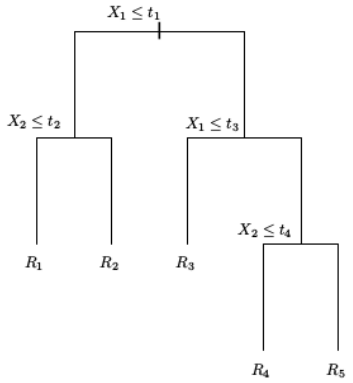
STAT 621 Lecture Notes
Regression and Classification Trees Part 2

Today we will continue our discussion of tree methods for predicting or classifying responses. We'll discuss some issues that you should be aware of with these models, and take a (superficial) look at some more advanced techniques.

First recall the basic idea of the tree model. Here's our simple example (from AISL):



Some Shortcomings of Tree Models:

(1) Variance:

(2) Lack of Smoothness:

(3) Modeling Additive Effects:

Some Extensions:

A number of teweaks on the basic tree model have been proposed to alleviate one or more of these limitations. We will focus on some common procedures: Bagging, Random Forests and Boosting. We will give a basic description of these and look at examples. Other interesting techniques discussed in ESL are Bump Hunting and Smooth Tree Models using multivariate adaptive regression splines.

Bagging: This uses bootstrap methods to reduce variability and improve prediction in tree models. The term Bagging is short for *Bootstrap Aggregation*.

- First the basic idea for a continuous response.

- Binary responses.

- Interpretations.

Random Forests: This is an extension of the Bagging algorithm that uses bootstrapping, but averages over sets of *decorrelated* trees to predict the response and measure variable importance. Here's how it works.

Example: Let's try using Bagging and Random Forests to model median housing prices in Boston suburbs, using data from the `Boston` data set. Here we will use the `randomForest` package for both procedures.

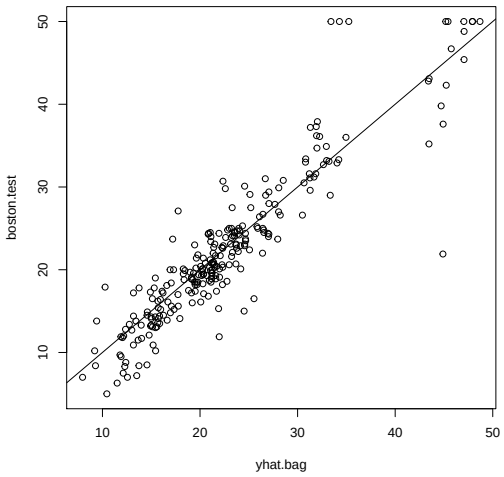
Recall our original model (in the previous set of lecture notes) fit a tree with 8 nodes and used the three variables `lstat`, `rm` and `dis` to predict median house price. The resulting predication MSE measured on the test data was about 25.05.

Bagging: Let's see if bagging will improve this fit. I'll use the same training set here that was used to fit the original model. For bagging we use all 13 predictors at each split (`mtry=13` below). The `importance` option requests numerical measures of importance for each predictor.

```
library(randomForest); library(MASS)
attach(Boston)

set.seed(1)
train = sample(1: nrow(Boston), nrow (Boston)/2)
boston.test = Boston[-train,"medv"]
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
```

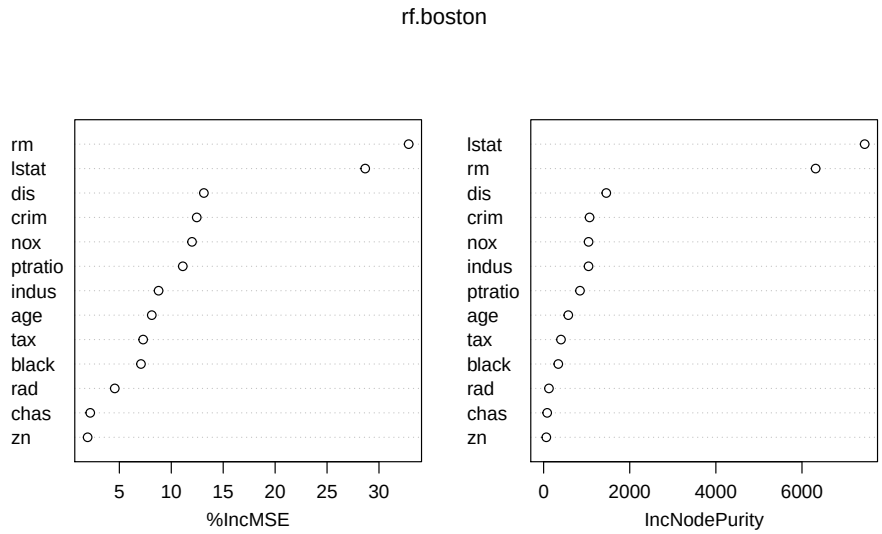
OK, well let's see how well we do predicting. Which variables are most important?

```
yhat.bag = predict(bag.boston, newdata=Boston[-train,])
plot(yhat.bag, boston.test)
abline(0,1)
mean((yhat.bag-boston.test)^2)
  [1] 13.33831

round(importance(bag.boston), 2)
        %IncMSE IncNodePurity
crim      15.56        963.94
zn         1.65         22.23
indus      7.26        191.92
chas       1.44         20.50
nox        9.50        242.39
rm        47.95       7469.85
age       11.79        316.53
dis       17.59        886.22
rad        3.52         69.04
tax        8.11        314.16
ptratio   13.96        273.75
black      7.21        251.61
lstat     37.87       9571.70
```



