Ensemble Learning

Table of Contents

Z.-H. Zhou, *Machine Learning*,
https://doi.org/10.1007/978-981-15-1967-3_8

8.1 Individual and Ensemble

Ensemble learning, also known as multiple classifier system and committee-based learning, trains and combines multiple learners to solve a learning problem.

As shown in ◘ Figure 8.1, the typical workflow of ensemble learning is training a set of individual learners first and then combining them via some strategies, where an individual learner is usually trained by an existing learning algorithm, such as the C4.5 algorithm and the BP neural network algorithm. An ensemble is said to be homogeneous if all individual learners are of the same type, e.g., a "decision tree ensemble" contains only decision trees, while a "neural network ensemble" contains only neural networks. For homogeneous ensembles, the individual learners are called *base learners*, and the corresponding learning algorithms are called *base learning algorithms*. In contrast, a heterogeneous ensemble contains different individual learners and learning algorithms, and there is no single base learner or base learning algorithm. For heterogeneous ensembles, the individual learners are unusually called *component learners* or simply individual learners.

8

By combining multiple learners, the generalization ability of an ensemble is often much stronger than that of an individual learner, and this is especially true for *weak learners*. Therefore, theoretical studies on ensemble learning often focus on weak learners, and hence base learners are sometimes called weak learners. In practice, however, despite that an ensemble of weak learners can theoretically obtain good performance, people still prefer strong learners for some reasons, such as reducing the number of individual learners and reusing existing knowledge about the strong learners.

Weak learners typically refer to learners with generalization ability just slightly better than random guessing, e.g., with an accuracy slightly above 50% in binary classification problems.

Intuitively, mixing things with different qualities will produce something better than the worst one but worse than the best one. Then, how can an ensemble produce better performance than the best individual learner?

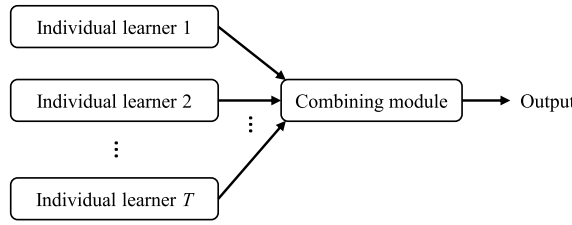


Fig. 8.1 The workflow of ensemble learning

	Testing sample 1	Testing sample 2	Testing sample 3
h_1	✓	✓	✗
h_2	✗	✓	✓
h_3	✓	✗	✓
Ensemble	✓	✓	✓

(a) Ensemble helps.

	Testing sample 1	Testing sample 2	Testing sample 3
h_1	✓	✓	✗
h_2	✓	✓	✗
h_3	✓	✓	✗
Ensemble	✓	✓	✗

(b) Ensemble doesn't help.

	Testing sample 1	Testing sample 2	Testing sample 3
h_1	✓	✗	✗
h_2	✗	✓	✗
h_3	✗	✗	✓
Ensemble	✗	✗	✗

(c) Ensemble hurts.

Fig. 8.2 Individual learners should be "accurate and diverse" (h_i is the ith learner)

Taking binary classification as an example, suppose three classifiers are applied to three testing samples, as shown in ◘ Figure 8.2, where the ticks indicate the correct classifications, and the crosses indicate the incorrect classifications. The classification of ensemble learning is made by voting. In ◘ Figure 8.2a, each classifier achieves an accuracy of 66.6%, while the ensemble achieves an accuracy of 100%. In ◘ Figure 8.2b, the three classifiers have made identical decisions, and the ensemble does not improve the result. In ◘ Figure 8.2c, each classifier achieves an accuracy of 33.3%, while the ensemble gives an even worse result. From this example, we see that a good ensemble should contain individual learners that are "accurate and diverse". In other words, individual learners must be not bad and have *diversity* (i.e., the learners are different).

An individual learner should be no worse than a weak learner.

Let us do a simple analysis with binary classification, that is, $y \in \{-1, +1\}$. Suppose the ground-truth function is f, and the error rate of each base learner is ϵ. Then, for each base learner h_i, we have

$$P(h_i(\boldsymbol{x}) \neq f(\boldsymbol{x})) = \epsilon. \tag{8.1}$$

For ease of discussion, we assume T is odd.

Suppose ensemble learning combines the T base learners by voting, then the ensemble will make an correct classification if more than half of the base learners are correct:

$$F(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{T} h_i(\boldsymbol{x})\right). \tag{8.2}$$

Assuming the error rates of base learners are independent, then, from Hoeffding's inequality, the error rate of the ensemble is

See Exercise 8.1.

$$\begin{aligned} P(F(\boldsymbol{x}) \neq f(\boldsymbol{x})) &= \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \\ &\leqslant \exp\left(-\frac{1}{2} T(1-2\epsilon)^2\right). \end{aligned} \tag{8.3}$$

The above equation shows that as the number of base learners T in the ensemble increases, the error rate decreases exponentially and eventually approaches zero.

The above analysis makes a critical assumption that the error rates of base learners are independent. However, this assumption is invalid in practice since the learners are trained to solve the same problem and thus cannot be independent. In fact, accuracy and diversity are two conflicting properties of individual learners. Generally, when the accuracy is already high, we usually need to sacrifice some accuracy if we wish to increase diversity. It turns out that the generation and combination of "accurate and diverse" individual learners are the fundamental issues in ensemble learning.

Current ensemble learning methods can be roughly grouped into two categories, depending on how the individual learners are generated. The first category, represented by *Boosting*, creates individual learners with strong correlations and generates the learners sequentially. The second category, represented by *Bagging* and *Random Forest*, creates individual learners independently and can parallelize the generation process.

8.2 Boosting

Boosting is a family of algorithms that convert weak learners to strong learners. Boosting algorithms start with training a base learner and then adjust the distribution of the training samples according to the result of the base learner such that incorrectly classified samples will receive more attention by subsequent base learners. After training the first base learner, the second base learner is trained with the adjusted training samples, and the result is used to adjust the training sample distribution again. Such a process repeats until the number of base learners reaches a predefined value T, and finally, these base learners are weighted and combined.

The most well-known Boosting algorithm is AdaBoost (Freund and Schapire 1997), as shown in ◘ Algorithm 8.1, where $y_i \in \{-1, +1\}$ and f is the ground-truth function.

There are multiple ways to derive the AdaBoost algorithm, but one that is easy to understand is based on the *additive model*, that is, using the linear combination of base learners

$$H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \tag{8.4}$$

to minimize the *exponential loss function* (Friedman et al. 2000)

Algorithm 8.1 AdaBoost

Input: Training set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
Base learner $\mathcal{L}$;
Number of training rounds T.

Process:

1: $\mathcal{D}_1(\boldsymbol{x}) = 1/m$; — Initialize the sample weight distribution.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $h_t = \mathcal{L}(D, \mathcal{D}_t)$; — Train classifier h_t using data set D that follows distribution $\mathcal{D}_t$.
4: $\epsilon_t = P_{\boldsymbol{x}\sim\mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$; — Estimate the error of h_t.
5: **if** $\epsilon_t > 0.5$ **then break**
6: $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; — Determine the weight of classifier h_t.
7: $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(\boldsymbol{x}) = f(\boldsymbol{x}); \\ \exp(\alpha_t), & \text{if } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}); \end{cases}$ — Update the sample distribution, where Z_t is the normalization factor that ensures $\mathcal{D}_{t+1}$ is a valid distribution.
$= \frac{\mathcal{D}_t(\boldsymbol{x})\exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t}$;
8: **end for**

Output: $F(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$.

$$\ell_{\exp}(H \mid \mathcal{D}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H(\boldsymbol{x})}\right]. \tag{8.5}$$

