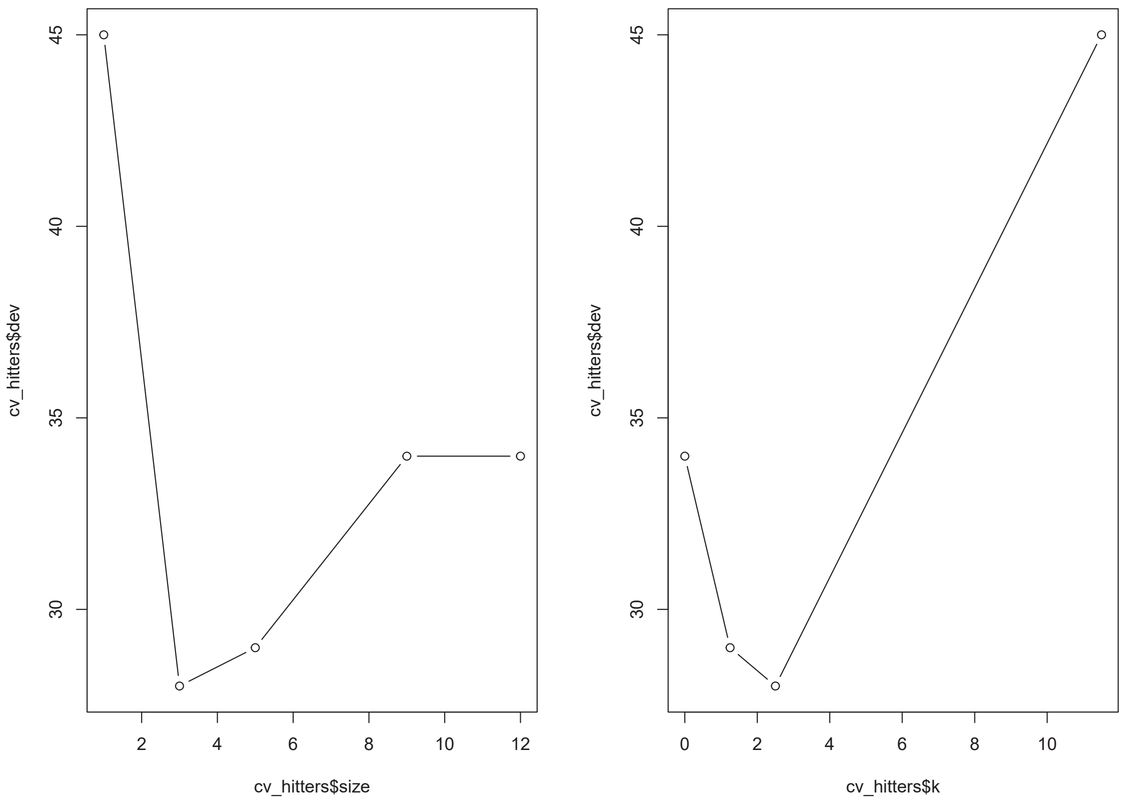
```
> set.seed(1)
> cv_hitters <- cv.tree(tree_hitters, FUN = prune.misclass)
> names(cv_hitters)
[1] "size"   "dev"    "k"      "method"
> cv_hitters
$size
[1] 12  9  5  3  1

$dev
[1] 34 34 29 28 45

$k
[1]  -Inf  0.00  1.25  2.50 11.50

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

$\xleftarrow{}$ cross-validation err.

