```
1 repeat
2     Create a pruned classification tree
3     Determine the path through the tree with the largest coverage
4     Add this path as a rule to the rule set
5     Remove the training set samples covered by the rule
6 until all training set samples are covered by a rule
```

Algorithm 14.1: The PART algorithm for constructing rule-based models (Frank and Witten 1998)

PART

C4.5Rules follows the philosophy that the initial set of candidate rules are developed simultaneously then post-processed into an improved model. Alternatively, rules can be created incrementally. In this way, a new rule can adapt to the previous set of rules and may more effectively capture important trends in the data.

Frank and Witten (1998) describe another rule model called PART shown in Algorithm 14.1. Here, a pruned C4.5 tree is created from the data and the path through the tree that covers the most samples is retained as a rule. The samples covered by the rule are discarded from the data set and the process is repeated until all samples are covered by at least one rule. Although the model uses trees to create the rules, each rule is created separately and has more potential freedom to adapt to the data.

The PART model for the grant data slightly favored the grouped category model. For this model, the results do not show an improvement above and beyond the previous models: the estimated sensitivity was 77.9 %, the specificity was 80.2 %, and the area under the ROC curve (not shown) was 0.809. The model contained 360 rules. Of these, 181 classify grants as successful while the other 179 classify grants as unsuccessful. Here, the five most prolific predictors were sponsor code (332 rules), contract value band (30 rules), the number of unsuccessful grants by chief investigators (27 rules), the number of successful grants by chief investigators (26 rules), and the number of chief investigators (23 rules).

14.3 Bagged Trees

Bagging for classification is a simple modification to bagging for regression (Sect. 8.4). Specifically, the regression tree in Algorithm 8.1 is replaced with an unpruned classification tree for modeling C classes. Like the regression

Table 14.1: The 2008 holdout set confusion matrix for the random forest model

	Observed class	
	Successful	Unsuccessful
Successful	491	144
Unsuccessful	79	843

This model had an overall accuracy of 85.7 %, a sensitivity of 86.1 %, and a specificity of 85.4 %

setting, each model in the ensemble is used to predict the class of the new sample. Since each model has equal weight in the ensemble, each model can be thought of as casting a vote for the class it thinks the new sample belongs to. The total number of votes within each class are then divided by the total number of models in the ensemble (M) to produce a predicted probability vector for the sample. The new sample is then classified into the group that has the most votes, and therefore the highest probability.

For the grant data, bagging models were built using both strategies for categorical predictors. As discussed in the regression trees chapter, bagging performance often plateaus with about 50 trees, so 50 was selected as the number of trees for each of these models. Figure 14.7 illustrates the bagging ensemble performance using either independent or grouped categories. Both of these ROC curves are smoother than curves produced with classification trees or J48, which is an indication of bagging's ability to reduce variance via the ensemble. Additionally, both bagging models have better AUCs (0.92 for both) than either of the previous models. For these data, there seems to be no obvious difference in performance for bagging when using either independent or grouped categories; the ROC curves, sensitivities, and specificities are all nearly identical. The holdout set performance in Fig. 14.7 shows an improvement over the J48 results (Fig. 14.6).

Similar to the regression setting, variable importance measures can be calculated by aggregating variable importance values from the individual trees in the ensemble. Variable importance of the top 16 predictors for both the independent and grouped category bagged models set are presented in Fig. 14.15, and a comparison of these results is left to the reader in Exercise 14.1.

14.4 Random Forests

Random forests for classification requires a simple tweak to the random forest regression algorithm (Algorithm 8.2): a classification tree is used in place of