If $H(\boldsymbol{x})$ minimizes the exponential loss, then we consider the partial derivative of (8.5) with respect to $H(\boldsymbol{x})$

$$\frac{\partial \ell_{\exp}(H \mid \mathcal{D})}{\partial H(\boldsymbol{x})} = -e^{-H(\boldsymbol{x})} P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) + e^{H(\boldsymbol{x})} P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}), \tag{8.6}$$

and setting it to zero gives

$$H(\boldsymbol{x}) = \frac{1}{2}\ln\frac{P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x})}{P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x})}. \tag{8.7}$$

Hence, we have

$$\begin{aligned} \operatorname{sign}(H(\boldsymbol{x})) &= \operatorname{sign}\left(\frac{1}{2}\ln\frac{P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x})}{P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x})}\right) \\ &= \begin{cases} 1, & P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) > P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}) \\ -1, & P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) < P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x}) \end{cases} \\ &= \underset{y\in\{-1,1\}}{\arg\max}\, P(f(\boldsymbol{x}) = y \mid \boldsymbol{x}), \end{aligned} \tag{8.8}$$

Here, we ignore the case of $P(f(\boldsymbol{x}) = 1 \mid \boldsymbol{x}) = P(f(\boldsymbol{x}) = -1 \mid \boldsymbol{x})$.

which implies that $\text{sign}(H(\boldsymbol{x}))$ achieves the Bayes optimal error rate. In other words, the classification error rate is minimized when the exponential loss is minimized, and hence the exponential loss function is a consistent surrogate function of the original 0/1 loss function. Since this surrogate function has better mathematical properties, e.g., continuously differentiable, it is used as the optimization objective replacing the 0/1 loss function.

See Sect. 6.7 for the "consistency" of surrogate functions.

In the AdaBoost algorithm, the base learning algorithm generates the first base classifier h_1 from the original training data and then iteratively generates the subsequent base classifiers h_t and associated weights α_t. Once the classifier h_t is generated from the distribution $\mathcal{D}_t$, its weight α_t is estimated by letting $\alpha_t h_t$ minimize the exponential loss function

$$\begin{aligned}\ell_{\exp}(\alpha_t h_t \mid \mathcal{D}_t) &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[e^{-\alpha_t}\mathbb{I}(f(\boldsymbol{x}) = h_t(\boldsymbol{x})) + e^{\alpha_t}\mathbb{I}(f(\boldsymbol{x}) \neq h_t(\boldsymbol{x}))\right] \\ &= e^{-\alpha_t}P_{\boldsymbol{x}\sim\mathcal{D}_t}(f(\boldsymbol{x}) = h_t(\boldsymbol{x})) + e^{\alpha_t}P_{\boldsymbol{x}\sim\mathcal{D}_t}(f(\boldsymbol{x}) \neq h_t(\boldsymbol{x})) \\ &= e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t, \end{aligned} \tag{8.9}$$

where $\epsilon = P_{\boldsymbol{x}\sim\mathcal{D}_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$. Setting the derivative of the exponential loss function

$$\frac{\partial \ell_{\exp}(\alpha_t h_t \mid \mathcal{D}_t)}{\partial \alpha_t} = -e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t \tag{8.10}$$

to zero gives the optimal α_t as

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right), \tag{8.11}$$

which is exactly the equation shown in line 6 of ◘ Algorithm 8.1.

The AdaBoost algorithm adjusts the sample distribution based on H_{t-1} such that the base learner h_t in the next round can correct some mistakes made by H_{t-1}. Ideally, we wish h_t to correct all mistakes made by H_{t-1}. The minimization of $\ell_{\exp}(H_{t-1} + \alpha_t h_t \mid \mathcal{D})$ can be simplified to

$$\begin{aligned}\ell_{\exp}(H_{t-1} + h_t \mid \mathcal{D}) &= \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})(H_{t-1}(\boldsymbol{x})+h_t(\boldsymbol{x}))}\right] \\ &= \mathbb{E}_{x\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})h_t(\boldsymbol{x})}\right]. \end{aligned} \tag{8.12}$$

Since $f^2(\boldsymbol{x}) = h_t^2(\boldsymbol{x}) = 1$, (8.12) can be approximated using Taylor expansion of $e^{-f(\boldsymbol{x})h_t(\boldsymbol{x})}$ as

$$\ell_{\exp}(H_{t-1}+h_t \mid \mathcal{D}) \simeq \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h_t(\boldsymbol{x})+\frac{f^2(\boldsymbol{x})h_t^2(\boldsymbol{x})}{2}\right)\right]$$
$$= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h_t(\boldsymbol{x})+\frac{1}{2}\right)\right]. \tag{8.13}$$

Hence, the ideal classifier is

$$\begin{aligned} h_t(\boldsymbol{x}) &= \arg\min_h \ell_{\exp}(H_{t-1}+h \mid \mathcal{D}) \\ &= \arg\min_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\left(1-f(\boldsymbol{x})h(\boldsymbol{x})+\frac{1}{2}\right)\right] \\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}f(\boldsymbol{x})h(\boldsymbol{x})\right] \\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}f(\boldsymbol{x})h(\boldsymbol{x})\right], \end{aligned} \tag{8.14}$$

where $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]$ is a constant. Let $\mathcal{D}_t$ denote a distribution

$$\mathcal{D}_t(\boldsymbol{x}) = \frac{\mathcal{D}(\boldsymbol{x})e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}. \tag{8.15}$$

According to the definition of mathematical expectation, the ideal classifier is equivalent to

$$\begin{aligned} h_t(\boldsymbol{x}) &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}f(\boldsymbol{x})h(\boldsymbol{x})\right] \\ &= \arg\max_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}[f(\boldsymbol{x})h(\boldsymbol{x})]. \end{aligned} \tag{8.16}$$

Since $f(\boldsymbol{x}), h(\boldsymbol{x}) \in \{-1, +1\}$, we have

$$f(\boldsymbol{x})h(\boldsymbol{x}) = 1 - 2\mathbb{I}(f(\boldsymbol{x}) \neq h(\boldsymbol{x})), \tag{8.17}$$

and the ideal classifier is

$$h_t(\boldsymbol{x}) = \arg\min_h \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_t}\left[\mathbb{I}(f(\boldsymbol{x}) \neq h(\boldsymbol{x}))\right]. \tag{8.18}$$

From (8.18), we see that the ideal classifier h_t minimizes the classification error under the distribution $\mathcal{D}_t$. Therefore, the weak classifier at round t is trained on the distribution $\mathcal{D}_t$, and its classification error should be less than 0.5 for $\mathcal{D}_t$. This idea is similar to the *residual approximation* to some extent. Considering the relationship between $\mathcal{D}_t$ and $\mathcal{D}_{t+1}$, we have

$$\begin{aligned}\mathcal{D}_{t+1}(\boldsymbol{x}) &= \frac{\mathcal{D}_t(\boldsymbol{x})e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]} \\ &= \frac{\mathcal{D}_t(\boldsymbol{x})e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})}}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]} \\ &= \mathcal{D}_t(\boldsymbol{x}) \cdot e^{-f(\boldsymbol{x})\alpha_t h_t(\boldsymbol{x})} \frac{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_{t-1}(\boldsymbol{x})}\right]}{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[e^{-f(\boldsymbol{x})H_t(\boldsymbol{x})}\right]},\end{aligned} \tag{8.19}$$

which is the update rule of a sample distribution as in line 7 of ◘ Algorithm 8.1.

From (8.11) and (8.19), we can see that the AdaBoost algorithm, as shown in ◘ Algorithm 8.1, can be derived by iteratively optimizing the exponential loss function based on an additive model.

