Homework 9 Solutions

Instructions

Due: 5pm on Wednesday, December 1st

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.

The output of the boosted model is

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

where $\lambda\hat{f}^b$ is the prediction obtained from the bth step of the process. But since we assume that each iteration uses trees of depth 1, then each iterative $\hat{f}^b$ is of the form

$$\hat{f}^b(x) = \hat{\beta}_0^b + \hat{\beta}_1^b I(x_j < c_b)$$

for real numbers $\hat{\beta}_0^b, \hat{\beta}_1^b, c_b$ and predictor x_j; that is, for an observation x, $\hat{f}^b$ predicts the value $\hat{\beta}_0^b$ if $x_j \geq c_b$ and predicts the value $\hat{\beta}_0^b + \hat{\beta}_1^b$ if $x_j < c_b$.

Let $\hat{f}_1$ be the sum of all $\hat{f}^b$ that use x_1 to split, let $\hat{f}_2$ be the sum of all $\hat{f}^b$ that use x_2 to split, and so on. Note that each $\hat{f}_j$ is a function of a single predictor x_j. Then

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x) = \sum_{j=1}^{p} \lambda \hat{f}_j(x_j)$$

as desired.

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 bootstrapped 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?

If we classify based on the majority vote approach using the rule: Class is red if $P(\text{Class is red } |X) > 0.5$, then our classifications for each boostrap sample are $G, G, G, G, R, R, R, R, R$ and the majority class is **red**.

On the other hand, the average probability is 0.449, which gives a classification of **green**.

```
mean( c( 0.1, 0.14, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75))
```

```
## [1] 0.449
```

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 comes 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, researchers 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 rsample
#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. Construct 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 will 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 λ 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?

---

a. Note that running `distribution = "multinomial"` will output a warning message. We ignore it for this assignment.

```
set.seed(1)
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.2
```

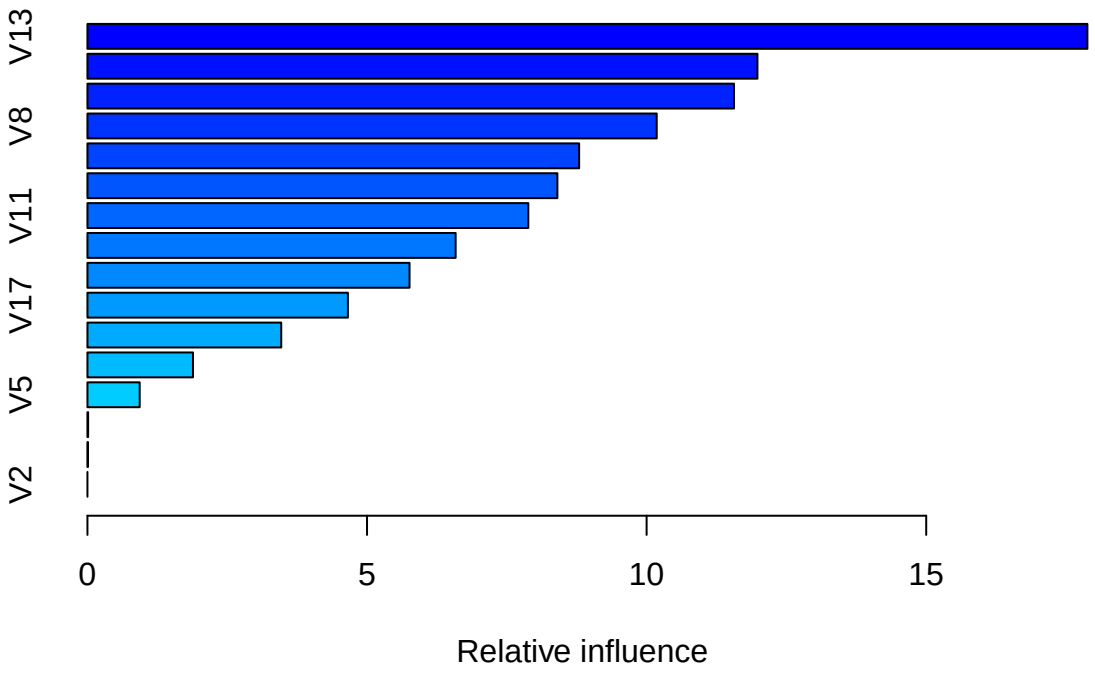
```
## Loaded gbm 2.1.8
```

```
boost_letters <-gbm(V1~., letters_trn, distribution = "multinomial", n.trees = 50, shrinkage = 0.1, int
```

```
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

b. The most important variable was V13 (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). The second most important variable was V12 (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.)

