Chapter 8

Bagging and Random Forests

As previously discussed, we will use bagging and random forests(rf) to construct more powerful prediction models.

8.1 Bagging

The bootstrap as introduced in Chapter [[ref]] is a very useful idea, where it can be used in many situations where it is very difficult to compute the standard deviation of a quantity of interest. Here, the bootstrap can be calculated to improve statistical learning of decision trees.

The decision trees in the last chapter suffer from high variance, meaning that if we split the training data into two parts at random and fit a decision tree to both halves, the results could be quite different.[1]

Bootstrap aggregation or *bagging* is a general-purpose procedure for reducing the variance of a statistical learning method. We use it here for decision trees. Recall that given 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.[2] A natural way to reduce the variance and increase the prediction accuracy of a statistical learning method is to take *many training sets* from

[1] In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets; linear regression tends to have low variance, if the ratio of n to p is moderately large.

[2] So, averaging a set of observations reduces the variance.

the population, build a separate prediction model using each training set, and average the resulting predictions. We could do the following:

1. Calculate $\hat{f}^1(x), \ldots, \hat{f}^B(x)$ using B separate training sets
2. Average these to find a single low-variance learning model given by

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

This is not practical! Why? We generally do not have access to multiple training sets. However, instead, we can bootstrap. That is, we can take repeated samples from the single training data set. Using this approach,

1. We generate B different bootstrapped training data sets.
2. Then we train our method on the bth bootstrapped training set in order to get $\hat{f}^{b*}(x)$.
3. Finally, we 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.

While bagging can improve predictions for many regression methods, it is very useful for regression trees. To apply bagging to regression trees we:

1. Construct B regression trees using B bootstrapped training sets.
2. We then average the predictions.
3. These trees are grown deep and are not pruned.
4. Each tree has a high variance with low bias. Averaging the B trees brings down the variance.
5. Bagging has been shown to give impressive improvements [[cite]] by combining hundreds or thousands of trees in a single procedures.

We have just described bagging in a regression context to predict a quantitative outcome Y. What if our outcome is qualitative? 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.

8.1.1 Out-of-Bag Error Estimation

There is a very straightforward way to estimate the test error of a bagged model, without the need to perform CV or the validation test set approach. The key to bagging is that the trees are repeatedly fit to bootstrapped subsets of observations. One can show that on average, each bagged tree make use of around two-third of the observations. The remaining one-third of the observations not used to fit a given bagged tree are called *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 approximately $B/3$ predictions for the ith observation. To obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal). This yields a single OOB prediction for the ith observation. An OOB prediction can be found in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or a classification error (for a classification problem) can be computed. Check the details on your own! The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.

Bagging typically results in improved accuracy over prediction using a single tree. However, the resulting model can be difficult to interpret. One advantage of decision trees is ease of interpretation. When we bag a large number of trees, we can no longer represent the resulting statistical learning procedure using a single tree and it is not clear which variables are the most important. Bagging improves prediction accuracy at the cost of interpretability.

Even though the collection of bagged trees is much more difficult to interpret that a single tree, we can obtain a summary of the importance of each predictor us the RSS (for bagging regression trees) or the Gini index (for bagging classification trees). In the case of regression trees, we can record the total amount that the RSS decreases due to splits over a given predictor,

averaged over all B trees. A large value indicates an important predictor. For classification trees, we can add up the total amount that the Gini index decreases by splits over a given predictor averaged over all B trees.

8.2 Random Forests

Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. As in bagging, we build a number of decision trees on bootstrapped training samples.

When building these decision trees, each time a split is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use *only one* of those m predictors. A fresh sample of m predictors is taken at each split, and typically we take $m \approx \sqrt{(p)}$, that is, the number of predictors considered at each split is approximately equal to, the square root of the total number of predictors.

When we build a random forest, at each split, the algorithm is not allowed to consider a majority of the available predictors. What is the reasoning?

Suppose the is one very strong predictor along with many moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. All of the bagged trees will look very similar. The predictions from the bagged trees will be highly correlated. Averaging highly correlated trees does not lead to as large of a reduction in variance as averaging many uncorrelated trees. This means that bagging will NOT lead to a substantial reduction in variance over a single tree for this example.