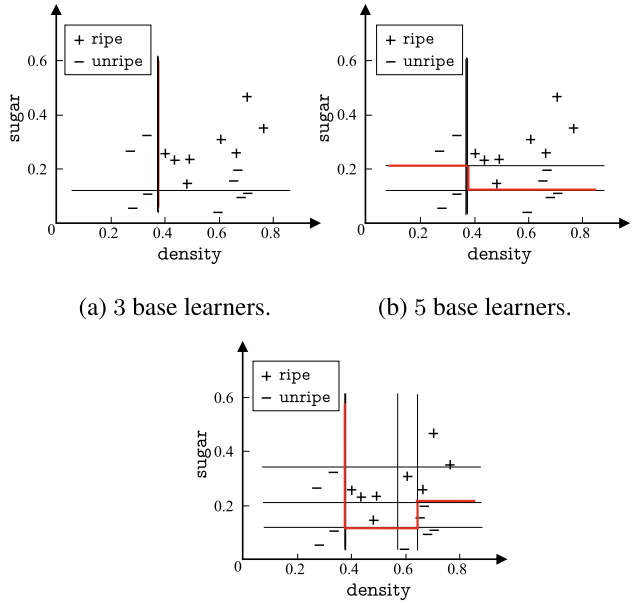
Boosting algorithms require the base learners to learn from specified sample distributions, and this is often accomplished by *re-weighting*; that is, in each round, a new weight is assigned to a training sample according to the new sample distribution. For base learning algorithms that do not accept weighted samples, *re-sampling* can be used; that is, in each round, a new training set is sampled from the new sample distribution. In general, there is not much difference between re-weighting and re-sampling in terms of prediction performance. Note that in each round, there is a sanity check on whether the current base learner satisfies some basic requirements. For example, line 5 of ◘ Algorithm 8.1 checks whether the current base learner is better than random guessing. If the requirements are not met, the current base learner is discarded and the learning process stops. In such cases, the number of rounds could still be far from the pre-specified limit T, which may lead to unsatisfactory performance due to the small number of base learners in the ensemble. However, if the re-sampling method is used, there is an option to "restart" to avoid early termination (Kohavi and Wolpert 1996). More specifically, a new training set is sampled according to the sample distribution again after discarding the current disqualified base learner, and then an alternative base learner is trained such that the learning process can continue to finish T rounds.

From the perspective of bias-variance decomposition, Boosting mainly focuses on reducing bias, and this is why an ensemble of learners with weak generalization ability can be so powerful. To give a more concrete illustration, we use decision stumps as the base learner and run the AdaBoost algorithm on the watermelon data set 3.0α (◘ Table 4.5). The decision boundaries of ensembles of different sizes together with the corresponding base learners are illustrated in ◘ Figure 8.3.

See Sect. 2.5 for the bias-variance decomposition.

A decision stump is a decision tree with a single layer. See Sect. 4.3.

The size refers to the number of base learners in an ensemble.



(a) 3 base learners. (b) 5 base learners.

(c) 11 base learners.

Fig. 8.3 AdaBoost on the watermelon data set 3.0α with ensemble sizes 3, 5, and 11. The decision boundaries of the ensemble and base learners are shown in red and black, respectively

8.3 Bagging and Random Forest

From Sect. 8.1, we know that the generalization ability of an ensemble depends on the independence of base learners. Though strict independence is not possible in practice, we can still make the learners as different as possible. One way of creating different base learners is to partition the original training set into several non-overlapped subsets and use each subset to train a base learner. Because the training subsets are different, the trained base learners are likely to be different as well. However, if the subsets are totally different, then it implies that each subset contains only a small portion of the original training set, possibly leading to poor base learners. Since a good ensemble requires each base learner to be reasonably good, we often allow the subsets to overlap such that each of them contains sufficient samples.

8.3.1 Bagging

Bagging stands for Bootstrap AGGregaING.

See Sect. 2.2.3 for bootstrap sampling.

Bagging (Breiman 1996a) is a representative method of parallel ensemble learning based on bootstrap sampling. It works as follows: given a data set with m samples, we randomly pick one sample and copy it to the sampling set; we keep it in the original data set such that it still has a chance to be picked up next time; repeating this process m times gives a data set containing m samples, where some of the original samples may appear more than once while some may never appear. From (2.1), we know that approximately 63.2% of the original samples will appear in the data set.

Applying the above process T times produces T data sets, and each contains m samples. Then, the base learners are trained on these data sets and combined. Such a procedure is the basic workflow of Bagging. When combining the predictions of base learners, Bagging adopts the simple voting method for classification tasks and the simple averaging method for regression tasks. When multiple classes receive the same number of votes, we can choose one at random or further investigate the confidence of votes. The Bagging algorithm is given in ◘ Algorithm 8.2.

That is, all base learners are equally weighted through voting or averaging.

Algorithm 8.2 Bagging

Input: Training set: $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
Base learning algorithm $\mathcal{L}$;
Number of training rounds T.

Process:

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $h_t = \mathcal{L}(D, \mathcal{D}_{bs})$.
3: **end for**

Output: $H(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y)$.

$\mathcal{D}_{bs}$ is the distribution of a data set generated by bootstrap.

Suppose that the computational complexity of a base learner is $O(m)$, then the complexity of Bagging is roughly $T(O(m) + O(s))$, where $O(s)$ is the complexity of voting or averaging. Since the complexity $O(s)$ is low and T is a constant that is often not too large, Bagging has the same order of complexity as the base learner, that is, Bagging is an efficient ensemble learning algorithm. Besides, unlike the standard AdaBoost, which only applies to binary classification, Bagging can be applied to multiclass classification and regression without modification.

For AdaBoost, modifications are needed to enable multiclass classification or regression. See (Zhou 2012) for some variants.

The bootstrap sampling brings Bagging another advantage: since each base learner only uses roughly 63.2% of the original training samples for training, the remaining 36.8% samples can be used as a validation set to get an *out-of-bag esti-*

mate (Breiman 1996a; Wolpert and Macready 1999) of the generalization ability. To get the out-of-bag estimate, we need to track the training samples used by each base learner. Let D_t denote the set of samples used by the learner h_t, and $H^{\text{oob}}(\boldsymbol{x})$ denote the out-of-bag prediction of the sample $\boldsymbol{x}$, that is, considering only the predictions made by base learners that did not use the sample $\boldsymbol{x}$ for training. Then, we have

See Sect. 2.2.3 for out-of-bag estimate.

$$H^{\text{oob}}(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} \mathbb{I}(h_t(\boldsymbol{x}) = y) \cdot \mathbb{I}(\boldsymbol{x} \notin D_t), \tag{8.20}$$

and the out-of-bag estimate of the generalization error of Bagging is

$$\epsilon^{\text{oob}} = \frac{1}{|D|} \sum_{(\boldsymbol{x},y) \in D} \mathbb{I}(H^{\text{oob}}(\boldsymbol{x}) \neq y). \tag{8.21}$$

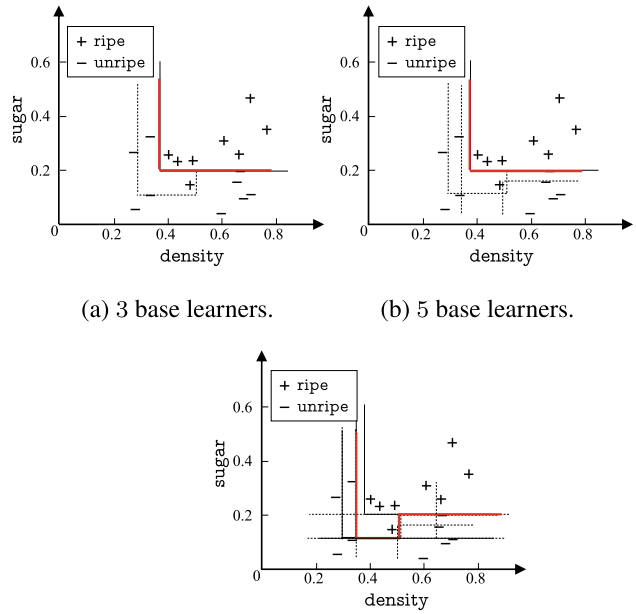
Other than estimating the generalization errors, the out-of-bag samples also have many other uses. For example, when the base learners are decision trees, the out-of-bag samples can be used for pruning branches or estimating the posterior probability of each node, and this is particularly useful when a node contains no training samples. When the base learners are neural networks, the out-of-bag samples can be used to assist early stopping to reduce the risk of overfitting.

