Tree-Based Methods
Bagging, Random Forests, Boosting

November 6, 2017

Bagging, random forests, and boosting use trees as building blocks to construct more powerful prediction models.

Bagging

The **decision trees** discussed before 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. 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**.
Bootstrap aggregation, or bagging, is a general-purpose procedure for **reducing** the **variance** of a **statistical learning method**; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
Recall that given a set of n independent observations Z1, . . . , Zn, each with variance 2, the variance of the mean Z of the observations is given by 2/n. In other words, averaging a set of observations reduces variance. Hence a natural way to reduce the variance and hence increase the prediction 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.

In other words, we could calculate $\hat{f}_1(x), \hat{f}_2(x), ..., \hat{f}_B(x)$ using B separate training sets, and average them in order to obtain a single low-variance statistical learning model, given by

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

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 b^{th} bootstrapped training set in order to get $\hat{f}_b(x)$, and finally average all the predictions, to obtain

$$\hat{f}_{bag}(x) = 1/B \sum_{b=1}^{B} \hat{f}_b(x).$$

This is called bagging.
While **bagging can improve predictions** for many regression methods, it is particularly **useful for decision trees**. To apply bagging to **regression trees,**

- we simply construct B regression trees using B bootstrapped training sets. Generally, the value of B is not critical. In practise we use a value of B sufficiently large that the error has settled down,
- 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. Bagging has been demonstrated to give impressive improvements in accuracy by combining together hundreds or even thousands of trees into a single procedure.

Random Forests

Random forests provide an improvement over bagged trees by way of a random small tweak that decorrelates the trees. As in bagging, we build a number forest of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree 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 choose $m \approx \sqrt{p}$ that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.
The main difference between bagging and random forests is the choice of predictor subset size m. For instance, if a random forest is built using $m = p$, then this amounts simply to bagging.
Using a small value of m will typically be helpful with a large number of correlated predictors.

Boosting

Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification.
Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.
Boosting has three tuning parameters overall.

- The number of trees B. Except we must note that boosting can over fit if B is too large, although this overfitting tends to not be very severe. We use cross validation to select B.
- Shrinkage parameter lambda, a small positive number. This controls the boosting learning rate. Typical values are 0.01 or 0.001, but it depends on the problem. Smaller lambda requires larger B to achieve good performance.
- The number of d splits in each tree, which controls the complexity of the boosting. Often $d = 1$ works well, where each tree is a stump, consisting of a single split.

In general, boosting can improve upon random forests, and are easier to interpret because of the smaller tree structure.

Bagging and Random Forests

Here we apply bagging and random forests to the **Boston data**, using the **randomForest** package in R.
Therefore, the **randomForest() function** can be used to perform **both** random forests and bagging. We perform as follows:

```
> library (randomForest)
> set.seed (1)
> bag.boston =randomForest(medv.,data=Boston ,subset =train ,
mtry=13, importance =TRUE)
> bag.boston


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

Mean of squared residuals: 11.08966
% Var explained: 86.57
```

The argument mtry=13 indicates that all 13 predictors should be considered for each split of the treein other words, that bagging should be done.
By default random forests uses $p/3$ for regression trees, and $\sqrt{p}$ for classification trees.
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)
> abline (0,1)
> mean(( yhat.bag -boston .test)^2)
[1] 13.16
```

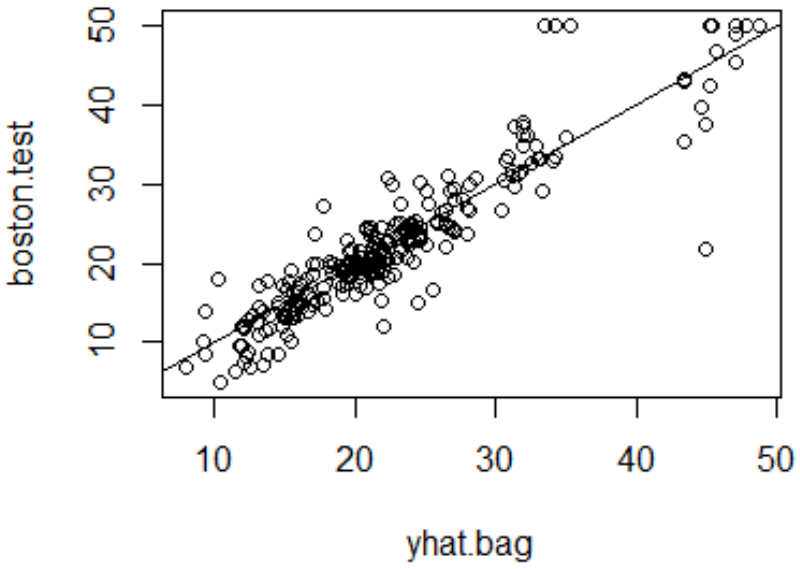

Figure 1

The test setMSE associated with the bagged regression tree is 13.16, almost half that obtained using an optimally-pruned single tree (ch14).

