Bagging and Random Forests

Matt Richey

4/11/2020

Setup

Load in some synthetic data so use with Decision Trees.

```
dataDir <- "/Users/richeym/Dropbox/COURSES/ADM/ADM_S20/CODE/Chap08"
file1 <- "RectData_Quant.csv"
file2 <- "Classify2D_Data3.csv"
regData.df <- read.csv(file.path(dataDir,file1))
classData.df <- read.csv(file.path(dataDir,file2)) %>%
  select(-row)
```

And the libraries....

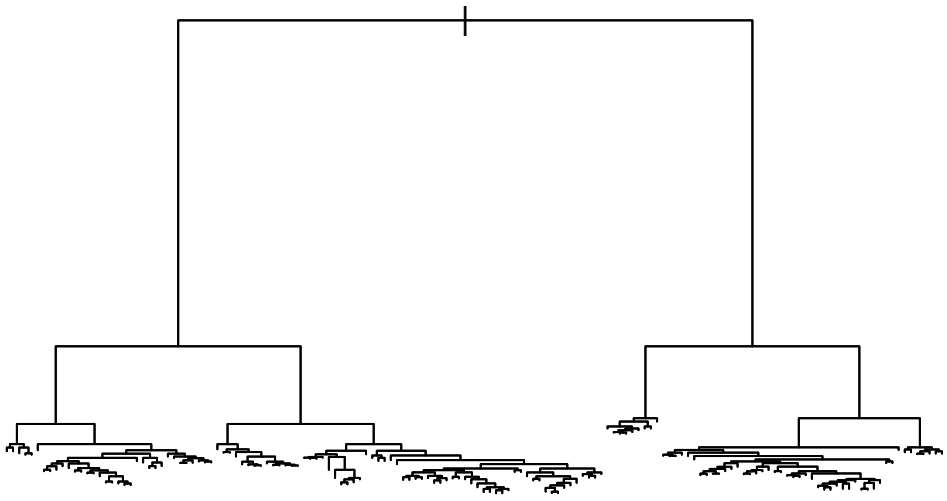
```
library(tidyverse)
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

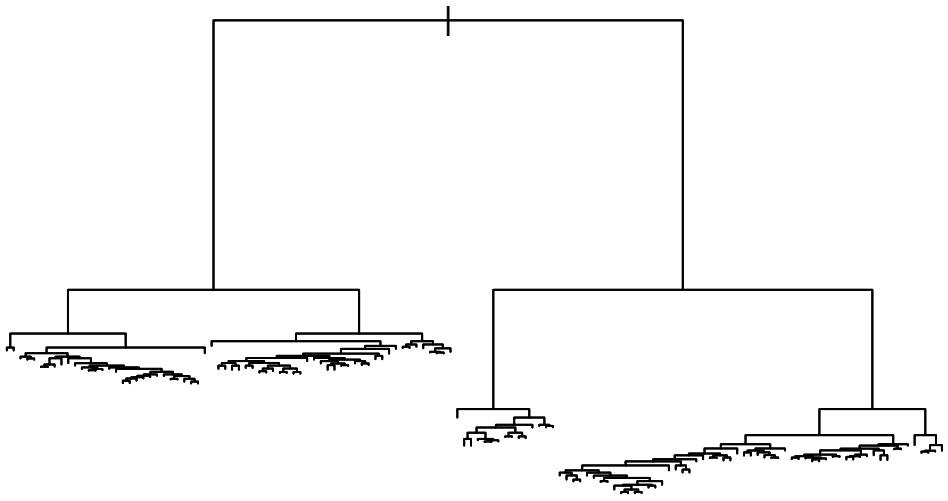
```
data.df <- classData.df

doBootPlot <- function(minDev,minSize){
 N <- nrow(data.df)
 boots <- sample(1:N,N,rep=T)
 boot.df <- data.df[boots,]
 mod.tree <- tree(class ~ x + y,
                  data=boot.df,
                  control=tree.control(nrow(boot.df),
                                       mindev=minDev,
                                       minsize=minSize))
 plot(mod.tree)
}
```

These are very different. This is a major deficit of decision trees. Small changes in the data result in large changes in the resulting tree. This effect gets more pronounced as the trees get deeper.

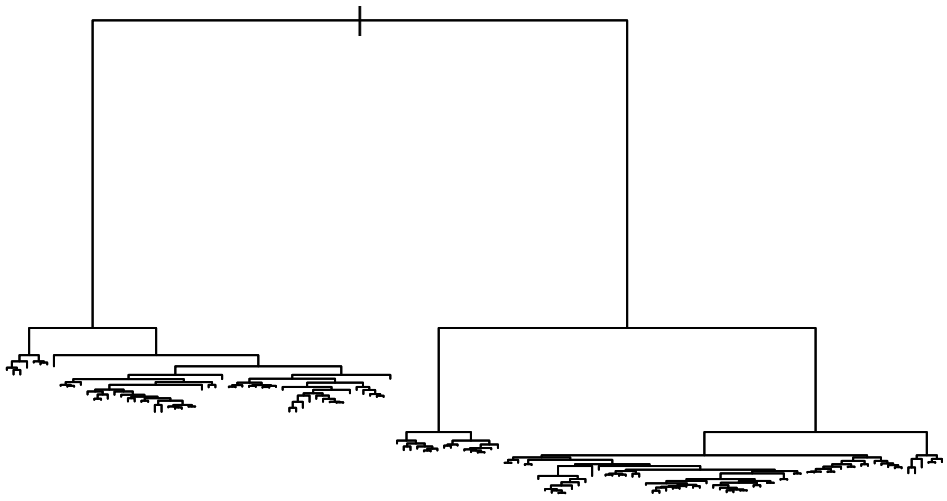
```
doBootPlot(.0001,10)
```



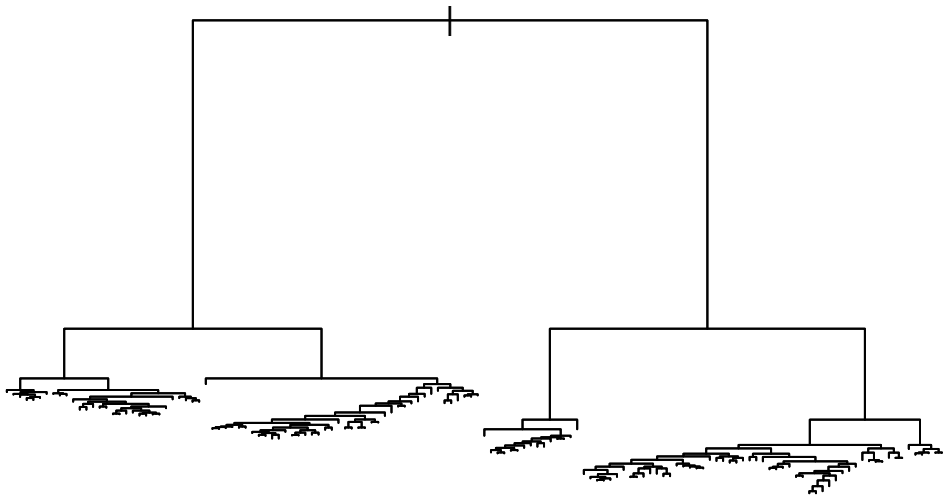
```
doBootPlot(.0001,10)
```



```
doBootPlot(.0001,10)
```



```
doBootPlot(.0001,10)
```



Bagging Basics

We can turn this deficit into an advantage. Here's the idea. For a given data set, generate a large number of bootstrapped data sets. For each build a **deep** regression tree. To make a prediction on new value, run the new value through **all** of the trees. Use the mean (or majority, if classification) as your prediction.

This is called Bagging, for Bootstrap Aggregation.

Let's do it by hand on our data. Notice that after building our bootstrapped data set, there will be a bunch of observations left over. These are called "Out of Bag" (OOB). We can use these to compute an estimate test error! That is, we do some cross-validation on the fly.