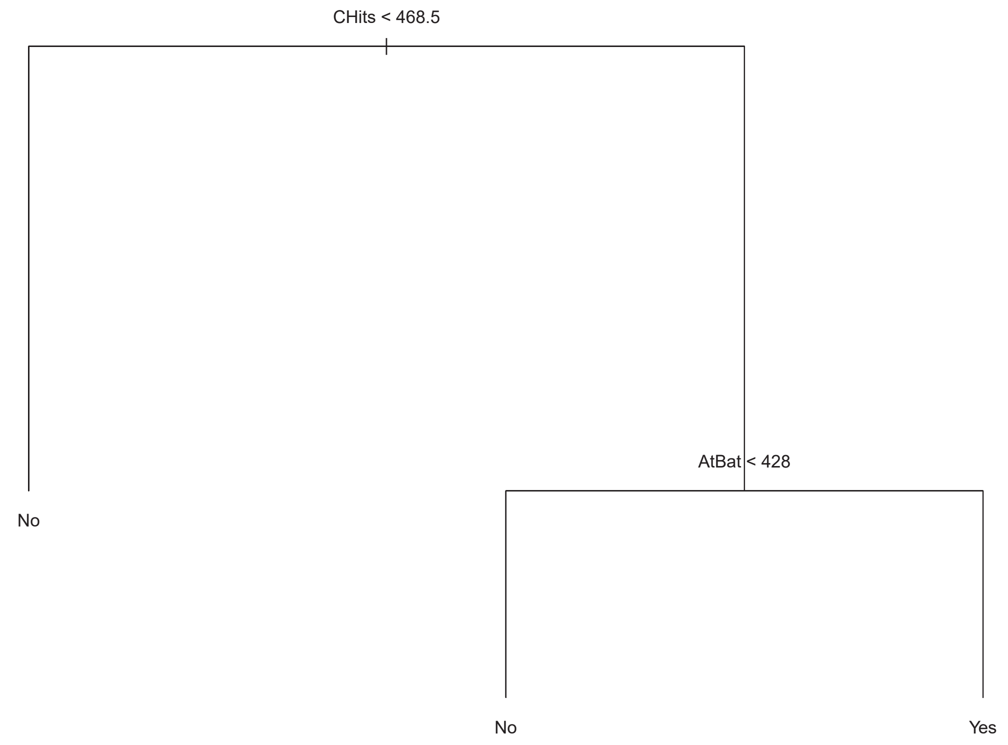
dev corresponds to the number of cross-validation errors. We plot the error rate as a function of both size and k.

```
> par(mfrow = c(1, 2))
> plot(cv_hitters$size, cv_hitters$dev, type = "b")
> plot(cv_hitters$k, cv_hitters$dev, type = "b")
```



We now apply the prune.misclass() function in order to prune the tree to obtain the three-node tree.

```
> prune_hitters <- prune.misclass(tree_hitters, best = 3)
> plot(prune_hitters)
> text(prune_hitters, pretty = 0)
```



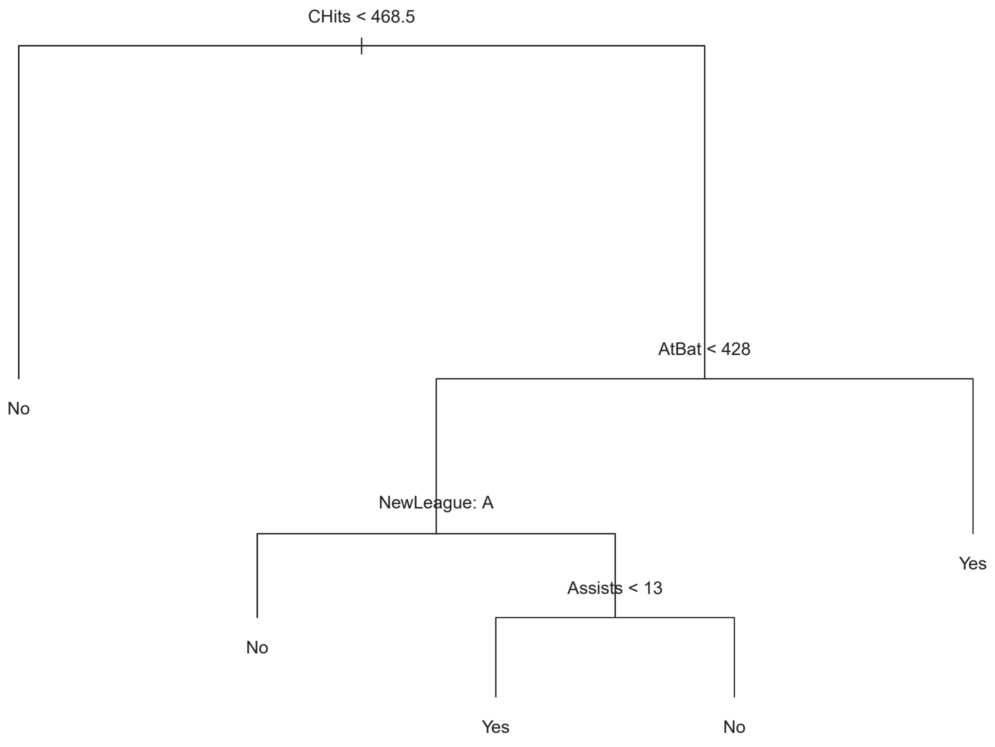
Once again, we apply the predict() function to see how well this pruned tree performs on the test data set.