How do random forests overcome this problem? They force each split to consider only a subset of the predictors. On average, $(p-m)/p$ of the splits will not even consider a strong predictor and so other predictors have more of a chance! This process decorrelates the trees, making the average of the resulting trees less variables and hence more reliable.

What is the main difference between bagging and random forests? It's the choice of the predictor subset size m. For example, if the random forest is built using $m = p$, then this is the same as bagging. Using a small value of m in building a random forest will typically be helpful when we have a large number of correlated predictors.

Example 8.1: Bagging and Random Forests We perform bagging on the Boston dataset using the randomForest package in R. The results from this example will depend on the version of R installed on your computer.[3] We can use the randomforest() function to perform both random forests and bagging.

```
setwd("~/Dropbox/multivar_spring14/multi_notes/ch8_bagging_rof/bagging")
library(ISLR)
library(MASS)
library(randomForest)
attach(Boston)
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
bag.boston <- randomForest(medv~., data=Boston, subset=train, mtry=13, importantance
> bag.boston

Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13, importantance = TRUE,
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 11.13692
                    % Var explained: 86.52
```

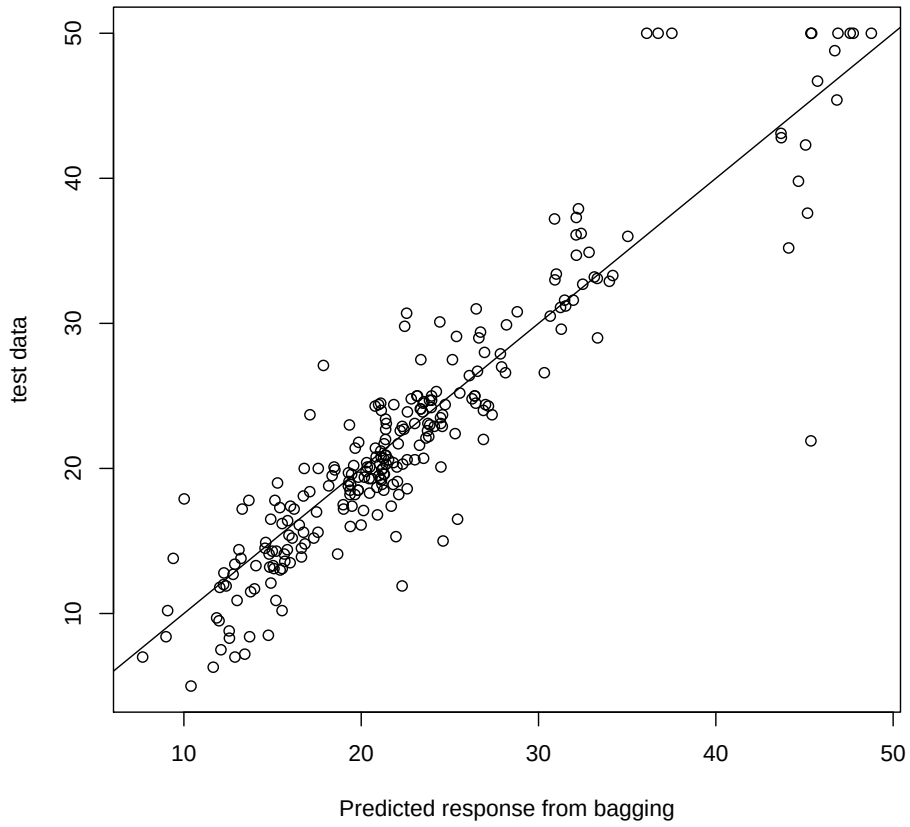
The argument mtry=13 indicates that all 13 predictors should be consider for each split of the tree. Basically, bagging should be done. How well does the bagged model perform on the test set?

The code below show that using tree=25 increases the test set MSE to 14.28.

```
bag.boston <- randomForest(medv~., data=Boston, subset=train, mtry=13, importantance
yhat.bag <- predict(bag.boston, newdata=Boston[-train,])
mean((yhat.bag - boston.test)^2)
```

Growing a random forest proceeds in exactly the same way, except we use a smaller value of the mtry argument. By default, randomForest() uses

[3] Recall that bagging is a special case of a random forest with $m = p$.




```
boston.test <- Boston[-train, "medv"]
yhat.bag <- predict(bag.boston, newdata=Boston[-train,])
plot(yhat.bag, boston.test, xlab="Predicted response from bagging", ylab="test data")
abline(0,1)
mean((yhat.bag - boston.test)^2)
```

FIGURE 8.1: Plot of the predictions of the test set of bagging versus the bagged test set. The test set MSE associated with the bagged regression tree is 12.71, about half that from using an optimally-pruned single tree. We could change the number of trees grown by randomForest() using the ntree argument.