```
N <- nrow(classData.df)

n <- 200
build <- sample(1:N,n,rep=F)
data.df <- classData.df[build,]

##Use this later
test <- sample(setdiff(1:N,build),n,rep=F)
test.df <- classData.df[test,]
```

Let's build a bagged set of trees.

```
dim(data.df)
```

```
## [1] 200   3
```

```
numBoots <- 100
minDev <- .0001
minSize <- 4
numPred <- ncol(data.df)-1
## Here's where we keep our trees and the errors
## Note: this has to be a list to handle complex objecsts
## such as trees.
bootedTrees <- list()
oobErrs <- numeric(numBoots)

for(k in 1:numBoots){
  boots <- sample(1:n,n,rep=T)
  boot.df <- data.df[boots,]
  ## Left over observations =  Out Of Bags (OOBs)
  oobs <- setdiff(1:n,boots)
  oob.df <- data.df[oobs,]
  #### Build the tree
  bootedTree<- tree(class ~ x + y,
                    data=boot.df,
                    control=tree.control(nrow(boot.df),
                                         mindev=0.0001,
                                         minsize=4))
  ## Predict on the OOBs
   preds <- predict(bootedTree,
                    newdata=oob.df,
                    type="class")
  ##compute the error rate
  oobErrs[k] <- with(oob.df,mean(class != preds))
  ##keep the trees.. these are the predictive model
  bootedTrees[[k]] <- bootedTree
}
```

From the obbErrs we can get a rough estimate of the error rate. Turns out we, can do slightly better. More on this later

```
##estimate the error
mean(oobErrs)
```

```
## [1] 0.2082216
```

Not too bad. We now want to use all these bootstrapped trees to make predictions on any data frame.

The function that does this is pretty straight forward. For an arbitrary data frame, run through the list of booted trees and make the predictions. Store the results and use that to make a prediction based on majority vote.

```
## T
probBoot <- function(data.df,bootTrees){
```

```
  numBoots <- length(bootTrees)
  ##row for each observation, column for each boot
   preds <- matrix(nrow=nrow(data.df),
                   ncol=numBoots)
  for(k in 1:numBoots){
   preds[,k]  <- predict(bootTrees[[k]],newdata=data.df,type="class")
  }
   ##take a vote across the rows to get a prediction
   ## For each observation, returns the proportion of predictions
   ## that are in the  first class.
  apply(preds=="1",1,mean)
}
```

```
probs <- probBoot(test.df,bootedTrees)
with(test.df,table(class,probs < 0.5))
```

```
##
## class FALSE TRUE
##     A   107   16
##     B    18   59
```

```
(err.bag <- with(test.df,mean((class=="B") != (probs < 0.5))))
```

```
## [1] 0.17
```

Comparison with Logistic Regression

Compare with logistic regression...

```
mod.log <- glm(class ~ x + y,
               data=data.df,
               family="binomial")
probs.log <- predict(mod.log,
                  newdata=test.df,
                  type="response")
with(test.df,table(class,probs.log > 0.5))
```

```
##
## class FALSE TRUE
##     A   105   18
##     B    16   61
```

```
(err.log <- with(test.df,mean((class=="B") != (probs.log > 0.5))))
```

```
## [1] 0.17
```

```
c(err.bag,err.log)
```

```
## [1] 0.17 0.17
```

In this case, it looks as if this simple "bagging" of trees works at least as well, if not better than, logistic regression. Of course, YMMV, but it turns out that bagging is a a very effective modeling tool. It works especially well in situations where there a large number of predictors.

Bagging in R

Of course, R has very efficients functions for doing bagging.

The bagging function is a special case of what is called a Random Forest (of trees).

```
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

Bagged Classification Trees in R

We can recreate the bagging from above using randomForest.

```
numTree <- 100
numPred <- 2
mod.bag <- randomForest(class ~ x + y,
                        data=data.df,
                        ntree=numTree,
                        mtry=numPred) ##mtry=2 = Number of predictors
mod.bag
```

```
##
## Call:
##  randomForest(formula = class ~ x + y, data = data.df, ntree = numTree,      mtry = numPred)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 16%
## Confusion matrix:
##    A  B class.error
## A 90 17   0.1588785
## B 15 78   0.1612903
```

The randomForest function use the OOB values to compute a prediction for each observation. The idea is simple. As it goes through the boots, it keeps track of the OOB predictions. Of course, only a fraction of the observations are present in any one OOB. But over the course of the bootstrapping, each observation gets predicted a fair number of times (about numTrees/3, as it turns out). The majority vote of each of these is the prediction. From this the OOB estimate of the error rate is determined.

Assignment

Modify our bootstrapped method above so that it computes a class prediction for each observation. Make sure that your results agree (up to randomness) with the randomForest function.

Plan: Create a matrix with one row for each observation and one column for each tree (bootstrap). Each time through, put the OOB predictions into their rows of the matrix (the column is the current boot strap index). Careful: the predictions from the tree are usually given values 1,2, instead of "A", "B".