From the perspective of bias-variance decomposition, Bagging helps to reduce the variance, and this is particularly useful for unpruned decision trees or neural networks that are unstable to data manipulation. To give a more concrete illustration, we use decision trees based on information gain as base learners to run the Bagging algorithm on the watermelon data set 3.0α (◘ Table 4.5). The decision boundaries of ensembles of different sizes together with the respective base learners are illustrated in ◘ Figure 8.4.

See Sect. 2.5 for the bias-variance decomposition.

8.3.2 Random Forest

Random Forest (RF) (Breiman 2001a) is an extension of Bagging, where randomized feature selection is introduced on top of Bagging. Specifically, traditional decision trees select an optimal split feature from the feature set of each node, whereas RF selects from a subset of k features randomly generated from the feature set of the node. The parameter k controls the randomness, where the splitting is the same as in traditional decision trees if $k = d$, and a split feature is randomly



(a) 3 base learners. (b) 5 base learners.

(c) 11 base learners.

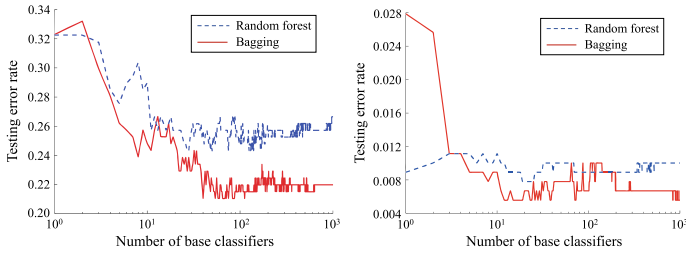
Fig. 8.4 Bagging on the watermelon data set 3.0α with ensemble sizes 3, 5, and 11. The decision boundaries of the ensemble and base learners are shown in red and black, respectively

selected if $k = 1$. Typically, the recommended value of k is $k = \log_2 d$ (Breiman 2001a).

Despite its ease of implementation and low computational cost, RF often shows surprisingly good performance in real-world applications and is honored as a representative of state-of-the-art ensemble learning methods. With a small modification, RF introduces feature-based "diversity" to Bagging by feature manipulation, while Bagging considers sample-based diversity only. RF further enlarges the difference between base learners, which leads to ensembles with better generalization ability.

See Sect. 8.5.3 for sample manipulation and feature manipulation.

The convergence property of RF is comparable to that of Bagging. As illustrated in ◘ Figure 8.5, RF usually starts with a poor performance at the beginning, especially when there is only one base learner in the ensemble since the feature manipulation reduces the performance of each base learner. Eventually, however, RF often converges to a lower generalization error after adding more base learners to the ensemble. It is worth mentioning that it is often more efficient to train an RF than applying Bagging since RF uses "randomized" decision trees that evaluate subsets of features for splitting, whereas



(a) The glass data set. (b) The auto-mpg data set.

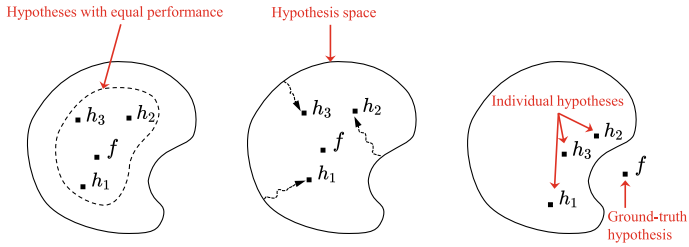
Fig. 8.5 The impact of the ensemble size on RF and Bagging on two UCI data sets

Bagging uses "deterministic" decision trees that evaluate all features for splitting.

8.4 Combination Strategies

As illustrated in ◘ Figure 8.6, combining individual learners is beneficial from the following three perspectives (Dietterich 2000):

- Statistical perspective: since the hypothesis space is often large, there are usually multiple hypotheses achieving the same performance. If a single learner is chosen, then the generalization performance solely depends on the quality of this single learner. Combining multiple learners will reduce the risk of incorrectly choosing a single poor learner.
- Computational perspective: learning algorithms are often stuck in local optimum, which may lead to poor generalization performance even when there is abundant training data. If choosing another way by repeating the learning process multiple times, the risk of being stuck in a terrible local



(a) Statistical reason. (b) Computational reason. (c) Representational reason.

Fig. 8.6 The benefits of combining learners (Dietterich 2000)

optimum is reduced, though it is not guaranteed to be the global optimum.

- Representational perspective: sometimes, the ground-truth hypothesis is not represented by any candidates in the hypothesis space of the current learning algorithm. In such cases, it is for sure that a single learner will not find the ground-truth hypothesis. On the other hand, the hypothesis space extends by combining multiple learners, and hence it is more likely to find a better approximation to the ground-truth hypothesis.

Suppose an ensemble contains T individual learners $\{h_1, h_2, \ldots, h_T\}$, where $h_i(\boldsymbol{x})$ is the output of h_i on sample $\boldsymbol{x}$. The rest of this section introduces several typical strategies for combing h_i.

8.4.1 Averaging

Averaging is the most commonly used combination strategy for numerical output $h_i(\boldsymbol{x}) \in \mathbb{R}$. Typical averaging methods include *simple averaging*

$$H(\boldsymbol{x}) = \frac{1}{T} \sum_{i=1}^{T} h_i(\boldsymbol{x}) \tag{8.22}$$

and *weighted averaging*

$$H(\boldsymbol{x}) = \sum_{i=1}^{T} w_i h_i(\boldsymbol{x}), \tag{8.23}$$

where w_i is the weight of individual learner h_i, and typically $w_i \geqslant 0, \sum_{i=1}^{T} w_i = 1$. Simple averaging is a special case of weighted averaging when $w_i = 1/T$.

In a study of Stacking regression, (Breiman 1996b) found that the weights must be non-negative to ensure the ensemble performs better than the best single learner. Hence, the non-negative constraint is often applied to the weights of learners.

Weighted averaging has been widely used since the 1950s (Markowitz 1952), and it was first used in ensemble learning in Perrone and Cooper (1993). Weighted averaging plays an important role in ensemble learning since other combination methods can all be viewed as its special cases or variants. Indeed, weighted averaging can be regarded as a fundamental motivation of ensemble learning studies. Given a set of base learners, different ensemble learning methods can be regarded as different ways of assigning the weights.

Typically, weights are learned from training data. However, the learned weights are often unreliable due to data insufficiency or noise. This is particularly true for large ensembles with many base learners because trying to learn too many

weights can easily lead to overfitting. In fact, many empirical studies and applications have shown that weighted averaging is not necessarily better than simple averaging (Xu et al. 1992; Ho et al. 1994; Kittler et al. 1998). Generally speaking, the weighted averaging method is a better choice when individual learners have considerable different performance, while the simple averaging method is preferred when individual learners share similar performance.

For example, setting the weights inversely proportional to the estimated errors of individual learners.

8.4.2 Voting

For classification, a learner h_i predicts a class label from a set of N class labels $\{c_1, c_2, \ldots, c_N\}$, and a common combination strategy is voting. For ease of discussion, let the N-dimensional vector $(h_i^1(\boldsymbol{x}); h_i^2(\boldsymbol{x}); \ldots; h_i^N(\boldsymbol{x}))$ denote the output of h_i on the sample $\boldsymbol{x}$, where $h_i^j(\boldsymbol{x})$ is the output of h_i on $\boldsymbol{x}$ for the class label c_j. Then, we can define the following voting methods:

- Majority voting:

$$H(\boldsymbol{x}) = \begin{cases} c_j, & \text{if } \sum_{i=1}^{T} h_i^j(\boldsymbol{x}) > 0.5 \sum_{k=1}^{N} \sum_{i=1}^{T} h_i^k(\boldsymbol{x}); \\ \text{reject}, & \text{otherwise.} \end{cases} \tag{8.24}$$

  That is, output the class label that receives more than half of the votes or refuses to predict if none of the class labels receive more than half of the votes.
- Plurality voting:

$$H(\boldsymbol{x}) = c_{\arg\max_j \sum_{i=1}^{T} h_i^j(\boldsymbol{x})}. \tag{8.25}$$

  That is, output the class label that receives the most votes and randomly select one in case of a tie.
- Weighted voting:

$$H(\boldsymbol{x}) = c_{\arg\max_j \sum_{i=1}^{T} w_i h_i^j(\boldsymbol{x})} \tag{8.26}$$

  That is, a plurality voting with weights assigned to learners, where w_i is the weight of h_i, and typically $w_i \geqslant 0$ and $\sum_{i=1}^{T} w_i = 1$ like the constraints on w_i in weighted averaging (8.23).