```
> tree_pred <- predict(prune_hitters, Hitters_test, type = "class")
> table(tree_pred, High_test)
         High_test
tree_pred No Yes
      No  78  11
      Yes 10  32
> mean(tree_pred!=High_test)
[1] 0.1603053
```

Now the test error is 0.16, so not only has the pruning process produced a more interpretable tree, but it has also slightly improved the classification accuracy.

If we increase the value of best, we obtain a larger pruned tree with lower classification accuracy:

```
> prune_hitters <- prune.misclass(tree_hitters, best = 5)
> plot(prune_hitters)
> text(prune_hitters, pretty = 0)
> tree_pred <- predict(prune_hitters, Hitters_test, type = "class")
> table(tree_pred, High_test)
         High_test
tree_pred No Yes
      No  74  10
      Yes 14  33
> mean(tree_pred!=High_test)
[1] 0.1832061
```

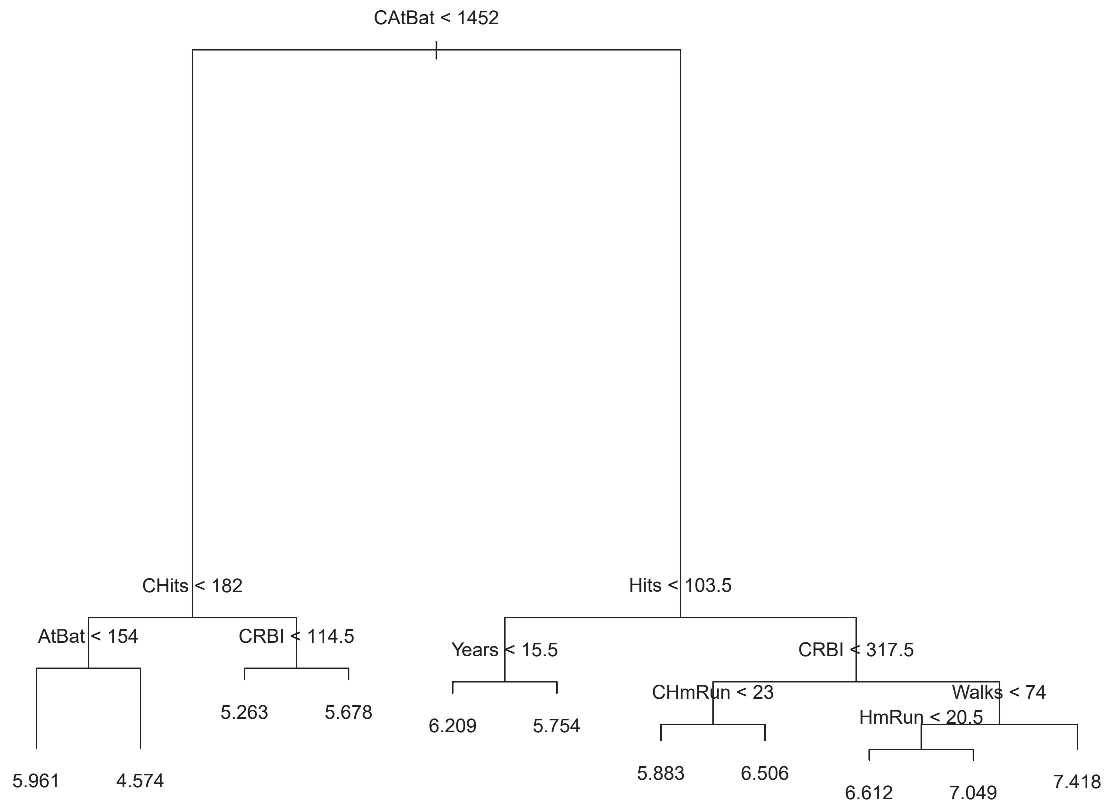


Now we fit a regression tree to the Hitters data set. First, we fit the tree to the training data.