When done, do a majority vote across each row. In each row, only about 1/3 of the values will be filled in. You have to ignore the NAs.

```
##      preds
## class  A  B
##     A 91 16
##     B 17 76
```

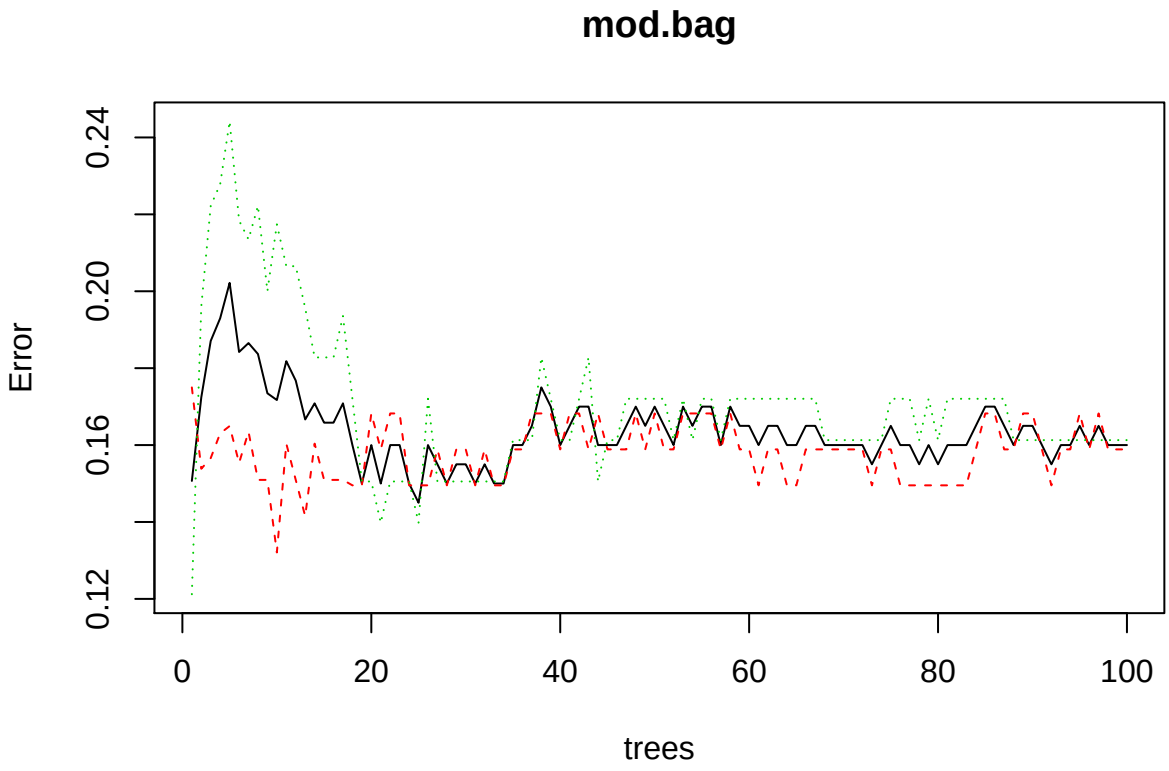
```
## [1] 0.165
```

```
##      oobPred
## class  A  B
##     A 91 16
##     B 17 76
```

```
## [1] 0.165
```

There is a built-in plot function showing the error rate as the number of trees increases.

```
plot(mod.bag)
```



As usual, predictions are straightforward.

```
preds <- predict(mod.bag,newdata=test.df)
with(test.df,table(class,preds))
```

```
##       preds
## class   A   B
##     A 102  21
##     B  16  61
```

```
with(test.df,mean(class != preds))
```

```
## [1] 0.185
```

Again, about the same error rate.

Solubility Data Set

Now we look at a real data set. The data are trying to related various chemical properties to solubility. The response variable, solubility, is continuous.

Hence we are doing Regression, not Classification, trees. For the most part, things work about the same way.

Get a look at the data.

```
solubil.df <- read.csv("APM_Solubility.csv")
dim(solubil.df)
```

```
## [1] 1267  229
```

```
head(names(solubil.df))
```

```
## [1] "FP001" "FP002" "FP003" "FP004" "FP005" "FP006"
```

```
N <- nrow(solubil.df)
numPred <- ncol(solubil.df)-1
```

You could do a correlation plot here to get a feel for the relationship between the predictors.

Test/validation data.

Extract a data set on which to build the model and another data set on which to test/validate the results.

```
n <- 500
build <- sample(1:N,n,rep=F)
data.df <- solubil.df[build,]

##Data to test/validate the model
n <- 500
test <- sample(setdiff(1:N,build),n,rep=F)
test.df <- solubil.df[test,]
```

Now we can buld a build a Bagged model with 100 tees using the R randomForest function.

Again, in randomForest that is a argument (mtry) which indicates the number of predictors to use at each branch. For now, we'll use all the predictors. Later, we'll limit the number of predictors.

```
numTree <- 100
mod.bag <- randomForest(solubility ~ .,
                        data=data.df,
                        ntree=numTree,
                        mtry=numPred)  ## Use all the predictors
mod.bag
```

```
##
## Call:
##  randomForest(formula = solubility ~ ., data = data.df, ntree = numTree,      mtry = numPred)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 228
##
##           Mean of squared residuals: 0.5410642
##                     % Var explained: 86.23
```

The "Mean of squared residuals" is the Out of Bag estimate of the MSE (as opposed to the error rate)

Assignment:

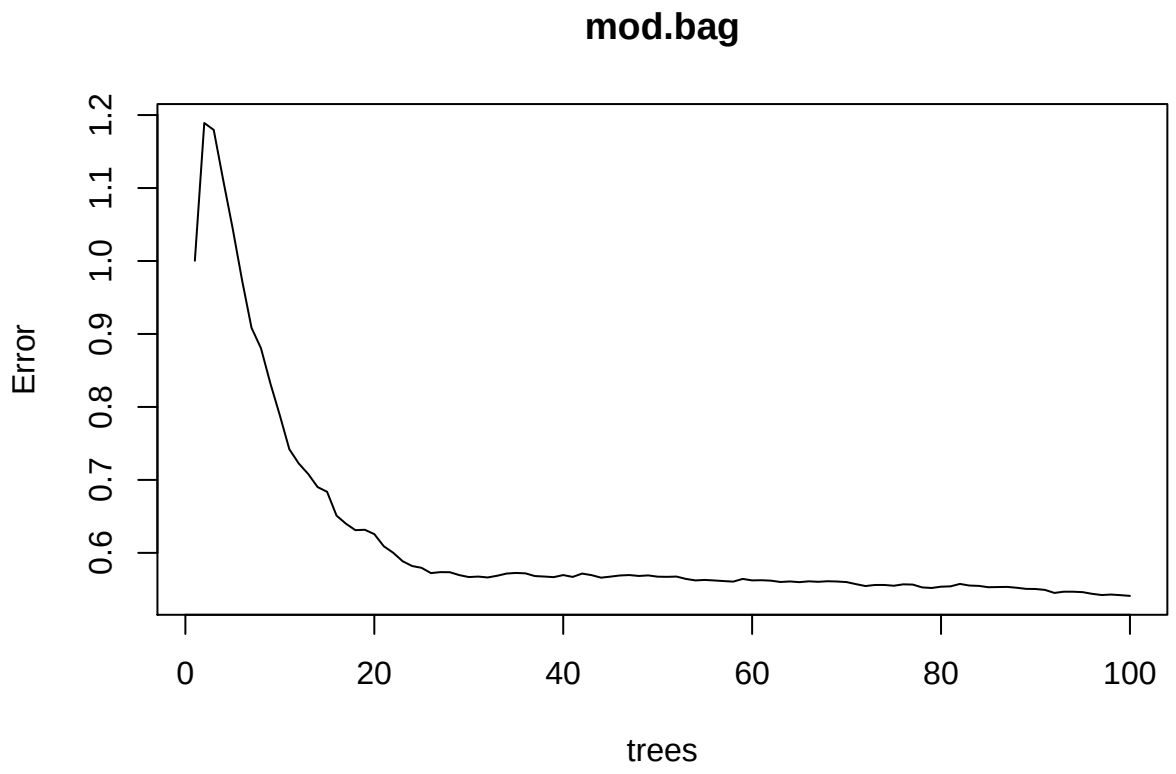
Replicate the results computed in mod.bag using a modification (to perform regression instead of classification) of the bootstrapping done above. For each bootstrapped tree, keep track of the oob predictions. When done, use these to make a prediction for each observation and then use these to estimate the MSE.