The standard majority voting (8.24) offers a "reject" option, which is an effective mechanism for tasks requiring reliability (e.g., medical diagnosis). However, if it is compulsory to make a prediction, then plurality voting can be used instead. For tasks

that do not allow rejections, both the majority and the plurality voting methods are called majority voting, or just voting.

Equations (8.24)¬(8.26) assume no particular value type for the output $h_i^j(\boldsymbol{x})$, but two common value types are

- Class label $h_i^j(\boldsymbol{x}) \in \{0, 1\}$: the output is 1 if h_i predicts the sample $\boldsymbol{x}$ as class c_j, and 0 otherwise. The corresponding voting is known as *hard voting*.
- Class probability $h_i^j(\boldsymbol{x}) \in [0, 1]$: an estimate to the posterior probability $P(c_j \mid \boldsymbol{x})$. The corresponding voting is known as *soft voting*.

Different types of $h_i^j(\boldsymbol{x})$ should not be mixed. For some learners, the class labels come with confidence values that can be converted into class probabilities. However, if the confidence values are unnormalized (e.g., the margin values in SVM), then, before using the confidence values as probabilities, we need to apply *calibration* techniques, such as Platt scaling (Platt 2000) and isotonic regression (Zadrozny and Elkan 2001). Interestingly, though the class probabilities are often imprecise, the performance of combining class probabilities usually outperforms that of combining class labels. Note that the class probabilities are not comparable if different types of base learners are used. In such cases, the class probabilities can be converted into class labels before voting, e.g., setting the largest $h_i^j(\boldsymbol{x})$ as 1 and the others as 0.

For example, a heterogeneous ensemble consists of different types of base learners.

8.4.3 Combining by Learning

When there is abundant training data, a more powerful combination strategy is *combining by learning*: using a meta-learner to combine the individual learners. A representative of such methods is *Stacking* (Wolpert 1992; Breiman 1996b). Here, we call the individual learners as first-level learners and call the learners performing the combination as second-level learners or meta-learners.

Stacking is also a well-known ensemble learning method on its own, and many ensemble learning methods can be viewed as its special cases or variants. We introduce it here since it is also a special combination method.

Stacking starts by training the first-level learners using the original training set and then "generating" a new data set for training the second-level learner. In the new data set, the outputs of the first-level learners are used as the input features, while the labels from the original training set remain unchanged. The Stacking algorithm is given in ◘ Algorithm 8.3, where the first-level learners are assumed heterogeneous.

The first-level learners can also be homogeneous.

Algorithm 8.3 Stacking

Input: Training set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
First-level learning algorithms $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_T$;
Second-level learning algorithm $\mathcal{L}$.

Process:

```
for t = 1, 2, ..., T do
    h_t = L_t(D);                          Generate first-level learner h_t using first-level learning algorithm L_t.
end for
D' = ∅;
for i = 1, 2, ..., m do
    for t = 1, 2, ..., T do
        z_it = h_t(x_i);
    end for
    D' = D' ∪ ((z_i1, z_i2, ..., z_iT), y_i);   Generate second-level training set.
end for
h' = L(D').                                Generate second-level learner h' using second-level learning algorithm on D'.
```

Output: $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_T(\boldsymbol{x}))$.

When generating the second-level training set from the first-level learners, there would be a high risk of overfitting if the generated training set contains the exact training samples used by the first-level learners. Hence, we often employ cross-validation or leave-one-out methods such that the samples that have not been used for the training of the first-level learners are used for generating the training set for the second-level learner. Taking k-fold cross-validation as an example. The original training set is partitioned into k roughly equal-sized partitions $D_1, D_2, \ldots, D_k$. Denote D_j and $\overline{D}_j = D \backslash D_j$ as the testing set and the training set of the jth fold, respectively. For T first-level learning algorithms, the first-level learner $h_t^{(j)}$ is obtained by applying the tth learning algorithm on the subset $\overline{D}_j$. Let $z_{it} = h_t^{(j)}(\boldsymbol{x}_i)$, then, for each sample $\boldsymbol{x}_i$ in D_j, the output from the first-level learners can be written as $\boldsymbol{z}_i = (z_{i1}; z_{i2}; \ldots; z_{iT})$, which is used as the input for the second-level learner with the original label y_i. By finishing the entire cross-validation process on the T first-level learners, we obtain a data set $D' = \{(\boldsymbol{z}_i, y_i)\}_{i=1}^{m}$ for training the second-level learner.

The generalization ability of a Stacking ensemble heavily depends on the representation of the input features for the second-level learner as well as the choice of the second-level learning algorithm. Existing studies show that the Multi-response Linear Regression (MLR) is a good second-level learning algorithm when the first-level learners output class probabilities (Ting and Witten 1999), and the MLR can do an even better job when different sets of features are used (Seewald 2002).

MLR is a classifier based on linear regression. It builds a linear regression model for each class by setting the output of samples belonging to this class as 1 and 0 for the rest. Then, a testing sample is classified as the class with the highest predicted output value.

This is how the StackingC algorithm is implemented in WEKA.

Bayes Model Averaging (BMA) assigns weights to different learners based on posterior probabilities, and it can be regarded as a special implementation of the weighted averaging method. Clarke (2003) compared Stacking and BMA and showed that, in theory, if there is limited noise in the data and the correct data generating model is among the models under consideration, then BMA is guaranteed to be no worse than Stacking. In practice, however, the correct data generating model is by no way guaranteed in the models under consideration, and sometimes even hard to be approximated by the models under consideration. Hence, Stacking usually performs better than BMA in practice since it is more robust than BMA; besides, BMA is also more sensitive to approximation errors.

8.5 Diversity

8.5.1 Error-Ambiguity Decomposition

As mentioned in Sect. 8.1, to have an ensemble with strong generalization ability, the individual learners should be "accurate and diverse". This section is devoted to providing a brief theoretical analysis of this aspect.

Let us assume that the individual learners $h_1, h_2, \ldots, h_T$ are combined via weighted averaging (8.23) into an ensemble for the regression task $f : \mathbb{R}^d \mapsto \mathbb{R}$. Then, for the sample $\boldsymbol{x}$, the *ambiguity* of the learner h_i is defined as

$$A(h_i|\boldsymbol{x}) = (h_i(\boldsymbol{x}) - H(\boldsymbol{x}))^2, \tag{8.27}$$

and the *ambiguity* of the ensemble is defined as

$$\begin{aligned}\overline{A}(h \mid \boldsymbol{x}) &= \sum_{i=1}^{T} w_i A(h_i \mid \boldsymbol{x}) \\ &= \sum_{i=1}^{T} w_i (h_i(\boldsymbol{x}) - H(\boldsymbol{x}))^2.\end{aligned} \tag{8.28}$$

The ambiguity term represents the degree of disagreement among individual learners on the sample $\boldsymbol{x}$, which reflects the level of diversity in some sense. The squared errors of the individual learner h_i and the ensemble H are, respectively,

$$E(h_i \mid \boldsymbol{x}) = (f(\boldsymbol{x}) - h_i(\boldsymbol{x}))^2, \tag{8.29}$$

$$E(H \mid \boldsymbol{x}) = (f(\boldsymbol{x}) - H(\boldsymbol{x}))^2. \tag{8.30}$$