```
> yhat=predict (tree.boston ,newdata =Boston [-train ,]) ######### Test data
> boston .test=Boston [-train ," medv"]
> plot(yhat ,boston .test)
> abline (0,1)
> mean((yhat -boston .test)^2)
[1] 25.05

-----------------------------
> yhat=predict (prune.boston ,newdata =Boston [-train ,]) ######### Test data
> boston.test=Boston [-train,"medv"]
> plot(yhat ,boston.test)
> abline (0,1)
> mean((yhat-boston.test)^2)
[1] 26.83413
```

We could change the number of trees grown by randomForest() using the ntree argument:

```
> bag.boston =randomForest(medv.,data=Boston ,subset =train ,
mtry=13, ntree =25)
> yhat.bag = predict (bag.boston ,newdata =Boston [-train ,])
```

```
> mean(( yhat.bag -boston .test)^2)
[1] 13.31
```

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

```
> importance(bag.boston)

         %IncMSE IncNodePurity
crim    12.474613     763.42709
zn       2.650784      23.73589
indus   11.147631     166.99922
chas     2.152210      17.71925
nox     12.727681     264.45916
rm      46.932534    7548.02659
age     12.205679     306.24819
dis     20.001149    1127.39194
rad      3.401881      71.58605
tax      7.447069     324.27562
ptratio 14.585914     254.98738
black    7.003223     262.89269
lstat   37.014266    9766.13211
```

Two measures of variable importance are measured. The %IncMSE is based on the mean decrease in accuracy in predictions on the out of bag samples, when the given variable was excluded from the model. IncNodePurity is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees. We can plot the importance with varImpPlot().

```
varImpPlot(bag.boston)
```

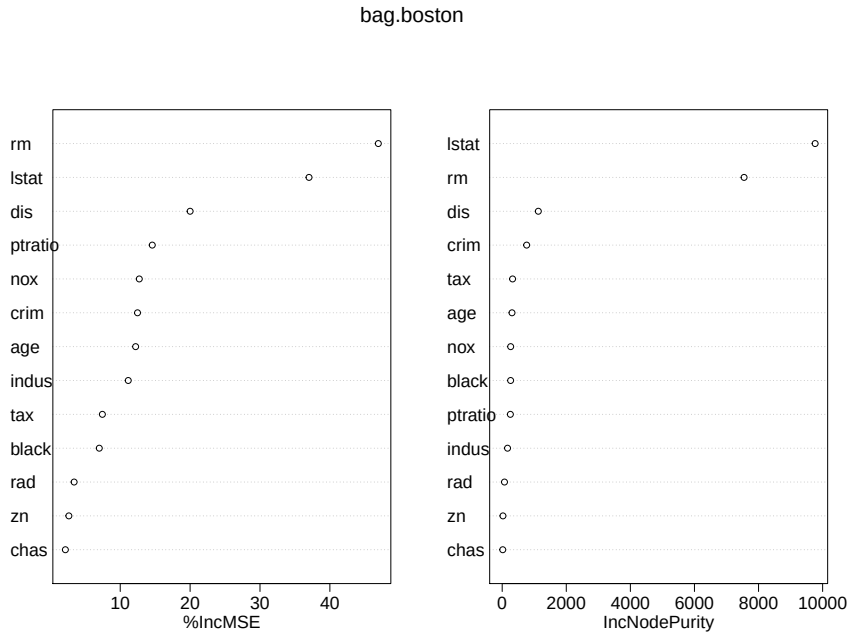

Figure 2: Variable importance

Random forest

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] 11.31
```

The test set MSE is 11.31; this indicates that random forests yielded an improvement over bagging in this case.
Using the importance() function, we can view the importance of each importance() variable.