Compare you bootstrapped prediction and oob MSE with the results from mod.bag. They should be very similar.

Back to Bagging.

The built-in plot function shows the decrease in MSE as the number of trees increases.

```
plot(mod.bag)
```



Bagging is nice because it is essentially impossible to overfit! The more trees, the better. However, as this plot shows, there are diminishing returns on the error rate decrease as the number of trees increased. In any case, the only real limitation on the number of trees is computing time (and your time).

Making predictions on the test data is easy.

```
preds.bag <- predict(mod.bag,
                     newdata=test.df)
```

And computing the MSE on the test data.

```
(mse.bag <- with(test.df,mean((solubility - preds.bag)^2)))
```

```
## [1] 0.5684503
```

Ok, this looks pretty good. We will, of course, compare this to the MSE estimates from other prediction methods

Digging deeper

If you ask the model nicely, it will provide a great deal more information. For example, for the test data, we can extract the error rates from each of the numTree trees!

```
pred <- predict(mod.bag,
                ## Use test data
                newdata=test.df,
                ## Keep predictions from all of the trees
                predict.all = TRUE)
```

Here are the final predictions for each observation.

```
predsBag <- pred$aggregate
length(predsBag)
```

```
## [1] 500
```

For each observation, here are the predictions from each of the individual trees! Note: One row for each observation, one column for each tree

```
allPreds <- pred$individual
dim(allPreds)
```

```
## [1] 500 100
```

For example, we could, if wanted, make a prediction from a single tree, say the 10th one.

```
oneTree <- 10
with(test.df,mean((allPreds[,oneTree]-solubility)^2))
```

```
## [1] 1.565267
```

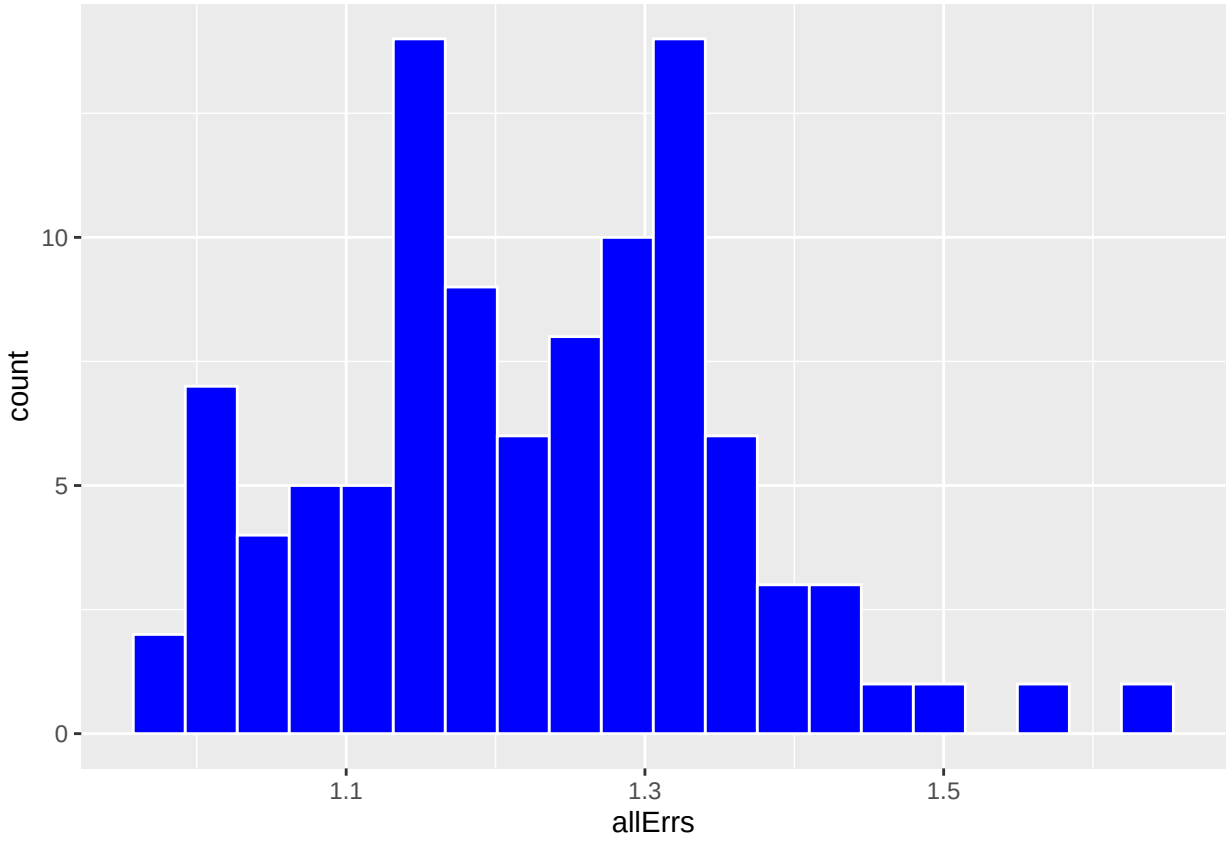
Not surprisingly, the prediction from a single tree isn't too impressive.

Let's make predictions from each of the trees.

```
allErrs <- map_dbl(1:numTree,
                   function(k) with(test.df,mean((allPreds[,k]-solubility)^2))   )
```

Let's look at a histogram of individual predictions

```
ggplot()+
  geom_histogram(aes(allErrs),fill="blue",color="white",bins=20)
```



Not too good. The overall mean of the individual error rates isn't too impressive: 1.2235763. Remember, the bagged tree has an overall MSE of less than 0.5. At first it is odd to think that an individual tree has a MSE near 1.0, but collectively they produce an error rate of 0.5.

Wisdom of the Crowds

We can easily retrieve the bagging estimate. Take the mean of each row=observation. This is exactly the prediction of the randomForest.

```
meanPreds <- apply(allPreds,1,mean)
head(cbind(meanPreds,predsBag))
```

```
##       meanPreds   predsBag
## 979  -1.8217750 -1.8217750
## 1212  0.7295183  0.7295183
## 677  -2.6022867 -2.6022867
## 95   -0.9316267 -0.9316267
## 1204 -0.5001333 -0.5001333
## 1037  0.5837550  0.5837550
```

What if we only used the first 10 trees? How would this compare with using all 100 trees?

```
subTrees <- 10
somePreds <- allPreds[,1:subTrees]
meanPreds <- apply(somePreds,1,mean)
mse.bag10 <- with(test.df,mean((meanPreds-solubility)^2))
c(mse.bag,mse.bag10)
```

```
## [1] 0.5684503 0.6444037
```

As expected, the prediction isn't quite as good. In a similar way, we could recreate plot(mod.bag) precisely. Try it.