```
> tree_hitters_reg <- tree(Log_Salary ~ . - Salary, Hitters, subset = train)
> summary(tree_hitters_reg)

Regression tree:
tree(formula = Log_Salary ~ . - Salary, data = Hitters, subset = train)
Variables actually used in tree construction:
[1] "CAtBat" "CHits"  "AtBat"  "CRBI"   "Hits"   "Years"  "CHmRun"
[8] "Walks"  "HmRun"
Number of terminal nodes:  11
Residual mean deviance:  0.1421 = 17.19 / 121
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.38300 -0.19250  0.01232  0.00000  0.15440  1.70200
```

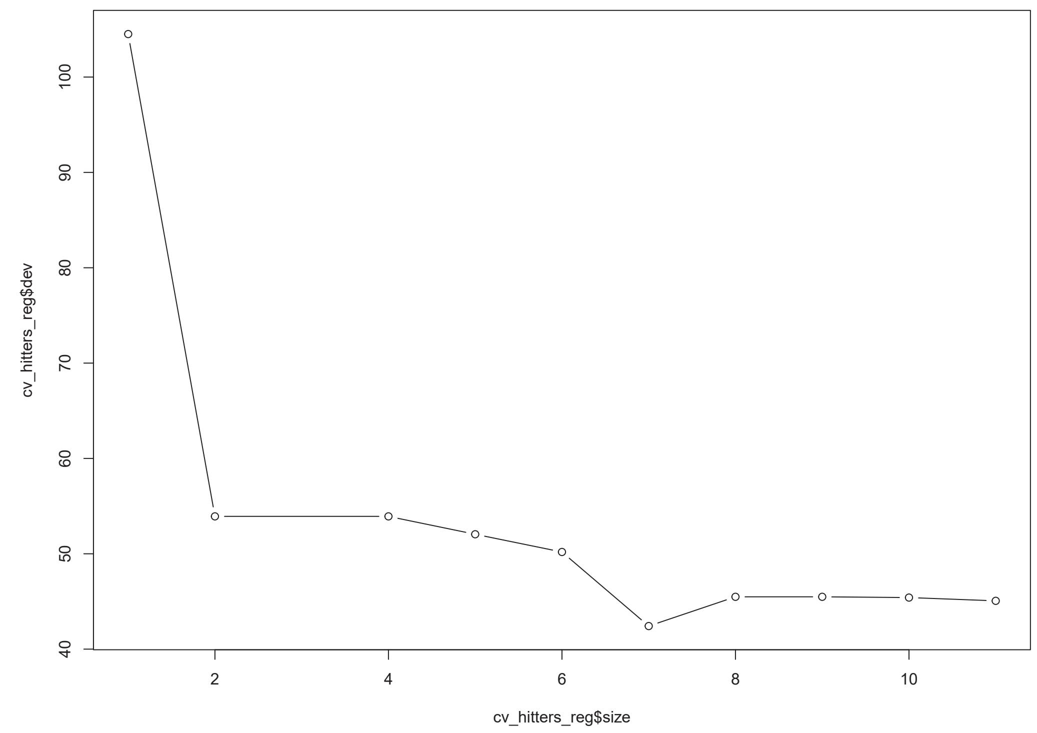
Notice that the output of summary() indicates that nine variables have been used in constructing the tree. In the context of a regression tree, the deviance is simply the sum of squared errors for the tree. We now plot the tree.

```
> plot(tree_hitters_reg)
> text(tree_hitters_reg, pretty = 0)
```



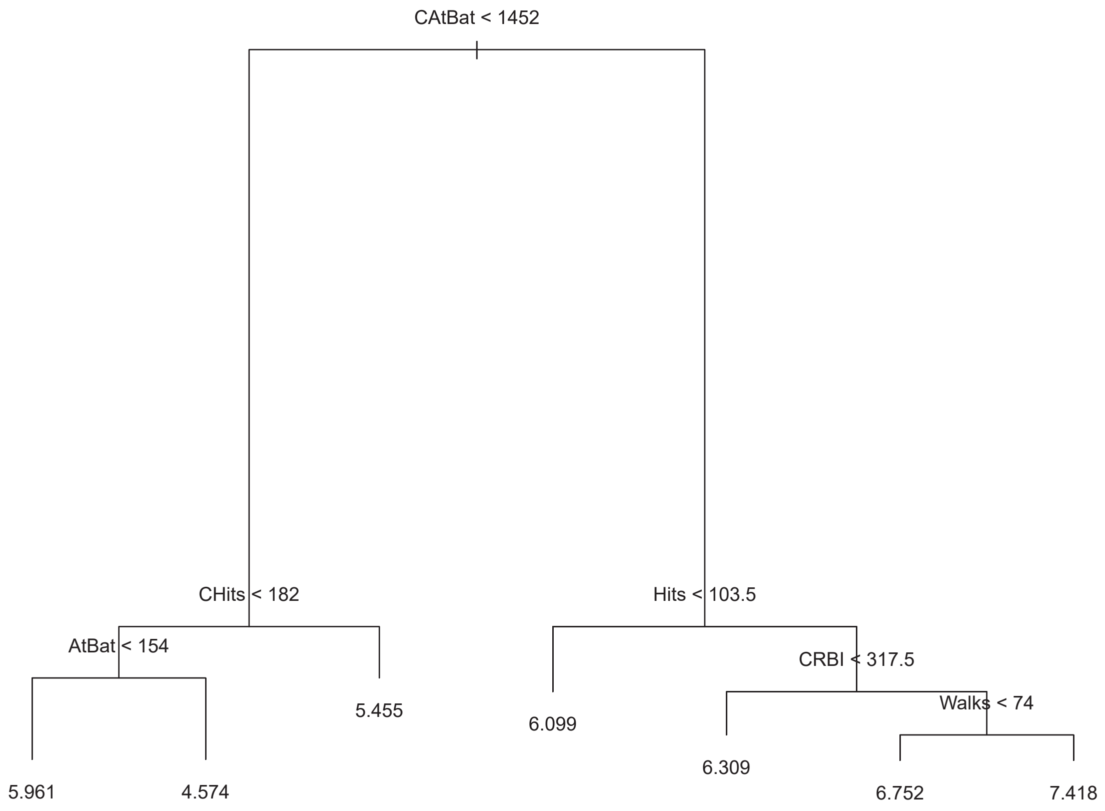
Now we use the cv.tree() function to see whether pruning the tree will improve performance.

```
> cv_hitters_reg <- cv.tree(tree_hitters_reg)
> plot(cv_hitters_reg$size, cv_hitters_reg$dev, type = "b")
```



In this case, the tree with 7 terminal nodes is selected by cross-validation. To prune the tree, we can use the prune.tree() function:

```
> prune_hitters_reg <- prune.tree(tree_hitters_reg, best = 7)
> plot(prune_hitters_reg)