Let $\overline{E}(h \mid \boldsymbol{x}) = \sum_{i=1}^{T} w_i \cdot E(h_i \mid \boldsymbol{x})$ denote the weighted average error of individual learners, then, we have

$$\begin{aligned}\overline{A}(h \mid \boldsymbol{x}) &= \sum_{i=1}^{T} w_i E(h_i \mid \boldsymbol{x}) - E(H \mid \boldsymbol{x}) \\ &= \overline{E}(h \mid \boldsymbol{x}) - E(H \mid \boldsymbol{x}). \end{aligned} \tag{8.31}$$

Since (8.31) holds for every sample $\boldsymbol{x}$, for all samples we have

$$\sum_{i=1}^{T} w_i \int A(h_i \mid \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} = \sum_{i=1}^{T} w_i \int E(h_i \mid \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} - \int E(H \mid \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}, \tag{8.32}$$

where $p(\boldsymbol{x})$ is the probability density of the sample $\boldsymbol{x}$. Similarly, the generalization error and the ambiguity term of the learner h_i on all samples are, respectively,

Here, we abbreviate $E(h_i)$ and $A(h_i)$ as E_i and A_i.

$$E_i = \int E(h_i \mid \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}, \tag{8.33}$$

$$A_i = \int A(h_i \mid \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}. \tag{8.34}$$

The generalization error of the ensemble is

Here, we abbreviate $E(H)$ as E.

$$E = \int E(H \mid \boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}. \tag{8.35}$$

Let $\overline{E} = \sum_{i=1}^{T} w_i E_i$ denote the weighted average error of individual learners, and $\overline{A} = \sum_{i=1}^{T} w_i A_i$ denote the weighted average ambiguity of individual learners. Then, substituting (8.33)¬(8.35) into (8.32) gives

$$E = \overline{E} - \overline{A}. \tag{8.36}$$

The elegant Equation (8.36) clearly shows that the generalization ability of an ensemble depends on the accuracy and diversity of individual learners. The above analysis is known as the *error-ambiguity decomposition* (Krogh and Vedelsby 1995).

Given the above analysis, some readers may propose that we can easily obtain the optimal ensemble by optimizing $\overline{E} - \overline{A}$ directly. Unfortunately, direct optimization of $\overline{E} - \overline{A}$ is hard in practice, not only because both terms are defined in the entire sample space but also because $\overline{A}$ is not a diversity measure that is directly operable; it is only known after we have the ensemble fully constructed. Besides, we should note that the above derivation process is only applicable to regression and is difficult to extend to classification.

8.5.2 Diversity Measures

Diversity measures, as the name suggests, are for measuring the diversity of individual learners in an ensemble. A typical approach is to measure the pairwise similarity or dissimilarity between learners.

Given a data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$, the *contingency table* of the classifiers h_i and h_j for binary classification (i.e., $y_i \in \{-1, +1\}$) is

See Sect. 2.3.2 for confusion matrix.

	$h_i = +1$	$h_i = -1$
$h_j = +1$	a	c
$h_j = -1$	b	d

where a is the number of samples predicted as positive by both h_i and h_j, and similarly for b, c, and d. With the contingency table, we list some representative diversity measures below as follows:

- Disagreement measure:

$$\text{dis}_{ij} = \frac{b + c}{m}, \tag{8.37}$$

  where the range of dis_{ij} is $[0, 1]$, and a larger value indicates a higher diversity.
- Correlation coefficient:

$$\rho_{ij} = \frac{ad - bc}{\sqrt{(a + b)(a + c)(c + d)(b + d)}}, \tag{8.38}$$

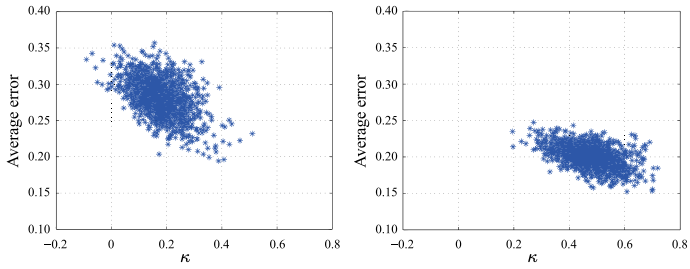
  where the range of ρ_{ij} is $[-1, 1]$. When h_i and h_j are unrelated, the value is 0. The value is positive when h_i and h_j are positively correlated, and negative when h_i and h_j are negatively correlated.
- Q-statistic:

$$Q_{ij} = \frac{ad - bc}{ad + bc}, \tag{8.39}$$

  where Q_{ij} has the same sign as the correlation coefficient ρ_{ij}, and $\left|Q_{ij}\right| \geqslant \left|\rho_{ij}\right|$.
- κ-statistic:

$$\kappa = \frac{p_1 - p_2}{1 - p_2}, \tag{8.40}$$

  where p_1 is the probability that these two classifiers agree on the prediction, and p_2 is the probability of agreement by



(a) AdaBoost ensemble. (b) Bagging ensemble.

Fig. 8.7 The κ-error diagrams on the UCI *tic-tac-toe* data set; every ensemble consists of 50 C4.5 decision trees

chance. Given a data set D, these two probabilities can be estimated as

$$p_1 = \frac{a+d}{m}, \tag{8.41}$$

$$p_2 = \frac{(a+b)(a+c)+(c+d)(b+d)}{m^2}. \tag{8.42}$$

If the two classifiers agree on all samples in D, then $\kappa = 1$; if the two classifiers agree by chance, then $\kappa = 0$. Normally, κ is a non-negative value, but it can be negative in rare cases where the agreement is even less than what is expected by chance.

The aforementioned diversity measures are all pairwise measures that can be easily visualized with 2-D plots. For example, the well-known "κ-error diagrams", as illustrated in ◘ Figure 8.7, plot each pair of classifiers as a point in a scatter plot, where the x-axis represents their κ value, and the y-axis represents their average error. The closer the points to the top of the plot, the lower the accuracy of individual classifiers; the closer the points to the right-hand side of the plot, the less the diversity of individual classifiers.

8.5.3 Diversity Generation

Our previous discussions showed that effective ensemble learning relies on diverse individual learners, but how can we enhance diversity? The general idea is to introduce some randomness into the learning process by manipulating the data samples, input features, output representations, and algorithm parameters.

- **Data sample manipulation**: given the original data set, we can manipulate it by generating multiple subsets to train different individual learners. Data sample manipulation is often based on sampling methods such as bootstrap sampling used by Bagging and sequential sampling used by AdaBoost. This approach is widely adopted due to its simplicity and effectiveness. Some base learners, such as decision trees and neural networks, are called *unstable base learners* since they are sensitive to data sample manipulation, that is, a small change to the training samples leads to significant changes to the base learners. Hence, data sampling manipulation is particularly effective for unstable base learners. On the other hand, some base learners, such as linear learners, SVM, naïve Bayes, and k-nearest neighbors, are *stable base learners* that are insensitive to data sample manipulation. Hence, other mechanisms are needed, such as input feature manipulation.
- **Input feature manipulation**: the training samples are usually described by a set of features, where different *subspaces* (subsets of features) offer different views of the data, and the individual learners trained on different subspaces will be different. The *Random Subspace* algorithm (Ho 1998) is a well-known ensemble method that relies on input feature manipulation. As illustrated in ◘ Algorithm 8.4, the random subspace algorithm extracts multiple subsets of features and uses each subset to train a base learner. For data with many redundant features, applying the random subspace algorithm improves not only the diversity but also the efficiency. Meanwhile, due to the redundant features, the performance of learners trained with a subset of features is often acceptable. Nevertheless, the input feature manipulation is often not a good choice when there is a limited number of features or low feature redundancy.

8

A subspace usually refers to a lower dimensional feature space mapped from the original higher dimensional feature space. The features describing the lower dimensional space are not necessarily the original features but transformed from the original features. See Chap. 10.

Algorithm 8.4 Random Subspace

Input: Training set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
Base learning algorithm $\mathcal{L}$;
Number of base learners T;
Number of features in subspace d'.

