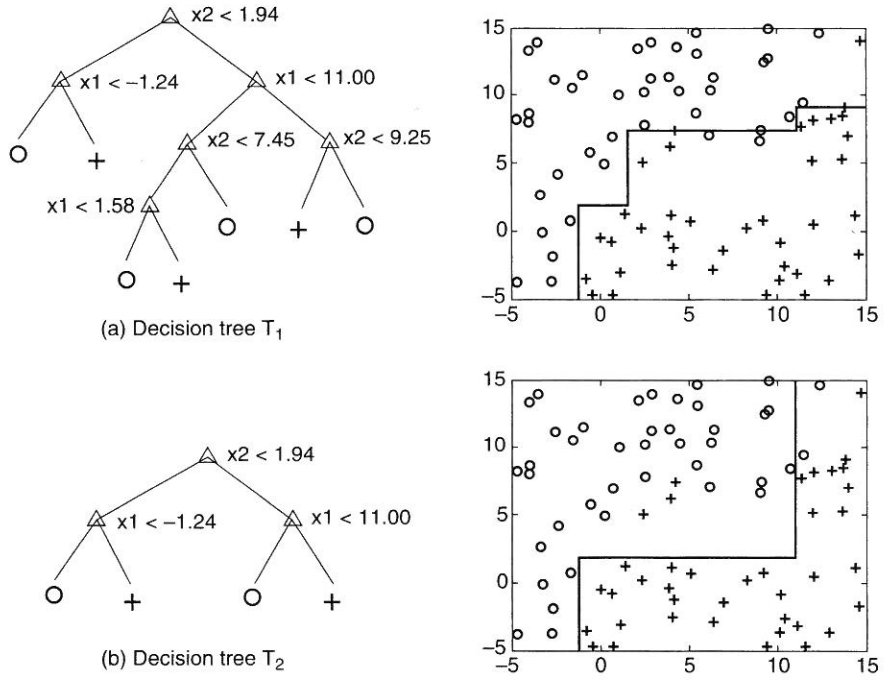


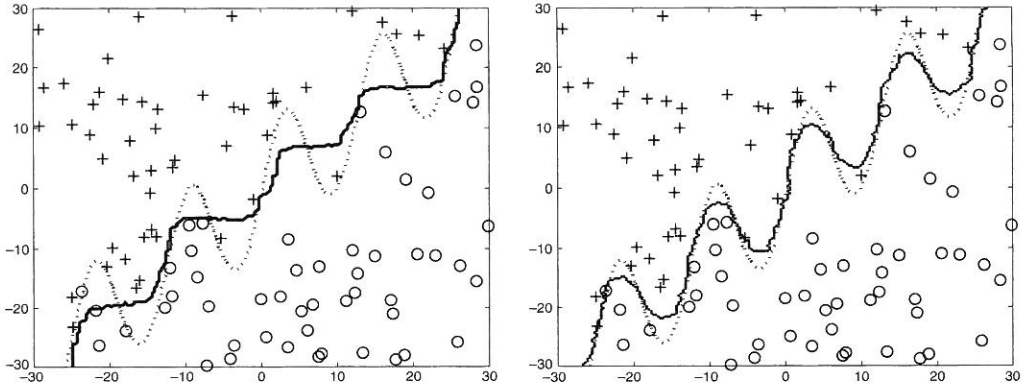
Figure 5.33. Two decision trees with different complexities induced from the same training data.

the observed difference for a decision tree classifier. This result suggests that the bias of a 1-nearest neighbor classifier is lower than the bias of a decision tree classifier.

On the other hand, the 1-nearest neighbor classifier is more sensitive to the composition of its training examples. If we examine the models induced from different training sets, there is more variability in the decision boundary of a 1-nearest neighbor classifier than a decision tree classifier. Therefore, the decision boundary of a decision tree classifier has a lower variance than the 1-nearest neighbor classifier.

5.6.4 Bagging

Bagging, which is also known as bootstrap aggregating, is a technique that repeatedly samples (with replacement) from a data set according to a uniform probability distribution. Each bootstrap sample has the same size as the original data. Because the sampling is done with replacement, some instances may appear several times in the same training set, while others may be omitted from the training set. On average, a bootstrap sample D_i contains approxi-



(a) Decision boundary for decision tree.

(b) Decision boundary for 1-nearest neighbor.

Figure 5.34. Bias of decision tree and 1-nearest neighbor classifiers.

Algorithm 5.6 Bagging algorithm.