Comparison to a simple linear model

Now let's get a comparison to other methods. Start witha linear model using all the predictors.

```
mod.lm <- lm(solubility ~ .,
                         data=data.df)
preds.lm <- predict(mod.lm,newdata=test.df)
```

```
## Warning in predict.lm(mod.lm, newdata = test.df): prediction from a rank-
## deficient fit may be misleading
```

```
(mse.lm <- with(test.df,mean((solubility - preds.lm)^2)))
```

```
## [1] 0.7313023
```

Note the warning coming out of the predict function. This indicates that there is some serious correlations going on in the data set. One would want to do some feature selection to clean it up.

Or, use penalized regression..... ## Ridge Regression

Now let's model with ridge regression. You could use lasso as well.

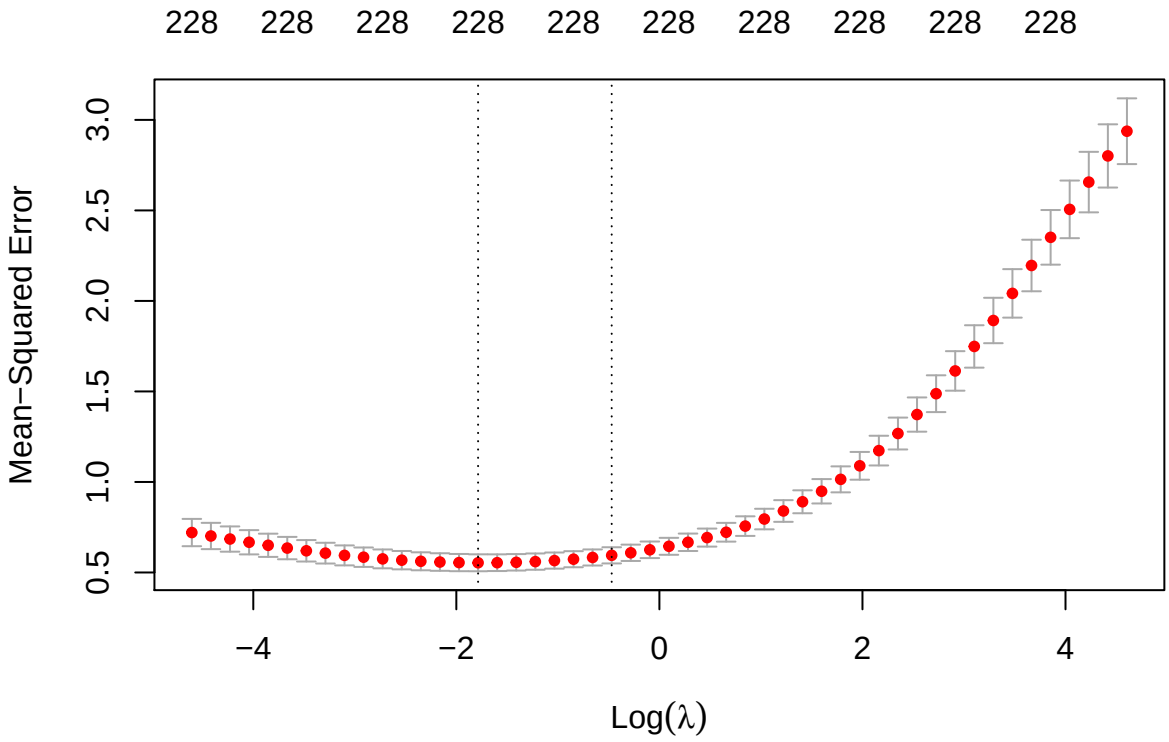
```
suppressMessages(library(glmnet))
```

Set up the data for glmnet.

```
data.x <- data.matrix(data.df[,1:numPred])
data.y <- data.matrix(data.df[,numPred+1])
```

Define a lambda grid and cross-validate.

```
lambda.grid <- 10^seq(-2,2,length=50)
mod.ridge.cv <- cv.glmnet(data.x,data.y,
                          lambda=lambda.grid,
                          alpha=0)
plot(mod.ridge.cv)
```



Kinda nice plot. There is a clear minimum indicating penalized regression is warranted here. Use the One SE value and make predictions.

```
lambda.opt <- mod.ridge.cv$lambda.1se
test.x <- data.matrix(test.df[,1:numPred])
test.y <- data.matrix(test.df[,numPred+1])
preds.ridge <- predict(mod.ridge.cv,newx=test.x,s=lambda.opt)
(mse.ridge <- with(test.df,mean((solubility - preds.ridge)^2)))
```

```
## [1] 0.6788216
```

Compare all the MSE values.

```
c(mse.lm,mse.ridge,mse.bag)
```

```
## [1] 0.7313023 0.6788216 0.5684503
```

So far, Bagging is winning.

Random Forests

Random forests introduce a twist to the bagging process. Instead of using all the predictors at each branch, only use a random subset of a particular size.

Why? Using all the predictors favors the "heavy hitters" each time. Even with bootstrapping, the same small set of predictors tend to dominate the branching decisions, at least early on when a lot of the MSE is accounted for. This means the bootstrapped trees are not entirely independent of each other.

By limiting to a fraction of the predictors each time, more predictors get a chance to play a role in the tree construction, especially at the early stages. This leads to trees which are much more independent of each other and hence more randomization of the predictions.

The number of predictors at each branch is controled by the randomForest parameter "mtry". There is no "best" value for mtry, but it has been found that if numPred is the total number of predictors, numPred/3 or sqrt(numPred) work well.

Let's build a random forest with numPred/3 predictors at each step.

