Bagging, Random Forests, Boosting

Trevor Hastie and Robert Tibshirani

Here, I am adapting part of the lab associated with Chapter 8 of the textbook.

The `tree` library is used to construct classification and regression trees.

```
#install.packages("tree")
library(tree)
library(ISLR2)
```

Bagging and Random Forests

Here we apply bagging and random forests to the `Boston` data, using the `randomForest` package in `R`. The exact results obtained in this section may depend on the version of `R` and the version of the `randomForest` package installed on your computer.

We start by splitting the data into training and testing.

```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston) / 2)
```

Recall that bagging is simply a special case of a random forest with $m = p$. Therefore, the `randomForest()` function can be used to perform both random forests and bagging. We perform bagging as follows:

```
#install.packages("randomForest")
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
bag.boston <- randomForest(medv ~ ., data = Boston,
    subset = train, mtry = 12, importance = TRUE)
bag.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 12, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 12
##
##           Mean of squared residuals: 11.25779
##                     % Var explained: 85.35
```

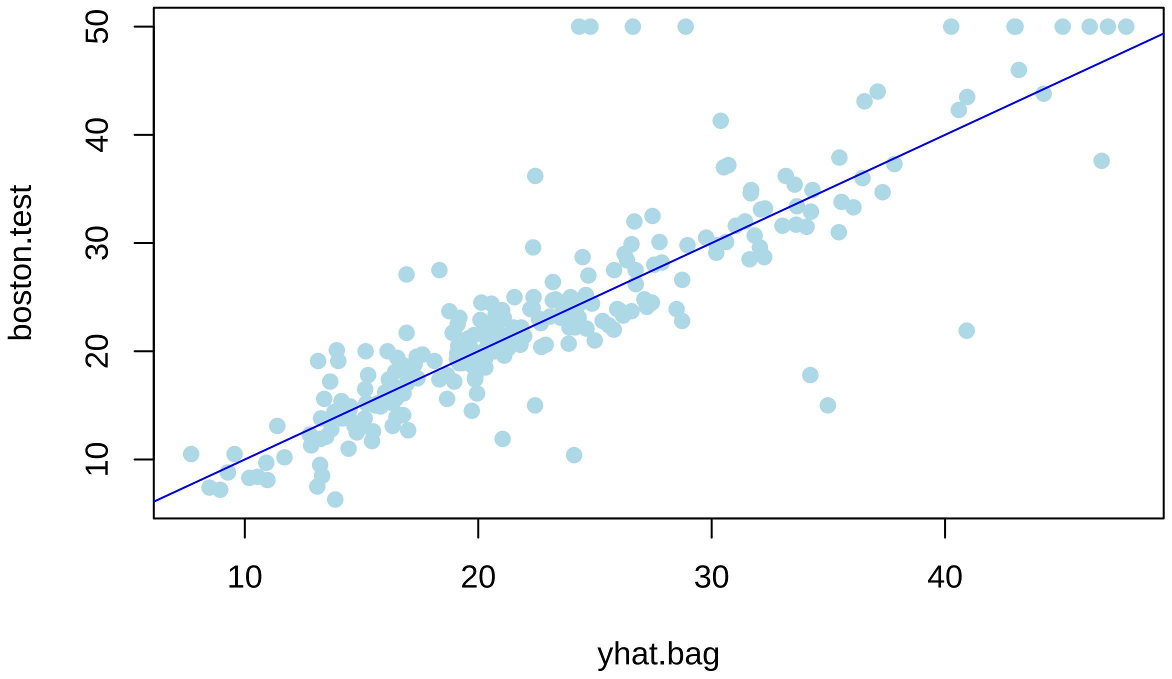
```
importance(bag.boston)
```

```
##            %IncMSE IncNodePurity
## crim    25.7789395     837.41518
## zn       4.5247559      63.76809
## indus    3.6515890     109.39475
## chas     -2.0917531      14.85085
## nox      18.9566434     278.64734
## rm       50.1429866   12345.48457
## age      17.4021374     361.03617
## dis       7.0482342     275.04378
## rad       0.9338507      71.48245
## tax       9.7454753     138.84145
## ptratio   8.7724030     149.61895
## lstat    45.2820959    4729.96535
```

The argument `mtry = 12` indicates that all 12 predictors should be considered for each split of the tree—in other words, that bagging should be done. How well does this bagged model perform on the test set?

```
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
boston.test <- Boston[-train, "medv"]

plot(yhat.bag, boston.test, pch=19, col="lightblue")
abline(0, 1, col="blue")
```



```
mean((yhat.bag - boston.test)^2)
```

```
## [1] 23.40359
```