Random Forest: Now let's fit a random forest model. Here we use fewer predictors at each split. The defaluts are $p/3$ for regression trees and $\sqrt{p}$ for classification trees. Below I set `mtry=6`. How do we do?

```
rf.boston = randomForest(medv~., data=Boston, subset=train, mtry=6, importance=TRUE)
yhat.rf = predict(rf.boston, newdata=Boston[-train,])
plot(yhat.rf, boston.test)
abline(0,1)                    # not shown
mean((yhat.rf-boston.test)^2)
[1] 11.50616

round(importance(rf.boston),2)
        %IncMSE IncNodePurity
crim      12.45       1067.13
zn         1.94         62.51
indus      8.78       1042.81
chas       2.20         82.51
nox       12.00       1043.79
rm        32.86       6317.14
age        8.12        572.32
dis       13.14       1455.04
rad        4.56        123.44
tax        7.29        402.99
ptratio   11.11        844.22
black      7.08        340.36
lstat     28.67       7456.00


varImpPlot(rf.boston)       # visual summary of importance
```

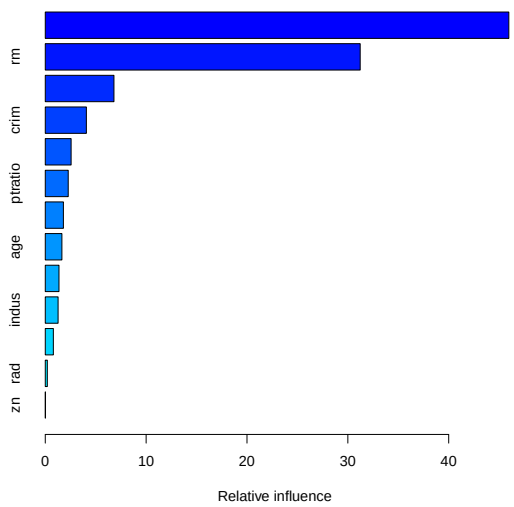


Boosting: Another technique to improve predictions. Here instead of fitting a separate tree to each bootstrap data set, boosting uses the previous fit to determine the next tree. Thus, trees are grown slowly. For more details see Algorithm 8.2 in AISL.

Example: Let's try boosting with the `Boston` data set. The R package `gbm` contains functions for doing that. The command is below. Since the response is continuous I use `distribution="gaussian"` (options for categorical responses are `binomial` and `multinomial`). Other options specify the number of trees to fit and the number of splits for each. The `summary` command displays both visual and tabular measures of variable importance

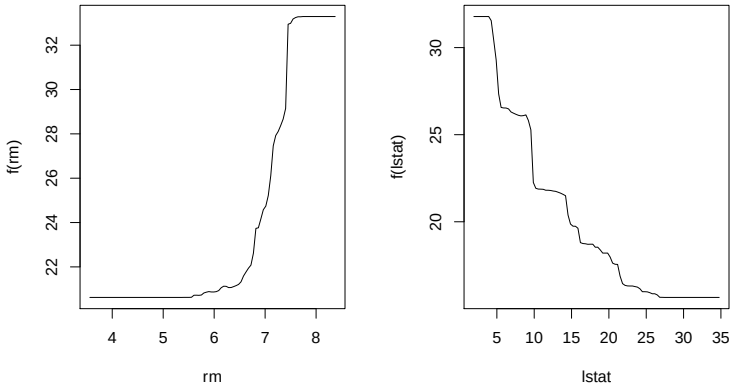
```
library(gbm)
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian",n.trees=5000,interaction.depth=4)
summary(boost.boston)
```

```
           var    rel.inf
lstat     lstat 45.9627334
rm           rm 31.2238187
dis         dis  6.8087398
crim       crim  4.0743784
nox         nox  2.5605001
ptratio ptratio  2.2748652
black     black  1.7971159
age         age  1.6488532
tax         tax  1.3595005
indus     indus  1.2705924
chas       chas  0.8014323
rad         rad  0.2026619
zn           zn  0.0148083
```



Let's look at some other results. We can plot *marginal* effects for individual predictors, and measure prediction MSE.

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

yhat.boost = predict(boost.boston, newdata=Boston[-train,], n.trees=5000)
mean((yhat.boost-boston.test)^2)
  [1] 11.84434
```



Comments?