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size N, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \underset{y}{\operatorname{argmax}} \sum_i \delta\big(C_i(x) = y\big)$.
$\{\delta(\cdot) = 1$ if its argument is true and 0 otherwise$\}$.

mately 63% of the original training data because each sample has a probability $1 - (1 - 1/N)^N$ of being selected in each D_i. If N is sufficiently large, this probability converges to $1 - 1/e \simeq 0.632$. The basic procedure for bagging is summarized in Algorithm 5.6. After training the k classifiers, a test instance is assigned to the class that receives the highest number of votes.

To illustrate how bagging works, consider the data set shown in Table 5.4. Let x denote a one-dimensional attribute and y denote the class label. Suppose we apply a classifier that induces only one-level binary decision trees, with a test condition $x \leq k$, where k is a split point chosen to minimize the entropy of the leaf nodes. Such a tree is also known as a **decision stump**.

Without bagging, the best decision stump we can produce splits the records at either $x \leq 0.35$ or $x \leq 0.75$. Either way, the accuracy of the tree is at

Table 5.4. Example of data set used to construct an ensemble of bagging classifiers.

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	−1	−1	−1	−1	1	1	1

most 70%. Suppose we apply the bagging procedure on the data set using ten bootstrap samples. The examples chosen for training in each bagging round are shown in Figure 5.35. On the right-hand side of each table, we also illustrate the decision boundary produced by the classifier.

We classify the entire data set given in Table 5.4 by taking a majority vote among the predictions made by each base classifier. The results of the predictions are shown in Figure 5.36. Since the class labels are either -1 or $+1$, taking the majority vote is equivalent to summing up the predicted values of y and examining the sign of the resulting sum (refer to the second to last row in Figure 5.36). Notice that the ensemble classifier perfectly classifies all ten examples in the original data.

The preceding example illustrates another advantage of using ensemble methods in terms of enhancing the representation of the target function. Even though each base classifier is a decision stump, combining the classifiers can lead to a decision tree of depth 2.

Bagging improves generalization error by reducing the variance of the base classifiers. The performance of bagging depends on the stability of the base classifier. If a base classifier is unstable, bagging helps to reduce the errors associated with random fluctuations in the training data. If a base classifier is stable, i.e., robust to minor perturbations in the training set, then the error of the ensemble is primarily caused by bias in the base classifier. In this situation, bagging may not be able to improve the performance of the base classifiers significantly. It may even degrade the classifier's performance because the effective size of each training set is about 37% smaller than the original data.

Finally, since every sample has an equal probability of being selected, bagging does not focus on any particular instance of the training data. It is therefore less susceptible to model overfitting when applied to noisy data.

5.6.5 Boosting

Boosting is an iterative procedure used to adaptively change the distribution of training examples so that the base classifiers will focus on examples that are hard to classify. Unlike bagging, boosting assigns a weight to each training

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

example and may adaptively change the weight at the end of each boosting round. The weights assigned to the training examples can be used in the following ways:

1. They can be used as a sampling distribution to draw a set of bootstrap samples from the original data.

2. They can be used by the base classifier to learn a model that is biased toward higher-weight examples.

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

This section describes an algorithm that uses weights of examples to determine the sampling distribution of its training set. Initially, the examples are assigned equal weights, $1/N$, so that they are equally likely to be chosen for training. A sample is drawn according to the sampling distribution of the training examples to obtain a new training set. Next, a classifier is induced from the training set and used to classify all the examples in the original data. The weights of the training examples are updated at the end of each boosting round. Examples that are classified incorrectly will have their weights increased, while those that are classified correctly will have their weights decreased. This forces the classifier to focus on examples that are difficult to classify in subsequent iterations.

The following table shows the examples chosen during each boosting round.

Boosting (Round 1):	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2):	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3):	4	4	8	10	4	5	4	6	3	4

Initially, all the examples are assigned the same weights. However, some examples may be chosen more than once, e.g., examples 3 and 7, because the sampling is done with replacement. A classifier built from the data is then used to classify all the examples. Suppose example 4 is difficult to classify. The weight for this example will be increased in future iterations as it gets misclassified repeatedly. Meanwhile, examples that were not chosen in the pre-

vious round, e.g., examples 1 and 5, also have a better chance of being selected in the next round since their predictions in the previous round were likely to be wrong. As the boosting rounds proceed, examples that are the hardest to classify tend to become even more prevalent. The final ensemble is obtained by aggregating the base classifiers obtained from each boosting round.

