title: "154 - Lecture 20 / Lab 3" author: "Gabe" date: "10/29/2020" output: pdf_document

```
require(randomForest)
require(e1071)
require(rpart)
```

Random Forests

The number of possible trees that could be produced is massive (on the first split there are $p(n-1)$ choices we consider). Thinking multiple steps ahead to lessen the issues of using a greedy algorithm will increase the computational complexity exponentially.

Instead, we grow many trees (which means we can parallelize it!). The space of possible trees is still too massive to really explore in any meaningful sense, and it's so big that taking "bad" moves sometimes is unlikely to be helpful (some "bad" moves lead to "great" partitions, but given the size of the space, most "bad" moves probably result in "bad" partitions). So, we'll still use a greedy algorithm and always make the "best" moves, we'll just limit the moves available.

So we will give it only a subset (*mtry*) of the predictor variables on which to split (different each iteration). The other tuning parameters are the maximum number of nodes (*maxnodes*), and the smallest allowable node size (*nodesize*), which are clearly related. They limit the size of the tree that is grown, which can control computation time. The defaults are no limit on the number of nodes, and the smallest node size is 1, i.e. grow the biggest trees possible (so that every terminal node consists of a single observation). This is not the obvious problem that it would be in CART, which would feel like a 1-NN classifier. Instead, predicting a new observation by pushing it down each tree in the ensemble will result in different partitions, that, while only consisting of a single observation, will often be a collection of different observations for each tree. But we can play around and try to optimize the choice of *nodesize* or *maxnodes* via cross-validation.

Additionally, we'll also give it a different data set each time, but a data set that *looks* like the real data. A bootstrap sample! So called *bagging*.

There's some fun stuff we'll get from this. A good estimate of misclassification rate (using majority vote) or sum of squared errors in the regression case is free because we have grown trees that haven't seen some of our data (*out-of-bag* observations). Built in cross-validation!

Additionally, since on every step we are doing a univariate analysis, we can discuss which variables seem to be useful.

Interpreting

We can rank the value of the variables in our data set in terms of the **variable importance**. There are a couple of ways we can do this. We can see how much Gini Impurity decreases on average when the variable is chosen to split on.

Alternatively, we can scramble the values of a variable, push every observation down the trees for which it is out-of-bag, and see how much worse classification (majority vote) gets. This is *accuracy-based importance*, while the average Gini decrease is what should be returned by *randomForest*.

A nice feature of random forests is that it is affine invariant, the results will not depend whether or not you scaled your data (as opposed to SVM, as the geometric margin of a data set depends on the choice of units).

Lab 3

Generate some data with $p = 2$ (not wheelhouse for RF but I want to visualize). Do so such that you can find the decision boundary coming from the Bayes classifier. Optimize your learners via cross-validation, and using the *plot.learner* function, visualize the resulting decision boundary and discuss.

```
plot.learner <- function(fit, x1LB, x1UB, x2LB, x2UB, data) {
  #Plots "preimage" of linear decision boundary generated in 3
space by added basis function
  #
  #Inputs
  #  fit: svm object
  #  x1LB, x1UB: lower and upper bounds for variable x1
  #  x2LB, x2UB: lower and upper bounds for variable x2
  #  data: data frame containing variables, need to have varia
bles named y, x1 and x2
  x1.grid <- seq(x1LB, x1UB, length.out=100)
  x2.grid <- seq(x2LB, x2UB, length.out=100)
  data.pred <- c()
  for (i in 1:length(x1.grid)) {
    for (j in 1:length(x2.grid)) {
      data.pred <- rbind(data.pred, c(x1.grid[i], x2.grid[j]))
    }
  }
  data.pred <- as.data.frame(data.pred)
  names(data.pred) <- c('x1', 'x2')

  plot(data$x1,data$x2,col=as.numeric(data$y)+8)
  points(data.pred[,1:2],col=as.numeric(predict(fit, newdata=d
ata.pred))+2, cex=.4, pch=19)
  points(data$x1,data$x2,col=as.numeric(data$y)+8, pch=19)
}

x1 <- runif(100)
x2 <- runif(100)
y <- 1 * (x1 < x2)
flip <- sample(100,10)
y[flip] <- -y[flip]+1  #add a little noise to the problem
y <- as.factor(y)
plot(x1,x2,col=as.numeric(y)+1, pch=19)
abline(0,1) #bayes classifier
```

plot.learner should work for randomForest and svm. rpart returns class proportions instead of classifications, and needs a hack:

```
plot.learner.rpart <- function(fit, x1LB, x1UB, x2LB, x2UB, da
ta) {
  #Plots preimage of linear decision boundary generated in 3 s
pace by added basis function
  #
  #Inputs
  #  fit: svm object
  #  x1LB, x1UB: lower and upper bounds for variable x1
  #  x2LB, x2UB: lower and upper bounds for variable x2
  #  data: data frame containing variables, need to have varia
bles named y, x1 and x2
  x1.grid <- seq(x1LB, x1UB, length.out=100)
  x2.grid <- seq(x2LB, x2UB, length.out=100)
  data.pred <- c()
  for (i in 1:length(x1.grid)) {
    for (j in 1:length(x2.grid)) {
      data.pred <- rbind(data.pred, c(x1.grid[i], x2.grid[j]))
    }
  }
  data.pred <- as.data.frame(data.pred)
  names(data.pred) <- c('x1', 'x2')

  plot(data$x1,data$x2,col=as.numeric(data$y)+8)
  points(data.pred[,1:2],col=round(predict(fit, newdata=data.p
red))[,1]+2, cex=.2, pch=19)
  points(data$x1,data$x2,col=as.numeric(data$y)+8, pch=19)
}

data.1 <- data.frame(y,x1,x2)
fit.rpart <- rpart(y~., data=data.1)
plot.learner.rpart(fit.rpart,0,1,0,1,data.1)
```

To appreciate variable importance, we can try to predict the transmission type of a car from the *mtcars* dataframe.

```
require(randomForest)
```

```
data(mtcars)
mtcars$am <- as.factor(mtcars$am)
fit <- randomForest(am~., data=mtcars)
fit
fit$importance
```

Have a look for the classification problem you explored in the most recent homework.