> text(prune_hitters_reg, pretty = 0)
```



Now we use the pruned tree to make predictions on the test set.

```
> yhat <- predict(prune_hitters_reg, newdata = Hitters[-train, ])
> Log_Salary_test <- Log_Salary[-train]
> mean((yhat - Log_Salary_test)^2)
[1] 0.2287691
```

$\leftarrow$ the test set MSE associated with the pruned regression tree.

Bagging

- The decision trees suffer from high variance.
- This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different.
- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method.
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

In other words, averaging a set of observations reduces variance.

- Hence a natural way to reduce the variance and increase the test set accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions.
- Of course, this is not practical because we generally do not have access to multiple training sets.
- Instead, we can bootstrap, by taking repeated samples from the (single) training data set. In this approach we generate B different bootstrapped training data sets.
- We then train our method on the bth bootstrapped training set in order to get $\hat{f}^{*b}(x)$, and finally average all the predictions, to obtain

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

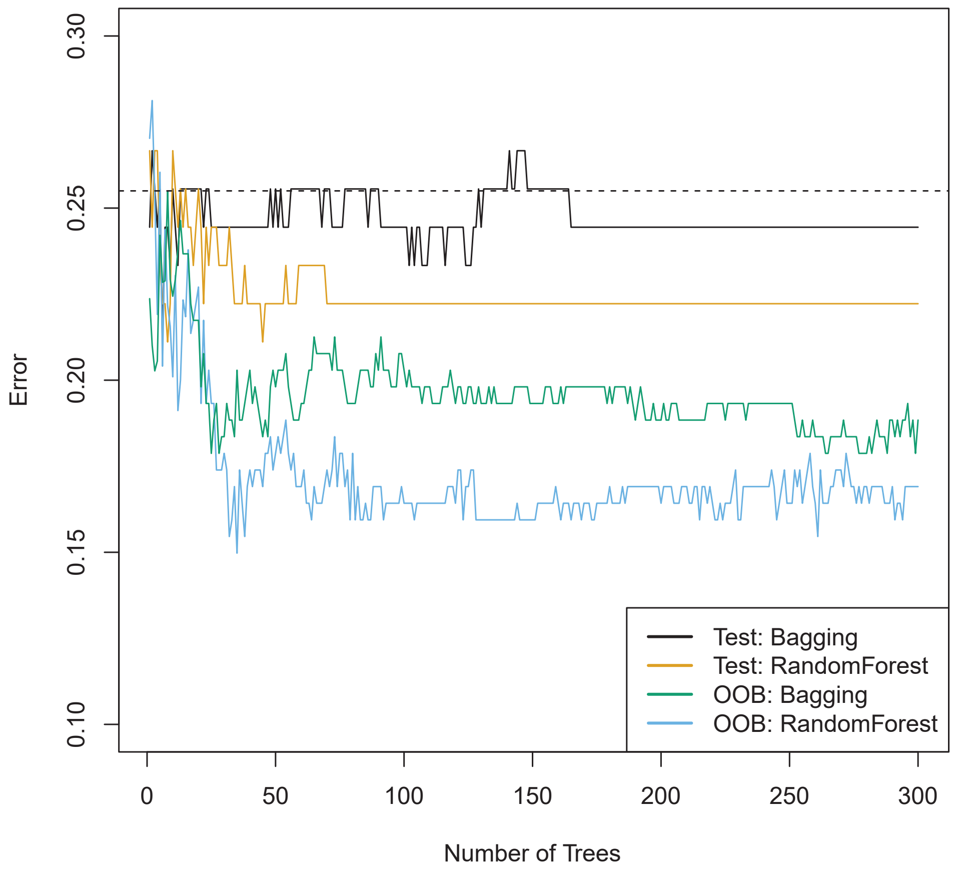
This is called *bagging*.