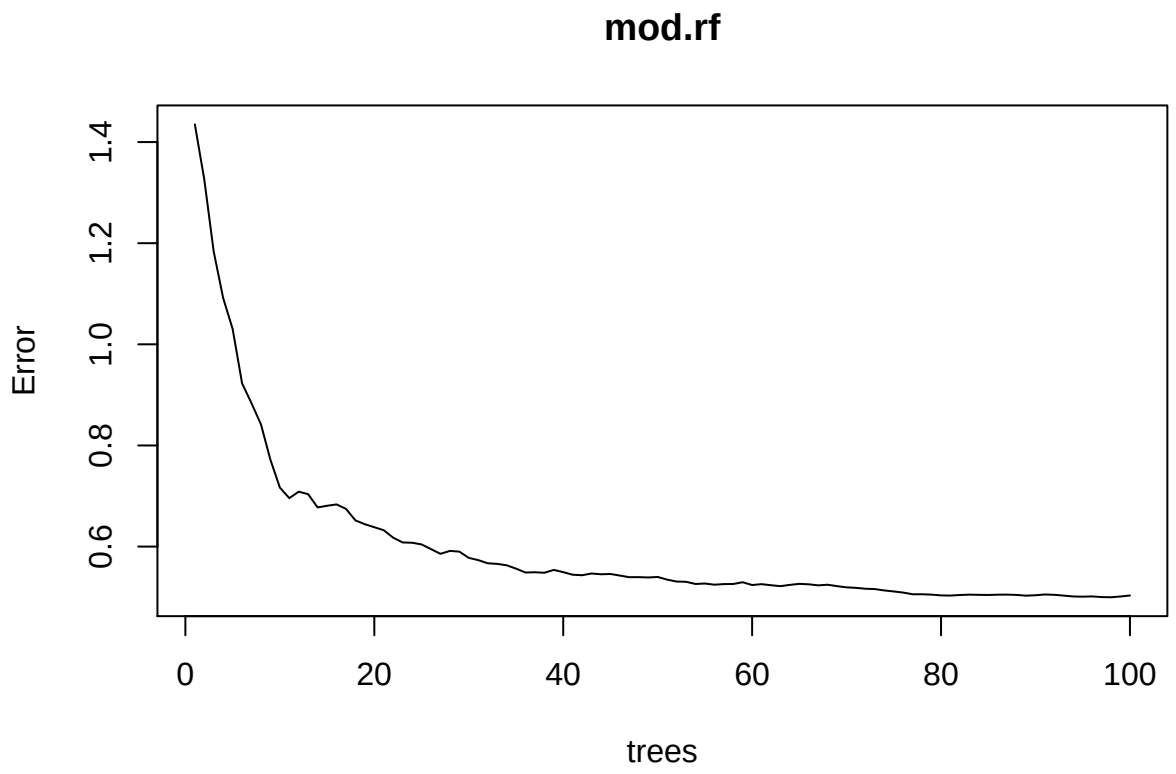
```
mod.rf <- randomForest(solubility ~ .,
                       data=data.df,
                       ntree=numTree,
                       ## Limit to 1/3 of predictors at each branch
                       mtry=(numPred)/3)
mod.rf
```

```
##
## Call:
##  randomForest(formula = solubility ~ ., data = data.df, ntree = numTree,      mtry = (numPred)/3)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 76
##
##           Mean of squared residuals: 0.5032514
##                     % Var explained: 87.2
```

Again, the "Mean of squared residuals" is the OOB estimate.

The plot is similar.

```
plot(mod.rf)
```



As before, we can make the predictions for each observation and compare with the other mse values.

```
preds.rf <- predict(mod.rf,newdata=test.df)
(mse.rf <- with(test.df,mean((solubility - preds.rf)^2)))
```

```
## [1] 0.5815482
```

How do the four methods stack up?

```
c(mse.lm,mse.ridge,mse.bag,mse.rf)
```

```
## [1] 0.7313023 0.6788216 0.5684503 0.5815482
```

It looks as if the bagging models outperform regression (simple or penalized) in this case.

Assignment

For the solubility data, compute the analog of Figure 17.2 of Compute Age Statistical Inference.

https://web.stanford.edu/~hastie/CASI_files/PDF/casi.pdf

Note: Replace Lasso (interactions) with Ridge Regression.

Do you reach the same conclusion, namely that bagging/random forests outperform the other models? # Predictor Importance As we build different bootstrapped versions of the data and hence different trees, the

predictors get used in different ways. This is especially true of a random forest when mtry is something like numPred/3.

We can measure the impact of a variable by keeping track of how much the MSE (or error rate, or binomial deviance, etc) changes each time that predictor is used. The total of all these changes in guage of the predictors "Importance".

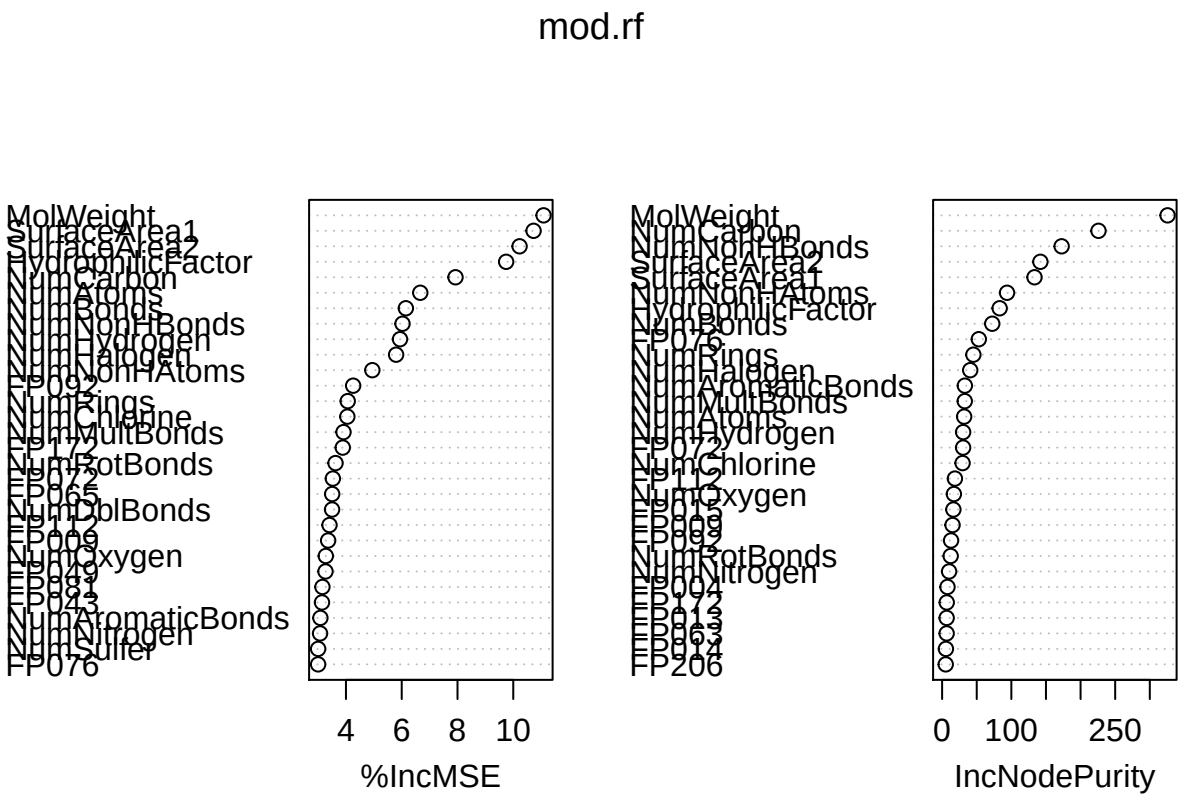
Importance variables are very helpful when it comes to trying to explain the results of a random forest model. We can't talk about 1000 or so trees, but we can talk about how various predictors impact the final predictions.