```
summary(boost_letters)
```



```
##     var      rel.inf
## V13 V13 17.882667759
## V12 V12 11.981626980
## V14 V14 11.562476322
## V8   V8 10.180351288
## V10 V10  8.793267664
## V15 V15  8.403978633
## V11 V11  7.884084718
## V9   V9  6.584732410
## V16 V16  5.760084531
## V17 V17  4.657783485
## V7   V7  3.465562273
## V4   V4  1.887169116
## V5   V5  0.934189754
## V6   V6  0.012524552
## V3   V3  0.009500514
## V2   V2  0.000000000
```

c.

```
preds_matrix<-predict(boost_letters, letters_tst, type = "response")
```

```
## Using 50 trees...
```

```
my_preds<-my_predictions(preds_matrix)
```

d. The overall misclassification rate was 0.314. In particular, the letters E, H and K were the most difficult to predict, with misclassification rate of 52.6%, 51.9%, and 51.5%.

Inspecting the confusion matrix, it looks like the letters most frequently confused for others were:

- E (mistaken for X 24 times)
- K (mistaken for R 23 times)
- B (mistaken for D 22 times)

However, it may be preferable to look at the conditional confusion matrix (i.e. for each letter, the proportion of times it was mislabled as another particular letter) since letters appear in the original data set with different frequencies. In this case, it turns out that the same story above holds, as the conditionally most frequently confused letters were:

- E (mistaken for X 14% of the time)
- K (mistaken for R 12% of the time)
- B (mistaken for D 12% of the time)

Finally, we should consider the combined rate two letters were mistaken for each other. For example, while E was mistaken for X 14% of the time, X was mistaken for E only 2% of the time. To do so, we compute the average conditional rate of mistakes for each letter pair:

- B and D were mixed up at average rate 0.09
- X and E were mixed up at average rate 0.08
- E and C were mixed up at average rate 0.08

It's fine if students only consider the first or second point when discussing letter pairs that were particularly difficult to distinguish

```r
#Confusion Matrix
letters_conf <- table(my_preds, letters_tst$V1)
 letters_conf
```