- To apply bagging to regression trees, we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions.
- These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance.
- To predict a qualitative outcome Y, for a given test observation, we can record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.
- The number of trees B is not a critical parameter with bagging; using a very large value of B will not lead to overfitting. In practice we use a value of B sufficiently large that the error has settled down.

<u>Out-of-Bag Error Estimation</u>

- There is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation or the validation set approach.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations.
- One can show that on average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation, which we average.
- An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed.

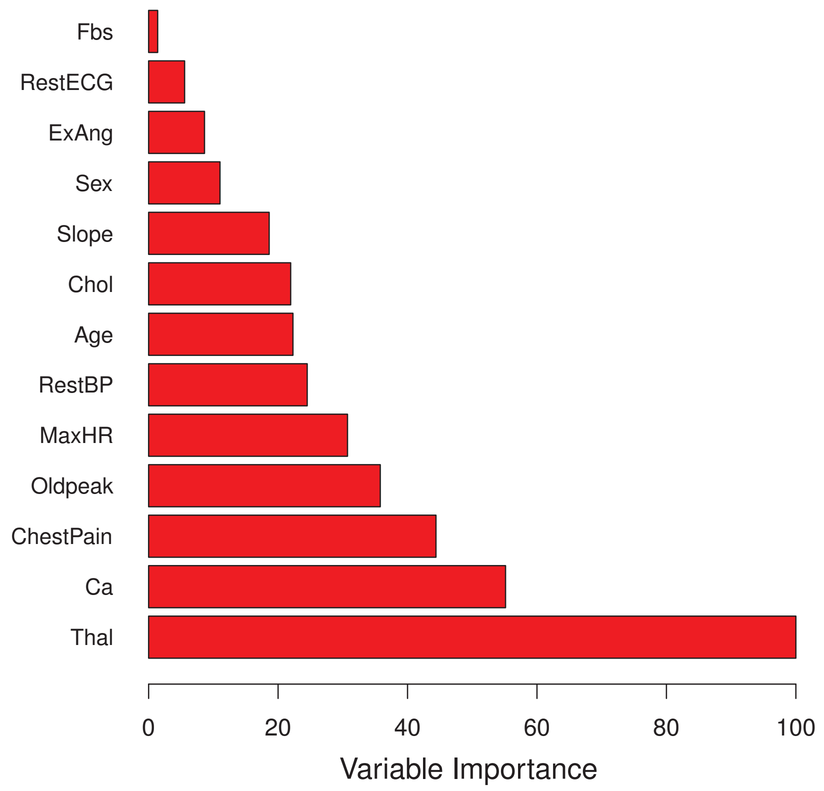
- It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.
- The OOB approach for estimating the test error is particularly convenient when performing bagging on large data sets for which cross-validation would be computationally onerous.



Bagging and random forest results for the Heart data. The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used. Random forests were applied with $m = \sqrt{p}$. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is — by chance — considerably lower.

Variable Importance Measures

- Although the collection of bagged trees is much more difficult to interpret than a single tree, one can obtain an overall summary of the importance of each predictor using the RSS (for bagging regression trees) or the Gini index (for bagging classification trees).
- In the case of bagging regression trees, we can record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, in the context of bagging classification trees, we can add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



A variable importance plot for the Heart data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.