Fortunately, the R randomForest function keeps track of this information.

```
mod.rf <- randomForest(solubility ~ .,
                       data=data.df,
                       ntree=numTree,
                       ## Limit to 1/3 of predictors at each branch
                       mtry=(numPred)/3,
                       ## Include this flag
                       importance=TRUE)
```

Here's a bare-bones look at importance.

```
varImpPlot(mod.rf)
```



Importance can be viewed from both a MSE reduction perspective and Node Purity perspective. We can that the predictor "MolWeight" plays the large role.

The plot looks much better zoomed. ## Improved Importance Plot This plot is not much too look at. We can fix this up pretty easily and make it more presentable.

```
head(mod.rf$importance)
```

```
##           %IncMSE IncNodePurity
## FP001 0.001168138     0.3205095
## FP002 0.005639256     1.1309882
## FP003 0.004341482     1.6637049
## FP004 0.023638585     7.7397970
## FP005 0.002800707     0.6850547
## FP006 0.008581893     2.2938003
```

```
## grab the importance values
importVals <-mod.rf$importance # importance(mod.rf)
## Just use  MSE
importVals <- importVals[,1]
## and the names
varNames <- row.names(mod.rf$importance)
```

Pack all this into a nice data frame.

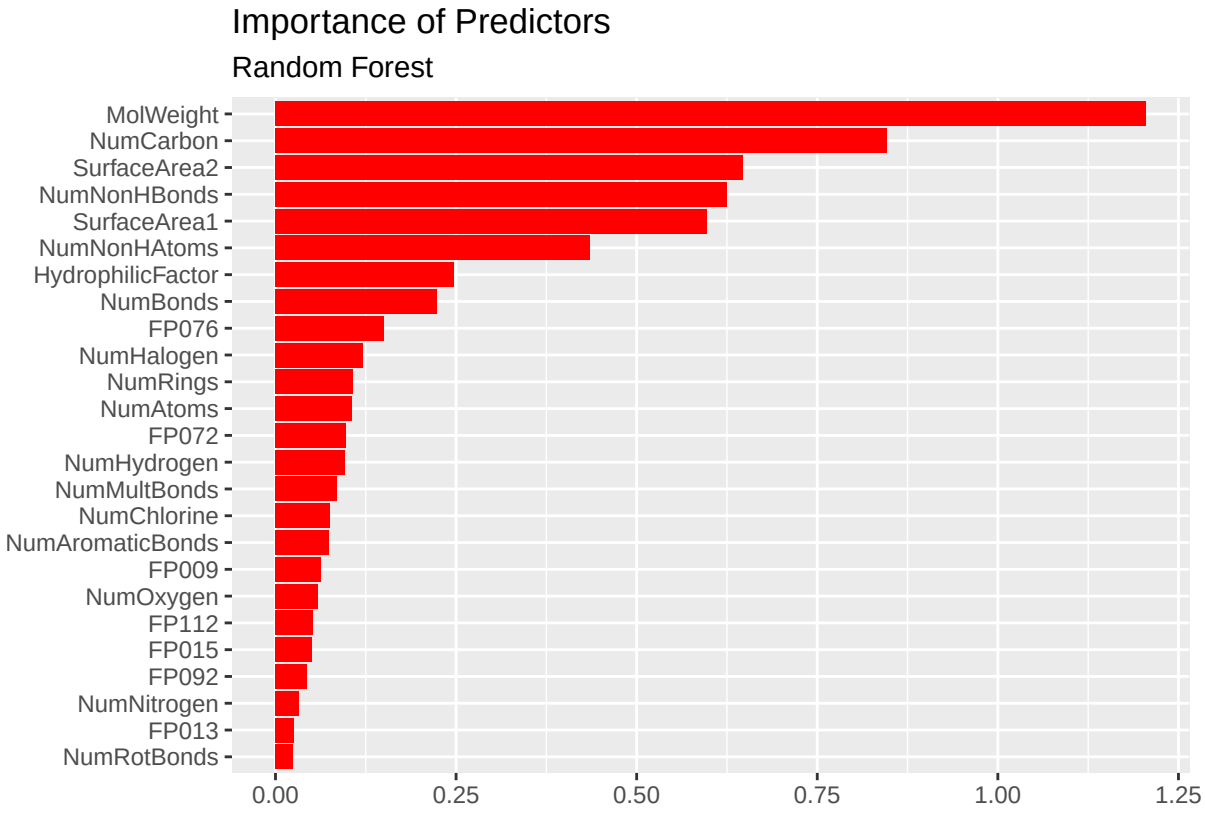
```
importanceRF.df <- data.frame(importance=importVals,
                      var=varNames)%>%
    arrange(desc(importance))
```

Fix the order of the var factor (otherwise R will want to put them in alphabetical order).

```
varNames <- importanceRF.df$var
importanceRF.df <- importanceRF.df%>%
    mutate(var=factor(var,levels=rev(varNames)))
```

And the Importance Plot...

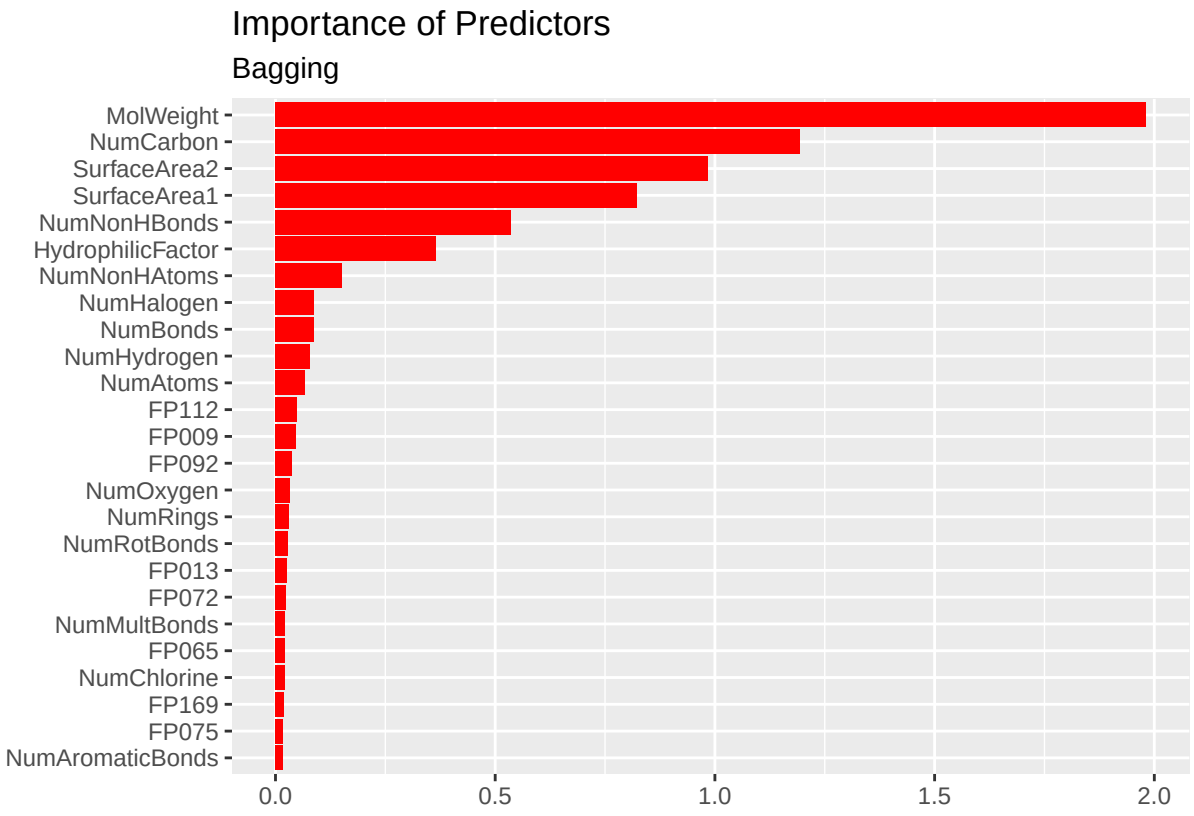
```
##only show the top 25
rf.gg <- importanceRF.df[1:25,] %>%
    ggplot()+
    geom_bar(aes(var,importance),stat="identity",fill="red")+
    coord_flip()+
    labs(title="Importance of Predictors",
         subtitle="Random Forest",
         x="",
         y="")
rf.gg
```