```
##
## my_preds   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q
##        A 158   0   0   0   0   0   0   0   1   0   0   7   1   0   0   0   1
##        B   0 134   2  13   6   7   3  15   7   8   5   5   1   3   1   7  12
##        C   1   0 117   0  16   0  12   0   2   4   7  13   2   2   0   0   5
##        D   1  22   1 137   0   7   5   8   6   6   2   0   0   6   7  10   0
##        E   0   0  11   1  83   0   1   0   0   1   8   2   0   0   0   0   7
##        F   0   1   0   0   0 137   0   0   1   5   0   0   0   0   0  13   0
##        G   1   1   8   0  13   3 121   3   0   0   5   6   0   0   7   7  10
##        H   0   0   0   0   0   1   0 102   0   1   3   0   1   0   0   1   0
##        I   0   0   0   0   0   4   0   0 158   1   0   1   0   0   0   0   0
##        J   0   0   0   2   0   0   1   1   5 126   0   0   0   4   0   2   0
##        K   0   0  16   0   5   0   5   5   0   0  94   1   5   4   0   0   1
##        L   1   0   0   1   0   0   0   0   2   0   0 132   0   0   0   0  15
##        M   3   7   0   2   0   1   1   1   0   2   9   0 175   4   1   1   5
##        N   1   0   3   6   1   3   0  14   0   0  11   0  11 149   6   0   0
##        O   6   1   9   7   0   0   2  11   0   4   0   0   3   3 149  10  18
##        P   0   0   0  11   0  10   2   0   3   0   0   0   0   9   1 143   0
##        Q   1   0   3   0   7   1   9   5   2   3   3   0   0   2   7   4 124
##        R   1  10   0   5   2   2  11  21   3   4  23   1   0   5   3   0   0
##        S   4   7   1   1   8   3   3   2   4   5   1   3   1   0   0   0   2
##        T   0   0   0   2   0  11   0   1   0   0   0   0   0   0   0   1   0
##        U   0   0   4   0   0   0   0   9   0   0   0   0   2   1   3   0   1
##        V   0   0   0   0   0   0   1   0   0   0   0   0   0   1   0   0   3
##        W   1   3   3   0   2   0  10   8   0   0   2   0   9   5  12   7   2
##        X   9   2   0   0  24   6   0   3   1   0  15   4   0   0   0   0   1
##        Y   2   3   0   0   2   6   0   2   0   0   5   7   0   8   0   4   0
##        Z   0   0   2   0   6   1   3   1   0   0   1   0   0   0   0   0   1
##
## my_preds   R   S   T   U   V   W   X   Y   Z
##        A   1  11   0   0   0   0   0   0   1
##        B   7  10   1   1   2   0  10   0   6
##        C   0   0   0   2   0   0   0   0   0
##        D   7   2   0   1   0   0   3   5   1
##        E   6   2   9   2   1   0   4   0   7
##        F   0   1  14   0   2   1   0   1   0
##        G   0   2   0   3   2   1   0   0   0
##        H   1   0   0   1   1   0  16   0   0
##        I   0   6   3   0   0   0   1   0   0
##        J   0   1   0   0   0   0   0   0   4
##        K   2   1   1   1   0   0   6   0   0
##        L   0   3   0   0   0   0   0   1   2
##        M   6   0   1  10   0  13   0   2   0
##        N   4   0   3  15   4  15   0   0   0
##        O   3   3   2   7   0   2   2   3   0
##        P   0   0   3   0   3   0   0   1   0
##        Q   1   3   0   8   0   2   0   6   0
##        R 153   3   0   0   2   1   3   0   2
##        S   0  93   1   0   1   0   7   6  13
##        T   0   2 123   5   0   0   0   8   0
##        U   0   0   3 144   4   0   0   3   0
##        V   0   0  13   3 161   1   0  11   0
##        W   2   0   0   4  17 137   0   4   0
##        X   4  15   5   0   0   0 121   1   2
##        Y   1   4  15   4   5   0   9 133   0
```

```
##        Z   0   8   3   1   0   0   9   0 126
```

```
#Overall Misclassification rate
letters_error <- (sum(letters_conf) - sum(diag(letters_conf)))/sum(letters_conf)
letters_error
```

```
## [1] 0.314
```

```
## Misclassification by letter
misclass<-c()
for (i in 1:26){
  n <- sum(letters_conf[,i])
  misclass[i]<-(n - letters_conf[i,i])/n
}

data.frame(LETTERS, misclass) %>% arrange(desc(misclass))
```

```
##    LETTERS  misclass
## 1        E 0.5257143
## 2        H 0.5188679
## 3        K 0.5154639
## 4        S 0.4529412
## 5        Q 0.4038462
## 6        T 0.3850000
## 7        X 0.3664921
## 8        G 0.3631579
## 9        C 0.3500000
## 10       F 0.3251232
## 11       U 0.3207547
## 12       P 0.3190476
## 13       B 0.2984293
## 14       Y 0.2810811
## 15       N 0.2766990
## 16       L 0.2747253
## 17       D 0.2712766
## 18       J 0.2588235
## 19       O 0.2436548
## 20       Z 0.2317073
## 21       R 0.2272727
## 22       V 0.2146341
## 23       W 0.2080925
## 24       I 0.1897436
## 25       M 0.1706161
## 26       A 0.1684211
```