$p/3$ variables when building a random forest of regression trees, and $\sqrt{(p)}$ variables when building a random forest of classification trees. Here we use a mtry=6.

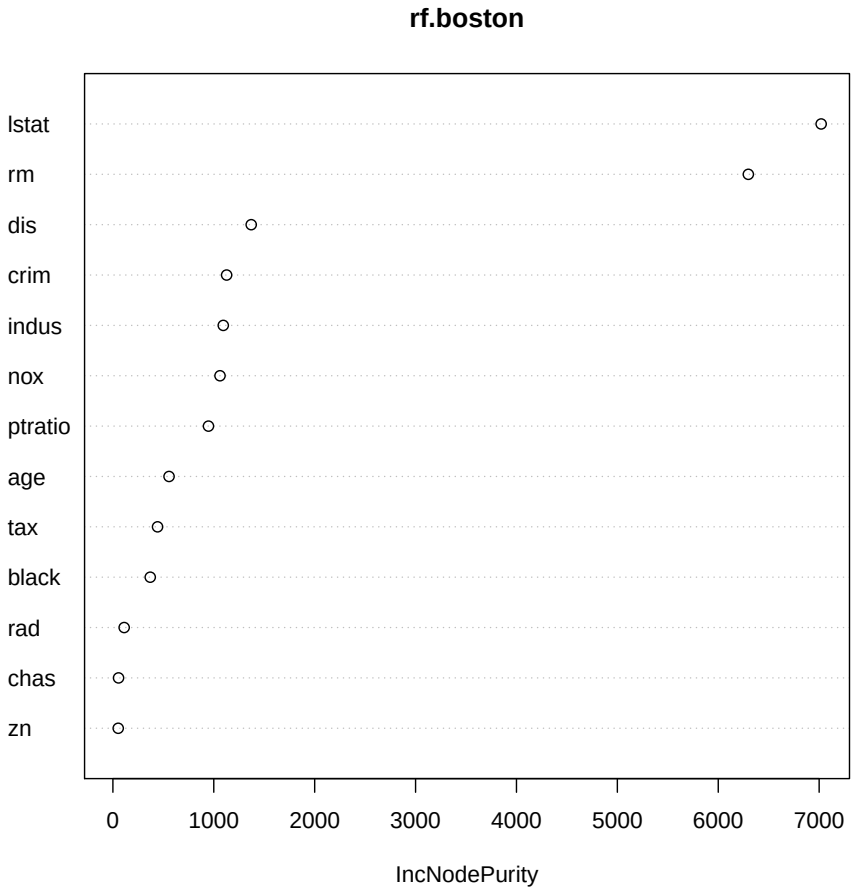
The test set MSE is 11.63, indicating that random forests yield an improvement over bagging.

```
set.seed(1)
rf.boston <- randomForest(medv~., data=Boston, subset=train, mtry=6, importantance =
yhat.rf <- predict(rf.boston, newdata=Boston[-train,])
mean((yhat.rf - boston.test)^2)
```

Using the importance() function, we can view the importance of each variable.

```
> importance(rf.boston)
        IncNodePurity
crim       1127.35130
zn           52.68114
indus      1093.92191
chas         56.01344
nox        1061.66818
rm         6298.06890
age         556.56899
dis        1371.10322
rad         111.89502
tax         442.61144
ptratio     947.18872
black       370.15308
lstat      7019.97824
```

Two measures of variable importance are reported. The former is based upon the mean decrease of accuracy in predictions in the out of bag samples when a given variable is excluded from the model. The latter is a measure of the total decrease in node impurity that results from splits over that variables, averaged over all trees. In the case of regression trees, the the node impurity is measure by the training RSS and for classification trees by the deviance. Plots of these importance measures can be produced using the varImpPlot() function.




```
varImpPlot(rf.boston)
```

FIGURE 8.2: Plot the increase in node purity and the importance of each predictor.

The results indicate that across all of the trees considered in the random forest, the wealth level of the community (lstat) and the house size (rm) are by far the two most important variables.