Process:

1: **for** $t = 1, 2, \ldots, T$ **do**
2: $\quad \mathcal{F}_t = \mathrm{RS}(D, d')$;
3: $\quad D_t = \mathrm{Map}_{\mathcal{F}_t}(D)$;
4: $\quad h_t = \mathcal{L}(D_t)$.
5: **end for**

Output: $H(\boldsymbol{x}) = \underset{y \in \mathcal{Y}}{\arg\max} \sum_{t=1}^{T} \mathbb{I}\left(h_t\left(\mathrm{Map}_{\mathcal{F}_t}(\boldsymbol{x})\right) = y\right)$.

d' is smaller than the number of original features d.

$\mathcal{F}_t$ includes d' randomly selected features, and D_t keeps only the features in $\mathcal{F}_t$.

- **Output representation manipulation**: diversity can also be enhanced by manipulating the output representations. One method is making small changes to the class labels. For example, the Flipping Output method (Breiman 2000) randomly flips the class labels of some samples. Alternatively, we can transform the output representation. For example, the Output Smearing (Breiman 2000) method converts classification outputs into regression outputs before constructing individual learners. Besides, we can also divide the original task into multiple subtasks that can be solved in parallel. For example, the ECOC method (Dietterich and Bakiri 1995) employs error-correcting output codes to divide a multiclass classification task into multiple binary classification tasks to train base learners.

See Sect. 3.5 for ECOC.

- **Algorithm Parameter Manipulation**: base learning algorithms often have parameters, such as the number of hidden neurons and initial connection weights in neural networks. By randomly setting the parameters, we can obtain individual learners that are quite different. For example, the *Negative Correlation* method (Liu and Yao 1999) employs a regularization term to enforce individual neural networks to use different parameters. For algorithms with limited parameters, we can replace some internal components with alternatives for manipulation. For example, using different split feature selection methods in individual decision trees. Note that when a single learner is to be used, we often determine the best parameters by training multiple learners with different parameter settings (e.g., cross-validation), though only one of them is selected. In contrast, ensemble learning utilizes all of these trained learners, and hence the practical computational cost of creating an ensemble is not much higher than creating a single learner.

Different diversity generation mechanisms can be used together. For example, the random forest introduced in Sect. 8.3.2 employs both the data sample manipulation and input feature manipulation. Some methods use more mechanisms at the same time (Zhou 2012).

8.6 Further Reading

Zhou (2012) is the main recommended reading about ensemble learning, which provides more detailed discussions on the content covered in this chapter. Kuncheva (2004), Rokach (2010b) are also good references. Schapire and Freund (2012) is a book dedicated to Boosting.

Boosting was initially developed in Schapire (1990) to provide a constructive answer to an important theoretical question proposed by Kearns and Valiant (1989): "Is weak learnability equivalent to strong learnability?" The original Boosting algorithm has only theoretical significance. After several years of hard work, Freund and Schapire (1997) proposed AdaBoost. Due to the importance of this work, Yoav Freund and Robert Schapire won the 2003 Gödel Prize, a prestigious award in theoretical computer science. Different ensemble learning methods often have significantly different working mechanisms and theoretical properties. For example, from the bias-variance decomposition point of view, Boosting focuses on reducing the bias while Bagging focuses on reducing the variance. Some attempts, such as MultiBoosting (Webb 2000), have been made to take advantage of both approaches. There already exist considerable theoretical research works on Boosting and Bagging, which can be found in Chaps. 2 and 3 of Zhou (2012).

The derivation given in Sect. 8.2 came from the *statistical view* (Friedman et al. 2000). This school of theory believes that AdaBoost is essentially optimizing exponential loss based on an additive model with an iterative process like Newton's method. It inspires the replace of the iterative method to other optimization methods, leading to AdaBoost variants, such as GradientBoosting (Friedman 2001) and LPBoost (Demiriz et al. 2008). However, there are significant differences between the practical behavior of AdaBoost and that derived from this school of theory (Mease and Wyner 2008), especially it could not explain the fundamentally amazing phenomenon of why AdaBoost does not overfit. Therefore, many researchers argue that the statistical view only explains a learning process that is similar to AdaBoost but not AdaBoost itself. The *margin theory* (Schapire et al. 1998) offers an intuitive explanation of this important phenomenon, but it has been in challenging in the past 15 years until recent studies established final conclusions and shed light on the design of new learning methods. See Zhou (2014) for more information.

A more strict description of this phenomenon is "Why AdaBoost can further improve the generalization ability even after the training error reaches zero?" Indeed, if training continues, overfitting will eventually happen.

In addition to the basic combination methods introduced in this chapter, many other advanced methods exist, including the methods based on Dempster-Shafer theory, Dynamic Classifier Selection, and mixture of experts. This chapter only introduced pairwise diversity measures. Kuncheva and Whitaker (2003), Tang et al. (2006) showed that most existing diversity measures have drawbacks. The understanding of diversity is considered as the holy grail problem in ensemble learning research. More information about combination methods and diversity can be found in Chaps. 4 and 5 of Zhou (2012).

After obtaining an ensemble, trying to eliminate some individuals to get a smaller ensemble is called *ensemble pruning*,

which helps reduce the storage and prediction time costs. Early studies mainly focused on pruning in sequential ensemble methods, but it was found that pruning hurts the generalization performance (Rokach 2010a). Zhou et al. (2002) disclosed that the pruning of parallel ensembles can reduce the ensemble size while improving the generalization ability, which opened the door to optimization-based ensemble pruning techniques. More information about this topic can be found in Chap. 6 of Zhou (2012).

Pruning of parallel ensembles is also called *selective ensemble*. However, nowadays, the term selective ensemble is often used as a synonym of ensemble pruning, which is also called *ensemble selection*.

Chapters 7 and 8 in Zhou (2012) discussed the use of ensemble learning in other learning tasks such as clustering, semi-supervised learning, and cost-sensitive learning. Indeed, ensemble learning is now widely used in most learning tasks, and almost all winning solutions in the KDD Cups (a famous data mining competition) employed ensemble learning.

Since an ensemble consists of multiple learners, it is a black box, even if the individual learners have good interpretability. There are attempts to improve the interpretability of ensembles, such as converting an ensemble to a single model or extracting symbolic rules from an ensemble. Research on this topic derived *twice-learning* techniques, such as NeC4.5 (Zhou and Jiang 2004), which can produce a single learner that outperforms the ensemble; later, similar techniques are called knowledge distillation. Besides, visualization techniques are also helpful in improving interpretability. More information on this topic can be found in Chap. 8 of Zhou (2012).

Exercises

8.1 Suppose we toss a coin with a probability of p it lands on heads and a probability of $1-p$ it lands on tails. Let $H(n)$ denote the number of heads tossing the coin n times, then the probability of getting at most k heads is

$$P(H(n) \leqslant k) = \sum_{i=0}^{k} \binom{i}{n} p^i (1-p)^{n-i}. \tag{8.43}$$

For $\delta > 0$ and $k = (p-\delta)n$, we have Hoeffding's inequality

$$P(H(n) \leqslant (p-\delta)n) \leqslant e^{-2\delta^2 n}. \tag{8.44}$$

Try to derive (8.3).

8.2 The exponential loss function is not the only consistent surrogate function for the 0/1 loss function. Considering (8.5), prove that every loss function $\ell(-f(\boldsymbol{x})H(\boldsymbol{x}))$ is a consistent surrogate function for the 0/1 loss function if $H(\boldsymbol{x})$ is monotonically decreasing in the interval $[-\infty, \delta]$ ($\delta > 0$).

8.3 Download or implement the AdaBoost algorithm, and then use unpruned decision trees as base learners to train an AdaBoost ensemble on the watermelon data set 3.0α. Compare the results with ◘ Figure 8.4.

The watermelon data set 3.0α is in ◘ Table 4.5.

8.4 GradientBoosting (Friedman 2001) is a common Boosting algorithm. Try to analyze its difference and commonality to the AdaBoost algorithm.