The importance of each variable

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

```
> importance (rf.boston )
     %IncMSE  IncNodePurity
crim  12.384  1051.54
zn  2.103  50.31
indus  8.390  1017.64
chas  2.294  56.32
nox  12.791  1107.31
rm  30.754  5917.26
age  10.334  552.27
dis  14.641  1223.93
rad  3.583  84.30
tax  8.139  435.71
ptratio  11.274  817.33
black  8.097  367.00
lstat  30.962  7713.63
```

Two measures of variable importance are reported. The former is based upon the mean decrease of accuracy in predictions on 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 variable, averaged over all trees (this was plotted in Figure 8.9). 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 )
```

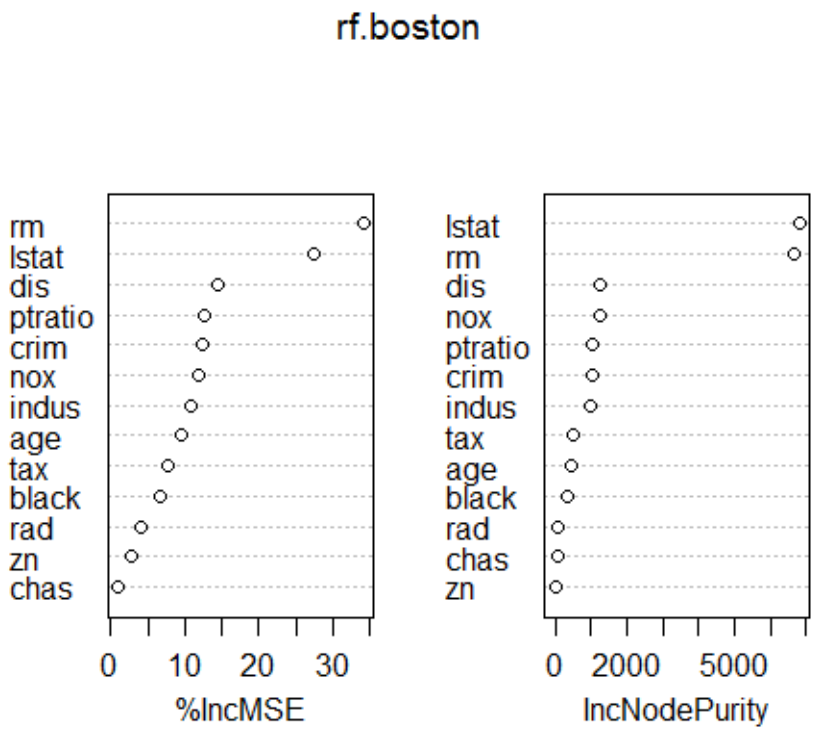

Figure 3: Importance of variables.

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.

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.

```
> library (gbm)
> 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
1 lstat 45.96
2 rm 31.22
3 dis 6.81
```

```
4 crim 4.07
5 nox 2.56
6 ptratio 2.27
7 black 1.80
8 age 1.64
9 tax 1.36
10 indus 1.27
11 chas 0.80
12 rad 0.20
13 zn 0.015
```

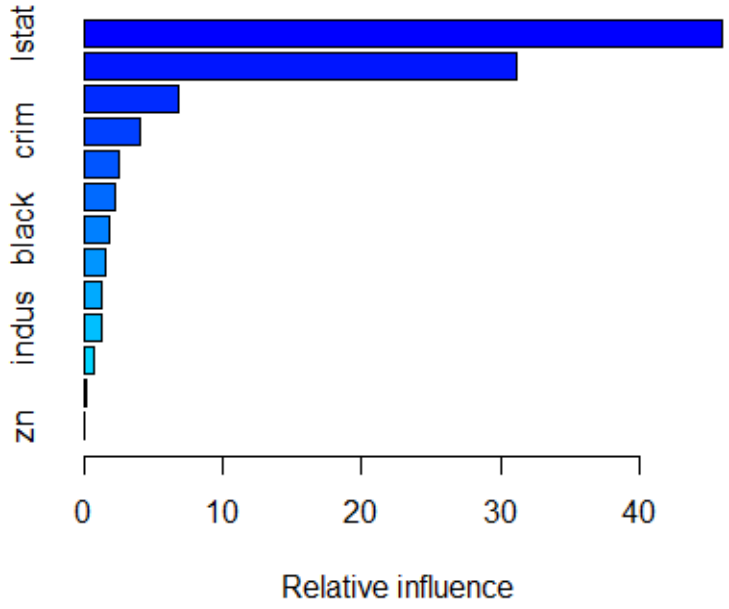

Figure 4: Boosting

partial dependence plot

We see that lstat and rm are by far the most important variables. We can also produce partial dependence plots for these two variables. These plots
illustrate the marginal effect of the selected variables on the response after integrating out the other variables. In this case, as we might expect, median house prices are increasing with rm and decreasing with lstat.

