Homework 8

Instructions

Due: 1:35pm on Wednesday, November 18th

1. Add your name between the quotation marks on the author line in the YAML above.
2. Compose your answer to each problem between the bars of red stars.
3. Commit your changes frequently.
4. Be sure to knit your .Rmd to a .pdf file.
5. Push both the .pdf and the .Rmd file to your repo on the class organization before the deadline.

Theory

Problem 1

Based on ISLR Exercise 8.2 It is mentioned in Section 8.2.3 of ISLR that boosting using depth-one trees (or *stumps*) leads to an *additive* model: that is, a model of the form

$$f(X) = \sum_{j=1}^{p} f_j(X_i)$$

Verify that this is the case. Hint: Start with equation 8.12 in Algorithm 8.2.

Problem 2

Based on ISLR Exercise 8.5

Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each ootstrapped sample and, for a specific value of X produce 10 estimates of P(Class is red | X):

0.1, 0.14, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75

There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in Chapter 8 of ISLR. The second approach is the classify based on the average probability. In this example, what is the final classification under each of these two approaches?

Applied

Problem 3

One of the most useful applications to come out of classification models has been character (i.e. letter) recognition. In the following problem, you will build your own character recognition system using boosted trees.

The data

Our data set consists of a catalog of the features extracted from 20,000 images of letters. They can be loaded in with the following code.

```
lettersdf <- read.csv("data/lettersdf.csv", header = T)
```

(Data come from: P. W. Frey and D. J. Slate. "Letter Recognition Using Holland-style Adaptive Classifiers". (Machine Learning Vol 6 #2 March 91))

Initially, the each image was made up of 45 x 45 pixels, where each was characterized as either "on" or "off" (black or white). In order to extract more meaningful predictors from the data, resesearchers went through and performed *feature extraction*, collapsing those 2025 dimensions into 16, each of which is a summary statistic calculated on the image. They are as follows:

1. (The actual letter that the image corresponds to.)
2. The horizontal position, counting pixels from the left edge of the image, of the center of the smallest rectangular box that can be drawn with all "on" pixels inside the box.
3. The vertical position, counting pixels from the bottom, of the above box.
4. The width, in pixels, of the box.
5. The height, in pixels, of the box.
6. The total number of "on" pixels in the character image.
7. The mean horizontal position of all "on" pixels relative to the center of the box and divided by the width of the box. This feature has a negative value if the image is "left- heavy" as would be the case for the letter L.
8. The mean vertical position of all "on" pixels relative to the center of the box and divided by the height of the box.
9. The mean squared value of the horizontal pixel distances as measured in 6 above. This attribute will have a higher value for images whose pixels are more widely separated in the horizontal direction as would be the case for the letters W or M.
10. The mean squared value of the vertical pixel distances as measured in 7 above.
11. The mean product of the horizontal and vertical distances for each "on" pixel as measured in 6 and 7 above. This attribute has a positive value for diagonal lines that run from bottom left to top right and a negative value for diagonal lines from top left to bottom right.
12. The mean value of the squared horizontal distance times the vertical distance for each "on" pixel. This measures the correlation of the horizontal variance with the vertical position.
13. The mean value of the squared vertical distance times the horizontal distance for each "on" pixel. This measures the correlation of the vertical variance with the horizontal position.
14. The mean number of edges (an "on" pixel immediately to the right of either an "off" pixel or the image boundary) encountered when making systematic scans from left to right at all vertical positions within the box. This measure distinguishes between letters like "W" or "M" and letters like 'T' or "L."
15. The sum of the vertical positions of edges encountered as measured in 13 above. This feature will give a higher value if there are more edges at the top of the box, as in the letter "Y."
16. The mean number of edges (an "on" pixel immediately above either an "off" pixel or the image boundary) encountered when making systematic scans of the image from bottom to top over all horizontal positions within the box.
17. The sum of horizontal positions of edges encountered as measured in 15 above.

You will want to build your model on a training data set and evaluate its performance on a separate test data set. Use the following code to generate training-test split:

```
#Note: we can't use our typical sample_frac and anti_join
#method since some observations are duplicates

set.seed(1)
train_index <- sample(1:nrow(lettersdf), nrow(lettersdf) * .75)
test_index <- (1:nrow(lettersdf))[-(train_index)]
```

```
letters_trn<-lettersdf %>% slice(train_index)
letters_tst<-lettersdf %>% slice(test_index)
```

a. Contruct a boosted tree to predict the class of the training images (the letters) based on its 16 features. This can be done with the `gbm()` function in the library of the same name. Look to the end of chapter 8 for an example of the implementation. Note that we'll be performing a boosted *classification* tree. It's very similar to the boosted regression tree except the method of calculating a residual is adapted to the classification setting. Use as your model parameters \$B = 50\$, $\lambda = 0.1$, and $d = 1$. Note that this is computationally intensive, so it may take a minute to run. (Note: you may want to add `cache = T` to the chunk options of the code chunk you used to create the model, so that R doesn't need to run the code and build your model each time you knit your document).

b. Which variable was most important to your boosted tree?

c. Now use this boosted model to predict the classes of the images in the test data set. Use the same number of trees and be sure to add the argument `type = "response"`. The output of this will be a 5000 X 26 X 1 array: for each image you'll have a predicted probability that it is from each of the 26 classes. To extract the vector of length 5000 of each final predicted class, you can use the following function:

```
##apply the following function to the output of predict()

my_predictions <- function(x){ LETTERS[apply(x, 1, which.max)]}
```

d. Create a 26 x 26 confusion matrix for the predicted and actual letters. What is the misclassification rate? What letter was most difficult to predict? Are there any letter pairs that are particularly difficult to distinguish?

e. Build a second boosted tree model that uses even *slower* learners, that is, decrease $\lambda$and increase B somewhat to compensate (the slower the learner, the more of them we need). Pick the parameters of your choosing for this, but be wary of trying to fit a model with too high a B. You don't want to wait an hour for your model to fit.

f. How does the misclassification rate compare to the rate from you original model?

g. Are there any letter pairs that became particularly easier/more difficult to distinguish?

Problem 4

Return to the Communities and Crime data set from HW 7, which can be loaded with the following code:

```
crime_trn<-read_csv("data/crime.csv")
crime_tst<-read_csv("data/crime_tst.csv")
```

a. Fit a `randomForest()` model that performs only bagging and no actual random forests (recall that bagging is the special case of random forests with $m = p$). Next, fit a second random forest model that uses $m = p/3$. Compute their test MSEs. Is this an improvement over the vanilla pruned regression tree? Does it beat your regression model?

b. One thing we lose by using these computational techniques to limit the variance is the clearly interpretable tree diagram. We can still salvage some interpretability by considering `importance()`. Construct a Variable Importance Plot (`varImpPlot()`) for the bagged and random forest models. So as not to create an overwhelming plot, tweak the arguments to only plot the top ten variables for each. How do these results compare to the list of predictors you hypothesized would be important in HW 7?

c. Construct a model based on a boosted tree with parameters of your choosing. How does the test MSE compare to your existing models (Bagged Trees, Random Forests, Decision Tree from HW 7)?