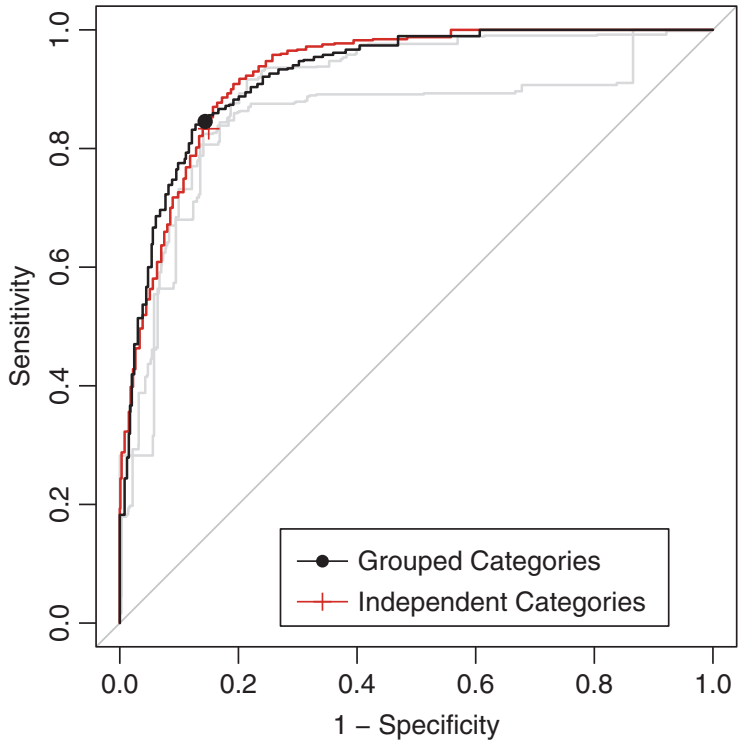

Fig. 14.7: The ROC curves for the bagged classification tree model. The area under the curves for both models was 0.92. The sensitivities and specificities were 82.98 and 85.71, respectively

a regression tree. As with bagging, each tree in the forest casts a vote for the classification of a new sample, and the proportion of votes in each class across the ensemble is the predicted probability vector.

While the type of tree changes in the algorithm, the tuning parameter of number of randomly selected predictors to choose from at each split is the same (denoted as m_{try}). As in regression, the idea behind randomly sampling predictors during training is to de-correlate the trees in the forest. For classification problems, Breiman (2001) recommends setting m_{try} to the square root of the number of predictors. To tune m_{try}, we recommend starting with five values that are somewhat evenly spaced across the range from 2 to P, where P is the number of predictors. We likewise recommend starting with an ensemble of 1,000 trees and increasing that number if performance is not yet close to a plateau.

For the most part, random forest for classification has very similar properties to the regression analog discussed previously, including:

- The model is relatively insensitive to values of m_{try}.
- As with most trees, the data pre-processing requirements are minimal.
- Out-of-bag measures of performance can be calculated, including accuracy, sensitivity, specificity, and confusion matrices.

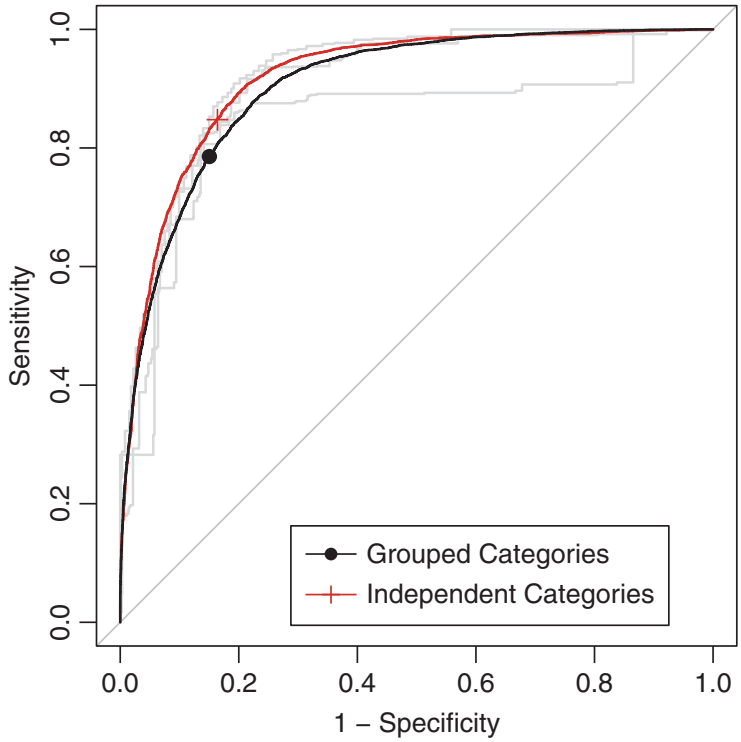

Fig. 14.8: The ROC curves for the random forest model. The area under the curve for independent categories was 0.92 and for the grouped category model the AUC was 0.9

One difference is the ability to weight classes differentially. This aspect of the model is discussed more in Chap. 16.

