to design the features on which the meta-model is built. These features should combine data set characteristics and relevant aspects of the trained model. Data set characteristics should go much further than simply listing the number and kind of features and the number of instances, as it is unlikely that anything can be predicted about a model's performance from just that information. For example, we can try to assess the noise level of a data set by measuring the size of a trained decision tree before and after pruning. Training simple models such as decision stumps on a data set and measuring their performance also gives useful information.

In Background 1.1 on p.20 we referred to the no free lunch theorem, which states that no learning algorithm can outperform any other learning algorithm over the set of all possible learning problems. As a corollary, we have that meta-learning over all possible learning problems is futile: if it wasn't, we could build a single hybrid model that uses a meta-model to tell us which base model would achieve better than random performance on a particular data set. It follows that we can only hope to achieve useful meta-learning over non-uniform distributions of learning problems.

11.4 Model ensembles: Summary and further reading

In this short chapter we have discussed some of the fundamental ideas underlying ensemble methods. What all ensemble methods have in common is that they construct several base models from adapted versions of the training data, on top of which some technique is employed to combine the predictions or scores from the base models into a single prediction of the ensemble. We focused on bagging and boosting as two of the most commonly used ensemble methods. A good introduction to model ensembles is given by Brown (2010). The standard reference on classifier combination is Kuncheva (2004) and a more recent overview is given by Zhou (2012).

☞ In Section 11.1 we discussed bagging and random forests. Bagging trains diverse models from samples of the training data, and was introduced by Breiman (1996*a*). Random forests, usually attributed to Breiman (2001), combine bagged decision trees with random subspaces; similar ideas were developed by Ho (1995) and Amit and Geman (1997). These techniques are particularly useful to reduce the variance of low-bias models such as tree models.

☞ Boosting was discussed in Section 11.2. The key idea is to train diverse models by increasing the weight of previously misclassified examples. This helps to reduce the bias of otherwise stable learners such as linear classifiers or decision stumps. An accessible overview is given by Schapire (2003). Kearns and Valiant (1989, 1994) posed the question whether a weak learning algorithm that performs just slightly better than random guessing can be boosted into an arbitrarily accurate strong learning algorithm. Schapire (1990) introduced a theoretical form of boosting to show the equivalence of weak and strong learnability. The AdaBoost algorithm on which Algorithm 11.3 is based was introduced by Freund and Schapire (1997). Schapire and Singer (1999) give multi-class and multi-label extensions of AdaBoost. A ranking version of AdaBoost was proposed by Freund *et al.* (2003). The boosted rule learning approach that can handle classifiers that may abstain was inspired by Slipper (Cohen and Singer, 1999), a boosted version of Ripper (Cohen, 1995).

☞ In Section 11.3 we discussed bagging and boosting in terms of bias and variance. Schapire, Freund, Bartlett and Lee (1998) provide a detailed theoretical and experimental analysis of boosting in terms of improving the margin distribution. I also mentioned some other ensemble methods that train a meta-model for combining the base models. Stacking employs a linear meta-model and was introduced by Wolpert (1992) for classification and extended by Breiman (1996*b*) for regression. Meta-decision trees were introduced by Todorovski and Dzeroski (2003).

☞ We also briefly discussed meta-learning as a technique for learning about the performance of learning algorithms. The field originated from an early empirical study documented by Michie *et al.* (1994). Recent references are Brazdil *et al.* (2009, 2010). Unpruned and unpruned decision trees were used to obtain data set characteristics by Peng *et al.* (2002). The idea of training simple models to obtain further data characteristics is known as landmarking (Pfahringer *et al.*, 2000).