Over the years, several implementations of the boosting algorithm have been developed. These algorithms differ in terms of (1) how the weights of the training examples are updated at the end of each boosting round, and (2) how the predictions made by each classifier are combined. An implementation called AdaBoost is explored in the next section.

AdaBoost

Let $\{(\mathbf{x}_j, y_j) \mid j = 1, 2, \ldots, N\}$ denote a set of N training examples. In the AdaBoost algorithm, the importance of a base classifier C_i depends on its error rate, which is defined as

$$\epsilon_i = \frac{1}{N}\left[\sum_{j=1}^{N} w_j \, I\Big(C_i(\mathbf{x}_j) \neq y_j\Big)\right], \tag{5.68}$$

where $I(p) = 1$ if the predicate p is true, and 0 otherwise. The importance of a classifier C_i is given by the following parameter,

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1 - \epsilon_i}{\epsilon_i}\right).$$

Note that α_i has a large positive value if the error rate is close to 0 and a large negative value if the error rate is close to 1, as shown in Figure 5.37.

The α_i parameter is also used to update the weight of the training examples. To illustrate, let $w_i^{(j)}$ denote the weight assigned to example $(\mathbf{x}_i, y_i)$ during the j^{th} boosting round. The weight update mechanism for AdaBoost is given by the equation:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(\mathbf{x_i}) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(\mathbf{x_i}) \neq y_i \end{cases}, \tag{5.69}$$

where Z_j is the normalization factor used to ensure that $\sum_i w_i^{(j+1)} = 1$. The weight update formula given in Equation 5.69 increases the weights of incorrectly classified examples and decreases the weights of those classified correctly.

Instead of u
classifier C_j is
penalize model
boosting round
rate higher tha
values, $w_i = 1/$
algorithm is su

Let us exar
in Table 5.4. 1
boosting round
The weights fo
using Equation

Without bo
With AdaBoos
The final predi
average of the
last row of Fig
examples in th

An importa
the ensemble is

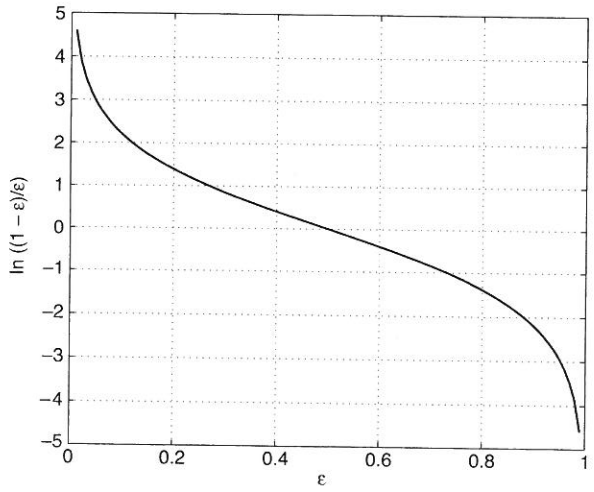


Figure 5.37. Plot of α as a function of training error ϵ.

Instead of using a majority voting scheme, the prediction made by each classifier C_j is weighted according to α_j. This approach allows AdaBoost to penalize models that have poor accuracy, e.g., those generated at the earlier boosting rounds. In addition, if any intermediate rounds produce an error rate higher than 50%, the weights are reverted back to their original uniform values, $w_i = 1/N$, and the resampling procedure is repeated. The AdaBoost algorithm is summarized in Algorithm 5.7.

Let us examine how the boosting approach works on the data set shown in Table 5.4. Initially, all the examples have identical weights. After three boosting rounds, the examples chosen for training are shown in Figure 5.38(a). The weights for each example are updated at the end of each boosting round using Equation 5.69.

Without boosting, the accuracy of the decision stump is, at best, 70%. With AdaBoost, the results of the predictions are given in Figure 5.39(b). The final prediction of the ensemble classifier is obtained by taking a weighted average of the predictions made by each base classifier, which is shown in the last row of Figure 5.39(b). Notice that AdaBoost perfectly classifies all the examples in the training data.

An important analytical result of boosting shows that the training error of the ensemble is bounded by the following expression:

$$e_{\text{ensemble}} \leq \prod_i \left[\sqrt{\epsilon_i(1-\epsilon_i)} \right], \tag{5.70}$$

Algorithm 5.7 AdaBoost algorithm.