8.5 Implement Bagging, and then use decision stumps as base learners to train a Bagging ensemble on the watermelon data set 3.0α. Compare the results with ◘ Figure 8.4.

8.6 Analyze the reasons why Bagging is not effective in improving the performance of naïve Bayes classifiers.

8.7 Analyze the reasons why it is faster to train a random forest than training a decision tree-based Bagging ensemble.

8.8 The MultiBoosting algorithm (Webb 2000) employs AdaBoost as the base learner in Bagging, while the Iterative Bagging algorithm (Breiman 2001b) employs Bagging as the base learner in AdaBoost. Discuss the pros and cons of each method.

8.9 * Design a visualized diversity measure. Use it to evaluate the ensembles obtained in Exercises 8.3 and 8.5, and compare with the κ-error diagrams.

8.10 * Design an ensemble learning algorithm that can improve the performance of k-nearest neighbors classifiers.

Break Time

Short Story: Leo Breiman in His Green Old Age

Leo Breiman (1928–2005) was a great statistician in the twentieth century. At the end of the twentieth century, he publicly stated that the statistics community had made statistics like a branch of abstract mathematics, which divorced from the original intention of statistics. Instead, he believed that "*Statistics existed for the purposes of prediction and explanation and working with data.*" As a statistician, he claimed that his research was more about machine learning since there were more data-related challenges in this area. In fact, Breiman was an outstanding machine learning researcher, who not only developed the CART decision tree, but also made three major contributions to ensemble learning: Bagging, Random Forest, and theoretical arguments about Boosting. Interestingly, all of these were completed after he retired in 1993 from the Department of Statistics, UC Berkeley.

In the early days, Breiman obtained a degree in physics from the California Institute of Technology and decided to study philosophy at Columbia University. However, the head of the Department of Philosophy told Breiman that two of his best Ph.D. students could not find jobs, and hence Breiman changed his mind and turned to study mathematics. After earning his master's and Ph.D. degrees from UC Berkeley, he became a professor at the University of California, Los Angeles (UCLA) teaching probabilities. 7 years later, he was bored of probabilities and resigned from the position. To say goodbye to probabilities, he spent half a year at home to write a book about probabilities and then started a career as a statistical consultant in the industry for 13 years. Then, he came back to the Department of Statistics, UC Berkeley as a professor. Breiman had a variety of life experiences. He spent his sabbatical leave working for UNESCO as an educational statistician in Liberia to teach statistics for out-of-school children. He was also an amateur sculptor, and he even managed business with partners selling ice in Mexico. Random forest, which Breiman thought was the most important research outcome of his life, was developed after he was in his 70s.

References

Breiman L (1996a) Bagging predictors. Mach Learn 24(2):123–140
Breiman L (1996b) Stacked regressions. Mach Learn 24(1):49–64
Breiman L (2000) Randomizing outputs to increase prediction accuracy. Mach Learn 40(3):113–120
Breiman L (2001a) Random forests. Mach Learn 45(1):5–32
Breiman L (2001b) Using iterated bagging to debias regressions. Mach Learn 45(3):261–277
Clarke B (2003) Comparing Bayes model averaging and stacking when model approximation error cannot be ignored. J Mach Learn Res 4:683–712
Demiriz A, Bennett KP, Shawe-Taylor J (2008) Linear programming Boosting via column generation. Mach Learn 46(1–3):225–254
Dietterich TG (2000) Ensemble methods in machine learning. In: Proceedings of the 1st international workshop on multiple classifier systems (MCS), pp 1–15. Cagliari, Italy
Dietterich TG, Bakiri G (1995) Solving multiclass learning problems via error-correcting output codes. J Artif Intell Res 2:263–286
Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139
Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting (with discussions). Ann Stat 28(2):337–407
Friedman JH (2001) Greedy function approximation: a gradient Boosting machine. Ann Stat 29(5):1189–1232
Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Patt Anal Mach Intell 20(8):832–844
Ho TK, Hull JJ, Srihari SN (1994) Decision combination in multiple classifier systems. IEEE Trans Patt Anal Mach Intell 16(1):66–75
Kearns M, Valiant LG (1989) Cryptographic limitations on learning Boolean formulae and finite automata. In: Proceedings of the 21st annual ACM symposium on theory of computing (STOC), pp 433–444. Seattle, WA
Kittler J, Hatef M, Duin R, Matas J (1998) On combining classifiers. IEEE Trans Patt Anal Mach Intell 20(3):226–239
Kohavi R, Wolpert DH (1996) Bias plus variance decomposition for zero-one loss functions. In: Proceedings of the 13th international conference on machine learning (ICML), pp 275–283. Bari, Italy
Krogh A, Vedelsby J (1995) Neural network ensembles, cross validation and active learning. In: Tesauro G, Touretzky DS, Leen TK (eds) Advances in neural information processing systems 7 (NIPS). MIT Press, Cambridge, MA, pp 231–238
Kuncheva LI (2004) Combining pattern classifiers: methods and algorithms. Wiley, Hoboken, NJ
Kuncheva LI, Whitaker CJ (2003) Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Mach Learn 51(2):181–207
Liu Y, Yao X (1999) Ensemble learning via negative correlation. Neural Netw 12(10):1399–1404
Markowitz H (1952) Portfolio selection. J Financ 7(1):77–91
Mease D, Wyner A (2008) Evidence contrary to the statistical view of boosting (with discussions). J Mach Learn Res 9:131–201
Perrone MP, Cooper LN (1993) When networks disagree: ensemble method for neural networks. In: Mammone RJ (ed) Artificial neural networks for speech and vision. Chapman & Hall, New York, NY, pp 126–142

Platt J (2000) Probabilities for SV machines. In: Smola A, Bartlett P, Schölkopf B, Schuurmans D (eds) Advances in large margin classifiers. MIT Press, Cambridge, MA, pp 61–74
Rokach L (2010a) Ensemble-based classifiers. Artif Intell Rev 33(1):1–39
Rokach L (2010b) Pattern classification using ensemble methods. World Scientific, Singapore
Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
Schapire RE, Freund Y (2012) Boosting: foundations and algorithms. MIT Press, Cambridge, MA
Schapire RE, Freund Y, Bartlett P, Lee WS (1998) Boosting the margin: a new explanation for the effectiveness of voting methods. Ann Stat 26(5):1651–1686
Seewald AK (2002) How to make stacking better and faster while also taking care of an unknown weakness. In: Proceedings of the 19th international conference on machine learning (ICML), pp 554–561. Sydney, Australia
Tang EK, Suganthan PN, Yao X (2006) An analysis of diversity measures. Mach Learn 65(1):247–271
Ting KM, Witten IH (1999) Issues in stacked generalization. J Artif Intell Res 10:271–289
Webb GI (2000) MultiBoosting: a technique for combining boosting and wagging. Mach Learn 40(2):159–196
Wolpert DH (1992) Stacked generalization. Neural Netw 5(2):241–260
Wolpert DH, Macready WG (1999) An efficient method to estimate Bagging's generalization error. Mach Learn 35(1):41–55
Xu L, Krzyzak A, Suen CY (1992) Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Trans Syst Man Cybern 22(3):418–435
Zadrozny B, Elkan C (2001) Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In: Proceedings of the 18th international conference on machine learning (ICML), pp 609–616. Williamstown, MA
Zhou Z-H (2012) Ensemble methods: foundations and algorithms. Chapman & Hall/CRC, Boca Raton, FL
Zhou Z-H (2014) Large margin distribution learning. In: Proceedings of the 6th IAPR international workshop on artificial neural networks in pattern recognition (ANNPR), pp 1–11. Montreal, Canada
Zhou Z-H, Jiang Y (2004) NeC4.5: neural ensemble based C4.5. IEEE Trans Knowl Data Eng 16(6):770–773
Zhou Z-H, Wu J, Wei T (2002) Ensembling neural networks: many could be better than all. Artif Intell 137(1–2):239–263