It's interesting to compare the importance plot to to that of a bagged model.

```
## Random Forests can supply imformation about the impact of each predictor
mod.bag <- randomForest(solubility ~ .,
                        data=data.df,
                        ntree=numTree,
                        ## BAGGING
                        mtry=(numPred),
                        ## Include this flag
                        importance=TRUE)
#### grab the importance values
importVals <-mod.bag$importance # importance(mod.rf)
## Just use  MSE
importVals <- importVals[,1]
## and the names
varNames <- row.names(mod.rf$importance)
importanceBag.df <- data.frame(importance=importVals,
                       var=varNames)%>%
    arrange(desc(importance))
##
varNames <- importanceBag.df$var
importanceBag.df <-importanceBag.df%>%
    mutate(var=factor(var,levels=rev(varNames)))

##only show the top 25
bag.gg <- importanceBag.df[1:25,] %>%
    ggplot()+
```

```
    geom_bar(aes(var,importance),stat="identity",fill="red")+
    coord_flip()+
    labs(title="Importance of Predictors",
         subtitle="Bagging",
         x="",
         y="")
bag.gg
```

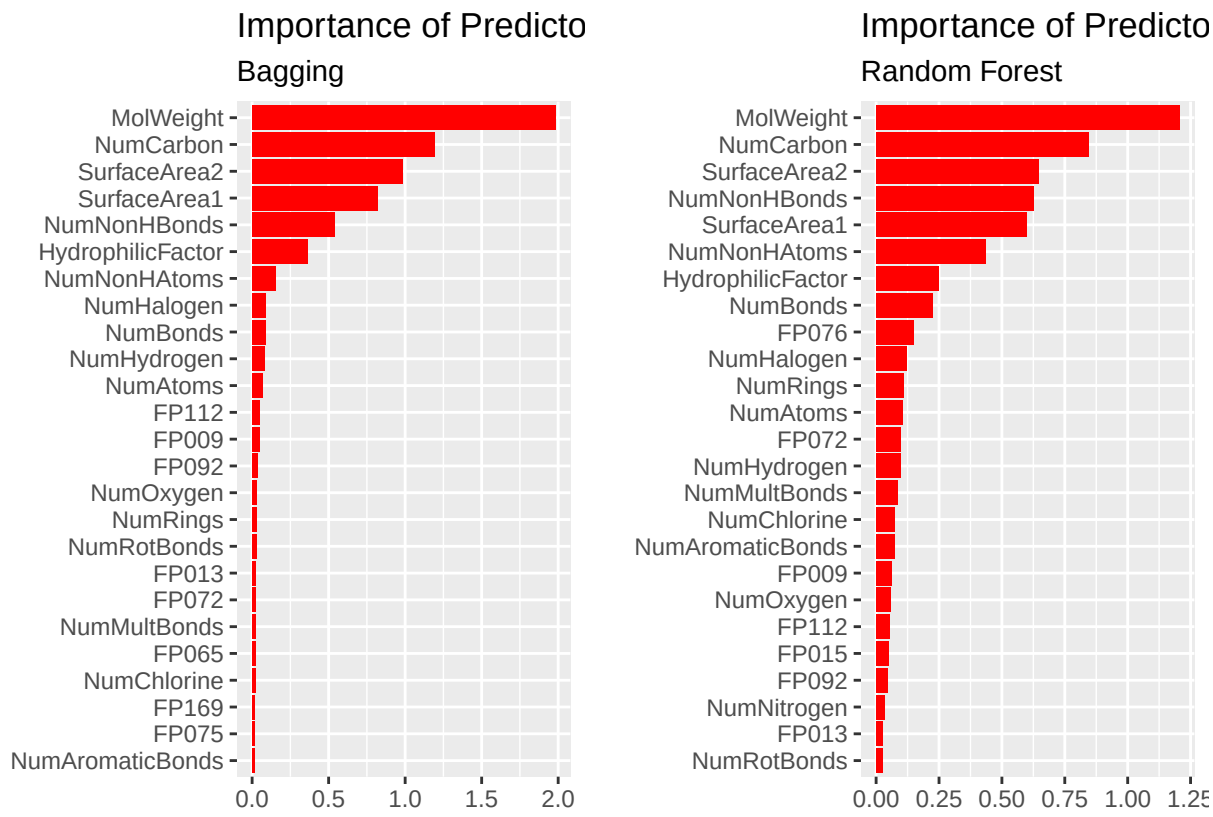


The two importance plots are similar, but there are some differences in the order of importance.

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:randomForest':
##
##     combine

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
grid.arrange(bag.gg,rf.gg,nrow=1)
```



Summary

Bagging, and Random Forests, in particular have proven to be very robust tools for prediction. In almost all cases, Random Forests are competitive with regression methods. Moreover, in many cases, Random Forests are demonstratively better methods. With Importance Plots, they also have a nice descriptive feature that makes them useful even for decriptive analysis.

Assignment

Use bagged trees and random forests to model the 2012 County Election Data we used before. In this case, the response variable (VoterProp) is quantitative. Which is better? Look at the importance plot. Do you note any connections between the important predictors from the tree perspective and the predictors that emerged from the forward/backward/lasso selection algorithms?

Replace the VoterProp variable with a binary variable, HighTurnOut, indicating whether VoterProp is greater than 0.5. Use bagged trees and random forests to predict HighTurnOut. Again, at connections between bagged trees and other other methods, respect to both performance and the identification of key variables.