The test set MSE associated with the bagged regression tree is 23.042, about two-thirds of that was obtained using an optimally-pruned single tree (we got 35.28688 last week). We could change the number of trees grown by `randomForest()` using the `ntree` argument:

```
bag.boston <- randomForest(medv ~ ., data = Boston,
    subset = train, mtry = 12, ntree = 25)
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
mean((yhat.bag - boston.test)^2)
```

```
## [1] 24.59162
```

Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the `mtry` argument. By default, `randomForest()` uses $p/3$ variables when building a random forest of regression trees, and $\sqrt{p}$ variables when building a random forest of classification trees. Here we use `mtry = 6`.

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

```
## [1] 20.06644
```

Looking at the test set MSE, we see that random forests yielded an improvement over bagging in this case.

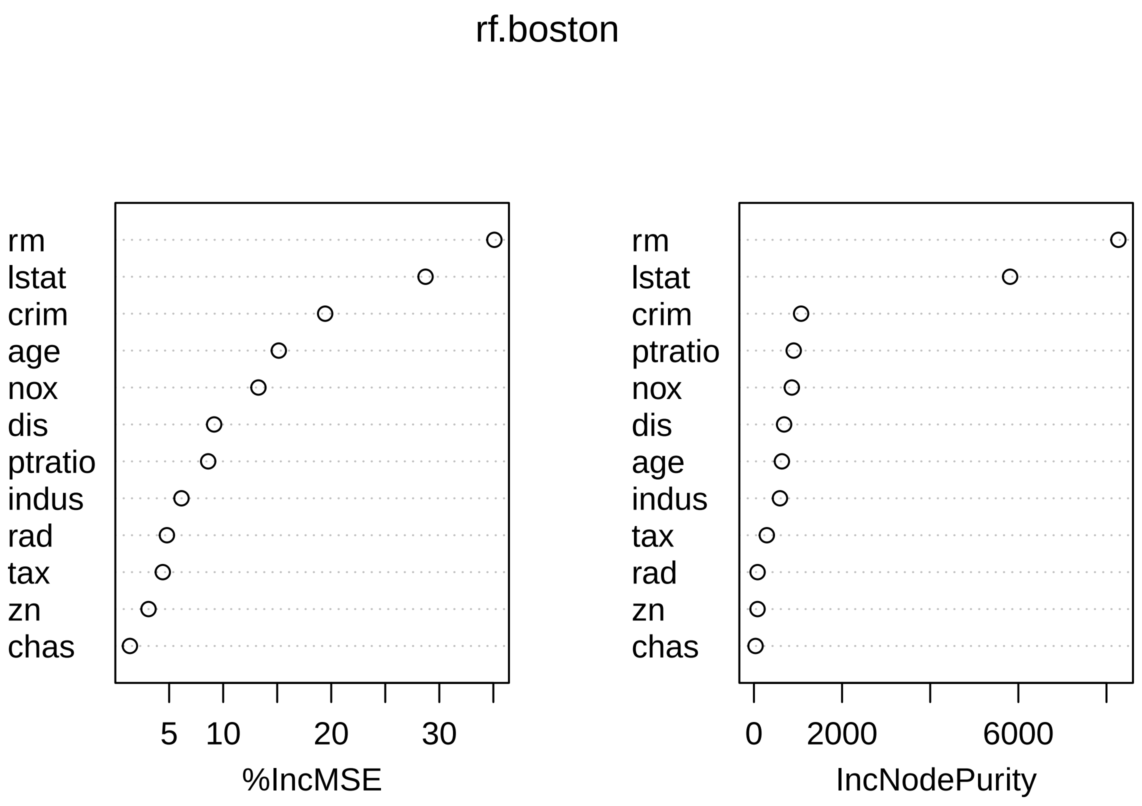
Using the `importance()` function, we can view the importance of each variable.

```
importance(rf.boston)
```

```
##            %IncMSE IncNodePurity
## crim    19.435587    1070.42307
## zn       3.091630      82.19257
## indus    6.140529     590.09536
## chas     1.370310      36.70356
## nox     13.263466     859.97091
## rm      35.094741    8270.33906
## age     15.144821     634.31220
## dis      9.163776     684.87953
## rad      4.793720      83.18719
## tax      4.410714     292.20949
## ptratio  8.612780     902.20190
## lstat   28.725343    5813.04833
```