Random forest models were built on both independent and grouped category models. The tuning parameter, m_{try}, was evaluated at values ranging from 5 to 1,000. For independent categories, the optimal tuned value of m_{try} was 100, and for grouped categories the value was also 250. Figure 14.8 presents the results, and in this case the independent categories have a slightly higher AUC (0.92) than the grouped category approach (0.9). The binary predictor model also has better sensitivity (86.1 % vs. 84.7 %) but slightly worse specificity (85.4 % vs. 87.2 %).

For single trees, variable importance can be determined by aggregating the improvement in the optimization objective for each predictor. For random forests, the improvement criteria (default is typically the Gini index) is aggregated across the ensemble to generate an overall variable importance measure. Alternatively, predictors' impact on the ensemble can be calculated using a permutation approach (Breiman 2000) as discussed in Sect. 8.5. Variable importance values based on aggregated improvement have been computed for the grant data for both types of predictors and the most important

predictors are presented in Fig. 14.15. The interpretation is left to the reader in Exercise 14.1.

Conditional inference trees can also be used as the base learner for random forests. But current implementations of the methodology are computationally burdensome for problems that are the relative size of the grant data. A comparison of the performance of random forests using CART trees and conditional inference trees is explored in Exercise 14.3.

14.5 Boosting

Although we have already discussed boosting in the regression setting, the method was originally developed for classification problems (Valiant 1984; Kearns and Valiant 1989), in which many weak classifiers (e.g., a classifier that predicts marginally better than random) were combined into a strong classifier. There are many species of boosting algorithms, and here we discuss the major ones.

AdaBoost

In the early 1990s several boosting algorithms appeared (Schapire 1990; Freund 1995) to implement the original theory. Freund and Schapire (1996) finally provided the first practical implementation of boosting theory in their famous AdaBoost algorithm; an intuitive version is provided in Algorithm 14.2.

To summarize the algorithm, AdaBoost generates a sequence of weak classifiers, where at each iteration the algorithm finds the best classifier based on the current sample weights. Samples that are incorrectly classified in the kth iteration receive more weight in the $(k+1)$st iteration, while samples that are correctly classified receive less weight in the subsequent iteration. This means that samples that are difficult to classify receive increasingly larger weights until the algorithm identifies a model that correctly classifies these samples. Therefore, each iteration of the algorithm is required to learn a different aspect of the data, focusing on regions that contain difficult-to-classify samples. At each iteration, a *stage weight* is computed based on the error rate at that iteration. The nature of the stage weight described in Algorithm 14.2 implies that more accurate models have higher positive values and less accurate models have lower negative values.[5] The overall sequence of weighted classifiers is then combined into an ensemble and has a strong potential to classify better than any of the individual classifiers.

[5] Because a weak classifier is used, the stage values are often close to zero.

1 Let one class be represented with a value of +1 and the other with a value of -1
2 Let each sample have the same starting weight $(1/n)$
3 **for** $k = 1$ *to* K **do**
4 Fit a weak classifier using the weighted samples and compute the kth model's misclassification error (err_k)
5 Compute the kth stage value as $\ln((1-err_k)/err_k)$.
6 Update the sample weights giving more weight to incorrectly predicted samples and less weight to correctly predicted samples
7 **end**
8 Compute the boosted classifier's prediction for each sample by multiplying the kth stage value by the kth model prediction and adding these quantities across k. If this sum is positive, then classify the sample in the +1 class, otherwise the -1 class.

Algorithm 14.2: AdaBoost algorithm for two-class problems

Boosting can be applied to any classification technique, but classification trees are a popular method for boosting since these can be made into weak learners by restricting the tree depth to create trees with few splits (also known as stumps). Breiman (1998) gives an explanation for why classification trees work particularly well for boosting. Since classification trees are a low bias/high variance technique, the ensemble of trees helps to drive down variance, producing a result that has low bias and low variance. Working through the lens of the AdaBoost algorithm, Johnson and Rayens (2007) showed that low variance methods cannot be greatly improved through boosting. Therefore, boosting methods such as LDA or KNN will not show as much improvement as boosting methods such as neural networks (Freund and Schapire 1996) or naïve Bayes (Bauer and Kohavi 1999).

Stochastic Gradient Boosting

As mentioned in Sect. 8.6, Friedman et al. (2000) worked to provide statistical insight of the AdaBoost algorithm. For the classification problem, they showed that it could be interpreted as a forward stagewise additive model that minimizes an exponential loss function. This framework led to algorithmic generalizations such as Real AdaBoost, Gentle AdaBoost, and LogitBoost. Subsequently, these generalizations were put into a unifying framework called gradient boosting machines which was previously discussed in the regression trees chapter.