1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Initialize the weights for all N examples.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all examples in the original training set, D.
7: $\epsilon_i = \frac{1}{N}\left[\sum_j w_j \, \delta(C_i(x_j) \neq y_j)\right]$ {Calculate the weighted error.}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Reset the weights for all N examples.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each example according to Equation 5.69.
14: **end for**
15: $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y))$.

where ϵ_i is the error rate of each base classifier i. If the error rate of the base classifier is less than 50%, we can write $\epsilon_i = 0.5 - \gamma_i$, where γ_i measures how much better the classifier is than random guessing. The bound on the training error of the ensemble becomes

$$e_{\text{ensemble}} \leq \prod_i \sqrt{1 - 4\gamma_i^2} \leq \exp\left(-2 \sum_i \gamma_i^2\right). \tag{5.71}$$

If $\gamma_i < \gamma*$ for all i's, then the training error of the ensemble decreases exponentially, which leads to the fast convergence of the algorithm. Nevertheless, because of its tendency to focus on training examples that are wrongly classified, the boosting technique can be quite susceptible to overfitting.

5.6.6 Random Forests

Random forest is a class of ensemble methods specifically designed for decision tree classifiers. It combines the predictions made by multiple decision trees, where each tree is generated based on the values of an independent set of random vectors, as shown in Figure 5.40. The random vectors are generated from a fixed probability distribution, unlike the adaptive approach used in AdaBoost, where the probability distribution is varied to focus on examples that are hard to classify. Bagging using decision trees is a special case of random forests, where randomness is injected into the model-building process

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

Figure 5.38. Example of boosting.

by randomly choosing N samples, with replacement, from the original training set. Bagging also uses the same uniform probability distribution to generate its bootstrapped samples throughout the entire model-building process.

It was theoretically proven that the upper bound for generalization error of random forests converges to the following expression, when the number of trees is sufficiently large.

$$\text{Generalization error} \leq \frac{\overline{\rho}(1-s^2)}{s^2}, \tag{5.72}$$

where $\overline{\rho}$ is the average correlation among the trees and s is a quantity that measures the "strength" of the tree classifiers. The strength of a set of classifiers refers to the average performance of the classifiers, where performance is measured probabilistically in terms of the classifier's margin:

$$\text{margin}, M(\mathbf{X}, Y) = P(\hat{Y}_\theta = Y) - \max_{Z \neq Y} P(\hat{Y}_\theta = Z), \tag{5.73}$$

where $\hat{Y}_\theta$ is the predicted class of $\mathbf{X}$ according to a classifier built from some random vector θ. The higher the margin is, the more likely it is that the

Round	Split Point	Left Class	Right Class	α
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

(a)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

(b)

Figure 5.39. Example of combining classifiers constructed using the AdaBoost approach.

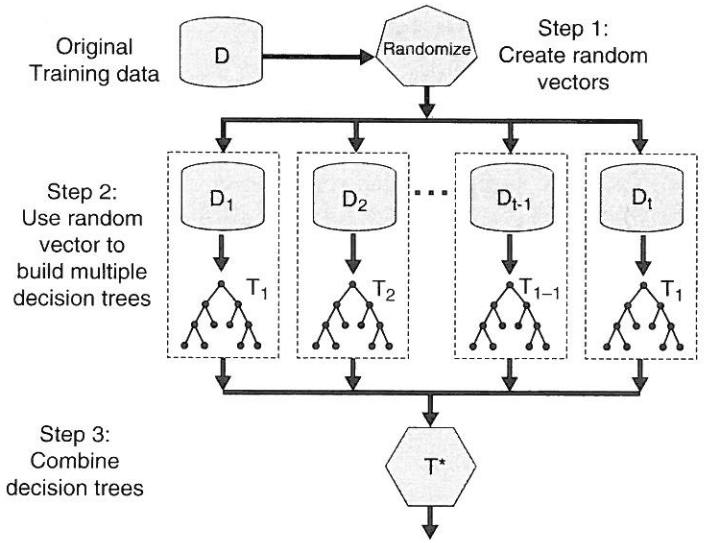


Figure 5.40. Random forests.

classifier correctly predicts a given example $\mathbf{X}$. Equation 5.72 is quite intuitive; as the trees become more correlated or the strength of the ensemble decreases, the generalization error bound tends to increase. Randomization helps to reduce the correlation among decision trees so that the generalization error of the ensemble can be improved.