Two measures of variable importance are reported. The first is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is permuted. The second is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees. In the case of regression trees, the node impurity is measured 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)
```



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

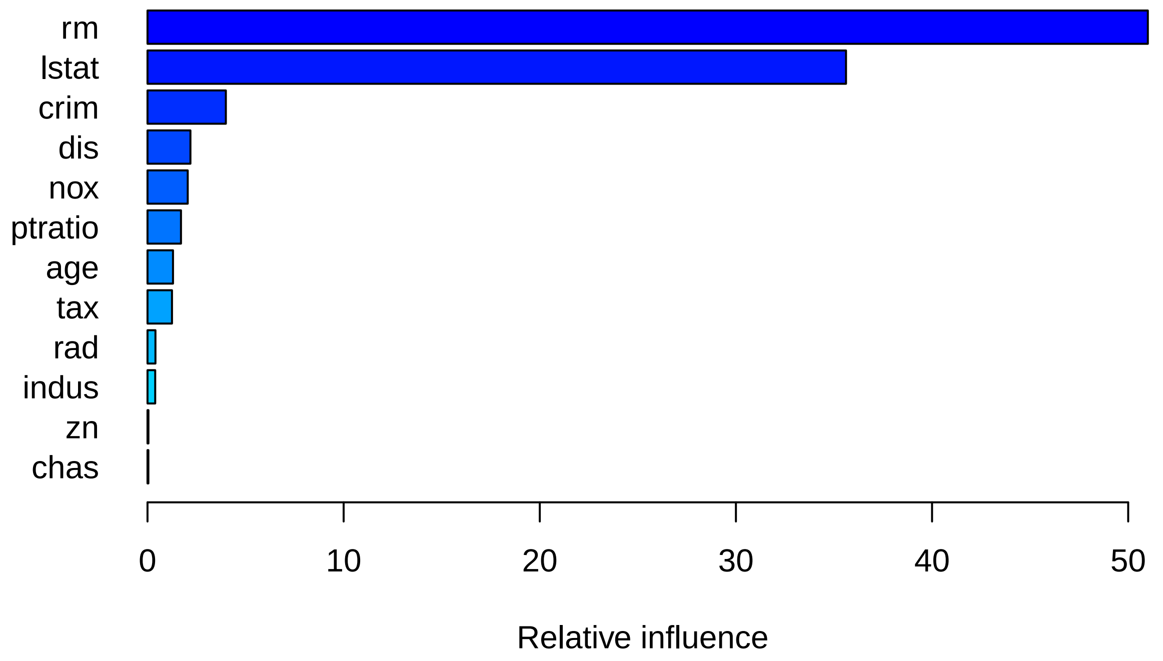
Boosting

Here we use the `gbm` package, and within it the `gbm()` function, to fit boosted regression trees to the `Boston` data set. We run `gbm()` with the option `distribution = "gaussian"` since this is a regression problem; if it were a binary classification problem, we would use `distribution = "bernoulli"`. The argument `n.trees = 5000` indicates that we want 5000 trees, and the option `interaction.depth = 4` limits the depth of each tree.

```
#install.packages("gbm")
library(gbm3)
set.seed(1)
boost.boston <- gbm(medv ~ ., data = Boston[train, ],
    distribution = "gaussian", n.trees = 5000,
    interaction.depth = 4)
```

The `summary()` function produces a relative influence plot and also outputs the relative influence statistics.

```
summary(boost.boston)
```



```
##             var     rel_inf
## rm           rm 51.00275717
## lstat     lstat 35.61373188
## crim       crim  3.99542680
## dis         dis  2.18787358
## nox         nox  2.05141644
## ptratio ptratio  1.70765318
## age         age  1.30098377
## tax         tax  1.24868581
## rad         rad  0.40512022
## indus     indus  0.39360782
## zn           zn  0.04778681
## chas       chas  0.04495651
```

We now use the boosted model to predict `medv` on the test set:

```
yhat.boost <- predict(boost.boston,
    newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 21.42144
```

The test MSE obtained is superior to the test MSE of random forests and bagging. If we want to, we can perform boosting with a different value of the shrinkage parameter λ in (8.10). The default value is 0.001, but this is easily modified. Here we take $\lambda = 0.2$.

```
boost.boston <- gbm(medv ~ ., data = Boston[train, ],
    distribution = "gaussian", n.trees = 5000,
    interaction.depth = 4, shrinkage = 0.2, verbose = F)
yhat.boost <- predict(boost.boston,
    newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
## [1] 19.18984
```

In this case, using $\lambda = 0.2$ leads to a lower test MSE than $\lambda = 0.001$.