```
> par(mfrow =c(1,2))
> plot(boost.boston ,i="rm")
> plot(boost.boston ,i=" lstat ")
```

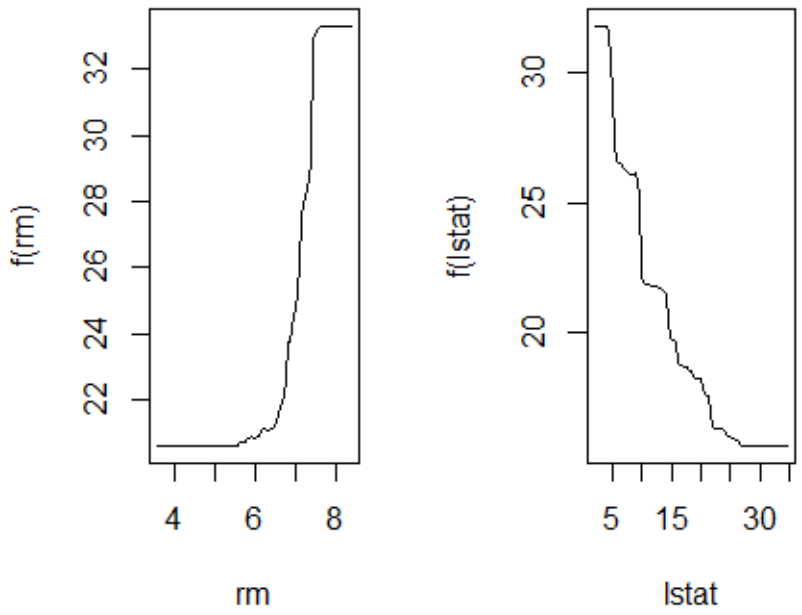

Figure 5: Partial dependence

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] 11.8
```

The test MSE obtained is 11.8; similar to the test MSE for random forests and superior to that for bagging.

R Code

```
install.packages("randomForest")
install.packages("MASS")
install.packages("knitr")
library (randomForest)
library (MASS)
library( knitr )
?Boston
View(Boston)
attach(Boston)
kable(head(Boston))
############################################################ Training and Test sets
############################ Traditional
```

```
set.seed (1)
?sample
train = sample (1: nrow(Boston ), nrow(Boston )/2)  ## Training data
boston.test=Boston [-train ,"medv"]   #### Test Data
###### Recent developet function
## Loading required package:
install.packages("caret")
install.packages("lattice")
install.packages("ggplot2")
library(caret)
library(lattice)
library(ggplot2)
set.seed(123)
split <- createDataPartition(y=Boston$medv, p = 0.5, list=FALSE)
train <- Boston[split,]
boston.test<- Boston[-split,]
######################################### Grow the  Tree
?randomForest()
bag.boston =randomForest(medv~.,data=Boston ,subset =train , mtry=13, importance =TRUE)
bag.boston
importance(bag.boston)
varImpPlot(bag.boston)
########################################### HOw good this  bagged model perform on the test se
yhat.bag = predict(bag.boston ,newdata =Boston [-train ,])
boston.test=Boston [-train ,"medv"]
plot(yhat.bag, boston.test)
abline (0,1)
mean(( yhat.bag-boston.test)^2)

#####################################################Randomforest Regression Tree (p/3 predictor)
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)
############################ Importance of variable
importance (rf.boston )
varImpPlot (rf.boston )
#############################################################Boosting Method
install.packages("gbm")
library (gbm)
set.seed (1)
boost.boston =gbm(medv~.,data=Boston [train ,], distribution="gaussian",n.trees =5000
, interaction.depth =4)
summary(boost.boston )
#############################################partial depentece plot
par(mfrow =c(1,2))
plot(boost.boston ,i="rm")
```

```
plot(boost.boston ,i="lstat")
######################################### How good Boosting modeld the regression Tree model
yhat.boost=predict (boost.boston ,newdata =Boston [-train ,],
n.trees =5000)
mean(( yhat.boost-boston.test)^2)
```