Each decision tree uses a random vector that is generated from some fixed probability distribution. A random vector can be incorporated into the tree-growing process in many ways. The first approach is to randomly select F input features to split at each node of the decision tree. As a result, instead of examining all the available features, the decision to split a node is determined from these selected F features. The tree is then grown to its entirety without any pruning. This may help reduce the bias present in the resulting tree. Once the trees have been constructed, the predictions are combined using a majority voting scheme. This approach is known as Forest-RI, where RI refers to random input selection. To increase randomness, bagging can also be used to generate bootstrap samples for Forest-RI. The strength and correlation of random forests may depend on the size of F. If F is sufficiently small, then the trees tend to become less correlated. On the other hand, the strength of the tree classifier tends to improve with a larger number of features, F. As a tradeoff, the number of features is commonly chosen to be $F = \log_2 d + 1$, where d is the number of input features. Since only a subset of the features needs to be examined at each node, this approach helps to significantly reduce the runtime of the algorithm.

If the number of original features d is too small, then it is difficult to choose an independent set of random features for building the decision trees. One way to increase the feature space is to create linear combinations of the input features. Specifically, at each node, a new feature is generated by randomly selecting L of the input features. The input features are linearly combined using coefficients generated from a uniform distribution in the range of $[-1, 1]$. At each node, F of such randomly combined new features are generated, and the best of them is subsequently selected to split the node. This approach is known as Forest-RC.

A third approach for generating the random trees is to randomly select one of the F best splits at each node of the decision tree. This approach may potentially generate trees that are more correlated than Forest-RI and Forest-RC, unless F is sufficiently large. It also does not have the runtime savings of Forest-RI and Forest-RC because the algorithm must examine all the splitting features at each node of the decision tree.

It has been shown empirically that the classification accuracies of random forests are quite comparable to the AdaBoost algorithm. It is also more robust to noise and runs much faster than the AdaBoost algorithm. The classification accuracies of various ensemble algorithms are compared in the next section.

Table 5.5. Comparing the accuracy of a decision tree classifier against three ensemble methods.

Data Set	Number of (Attributes, Classes, Records)	Decision Tree (%)	Bagging (%)	Boosting (%)	RF (%)
Anneal	(39, 6, 898)	92.09	94.43	95.43	95.43
Australia	(15, 2, 690)	85.51	87.10	85.22	85.80
Auto	(26, 7, 205)	81.95	85.37	85.37	84.39
Breast	(11, 2, 699)	95.14	96.42	97.28	96.14
Cleve	(14, 2, 303)	76.24	81.52	82.18	82.18
Credit	(16, 2, 690)	85.8	86.23	86.09	85.8
Diabetes	(9, 2, 768)	72.40	76.30	73.18	75.13
German	(21, 2, 1000)	70.90	73.40	73.00	74.5
Glass	(10, 7, 214)	67.29	76.17	77.57	78.04
Heart	(14, 2, 270)	80.00	81.48	80.74	83.33
Hepatitis	(20, 2, 155)	81.94	81.29	83.87	83.23
Horse	(23, 2, 368)	85.33	85.87	81.25	85.33
Ionosphere	(35, 2, 351)	89.17	92.02	93.73	93.45
Iris	(5, 3, 150)	94.67	94.67	94.00	93.33
Labor	(17, 2, 57)	78.95	84.21	89.47	84.21
Led7	(8, 10, 3200)	73.34	73.66	73.34	73.06
Lymphography	(19, 4, 148)	77.03	79.05	85.14	82.43
Pima	(9, 2, 768)	74.35	76.69	73.44	77.60
Sonar	(61, 2, 208)	78.85	78.85	84.62	85.58
Tic-tac-toe	(10, 2, 958)	83.72	93.84	98.54	95.82
Vehicle	(19, 4, 846)	71.04	74.11	78.25	74.94
Waveform	(22, 3, 5000)	76.44	83.30	83.90	84.04
Wine	(14, 3, 178)	94.38	96.07	97.75	97.75
Zoo	(17, 7, 101)	93.07	93.07	95.05	97.03

5.6.7 Empirical Comparison among Ensemble Methods

Table 5.5 shows the empirical results obtained when comparing the performance of a decision tree classifier against bagging, boosting, and random forest. The base classifiers used in each ensemble method consist of fifty decision trees. The classification accuracies reported in this table are obtained from ten-fold cross-validation. Notice that the ensemble classifiers generally outperform a single decision tree classifier on many of the data sets.

5.7 Class Imbalance Problem

Data sets with imbalanced class distributions are quite common in many real applications. For example, an automated inspection system that monitors products that come off a manufacturing assembly line may find that the num-