```
#mixup rate
letters_conf_prop<-letters_conf %>% as.data.frame.matrix() %>%  mutate_all( funs(./sum(.))) %>% as.matrix()
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

```r
diag(letters_conf_prop)<-rep(0, times = 26)

letters_conf_prop %>% round(digits = 2)
```

```
##      A    B    C    D    E    F    G    H    I    J    K    L    M    N    O
## A 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.04 0.00 0.00 0.00
## B 0.00 0.00 0.01 0.07 0.03 0.03 0.02 0.07 0.04 0.05 0.03 0.03 0.00 0.01 0.01
## C 0.01 0.00 0.00 0.00 0.09 0.00 0.06 0.00 0.01 0.02 0.04 0.07 0.01 0.01 0.00
## D 0.01 0.12 0.01 0.00 0.00 0.03 0.03 0.04 0.03 0.04 0.01 0.00 0.00 0.03 0.04
## E 0.00 0.00 0.06 0.01 0.00 0.00 0.01 0.00 0.00 0.01 0.04 0.01 0.00 0.00 0.00
## F 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.03 0.00 0.00 0.00 0.00 0.00
## G 0.01 0.01 0.04 0.00 0.07 0.01 0.00 0.01 0.00 0.00 0.03 0.03 0.00 0.00 0.04
## H 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.02 0.00 0.00 0.00 0.00
## I 0.00 0.00 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.01 0.00 0.01 0.00 0.00 0.00
## J 0.00 0.00 0.00 0.01 0.00 0.00 0.01 0.00 0.03 0.00 0.00 0.00 0.00 0.02 0.00
## K 0.00 0.00 0.09 0.00 0.03 0.00 0.03 0.02 0.00 0.00 0.00 0.01 0.02 0.02 0.00
## L 0.01 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00
## M 0.02 0.04 0.00 0.01 0.00 0.00 0.01 0.00 0.00 0.01 0.05 0.00 0.00 0.02 0.01
## N 0.01 0.00 0.02 0.03 0.01 0.01 0.00 0.07 0.00 0.00 0.06 0.00 0.05 0.00 0.03
## O 0.03 0.01 0.05 0.04 0.00 0.00 0.01 0.05 0.00 0.02 0.00 0.00 0.01 0.01 0.00
## P 0.00 0.00 0.00 0.06 0.00 0.05 0.01 0.00 0.02 0.00 0.00 0.00 0.00 0.04 0.01
## Q 0.01 0.00 0.02 0.00 0.04 0.00 0.05 0.02 0.01 0.02 0.02 0.00 0.00 0.01 0.04
## R 0.01 0.05 0.00 0.03 0.01 0.01 0.06 0.10 0.02 0.02 0.12 0.01 0.00 0.02 0.02
## S 0.02 0.04 0.01 0.01 0.05 0.01 0.02 0.01 0.02 0.03 0.01 0.02 0.00 0.00 0.00
## T 0.00 0.00 0.00 0.01 0.00 0.05 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
## U 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.00 0.01 0.00 0.02
## V 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
## W 0.01 0.02 0.02 0.00 0.01 0.00 0.05 0.04 0.00 0.00 0.01 0.00 0.04 0.02 0.06
## X 0.05 0.01 0.00 0.00 0.14 0.03 0.00 0.01 0.01 0.00 0.08 0.02 0.00 0.00 0.00
## Y 0.01 0.02 0.00 0.00 0.01 0.03 0.00 0.01 0.00 0.00 0.03 0.04 0.00 0.04 0.00
## Z 0.00 0.00 0.01 0.00 0.03 0.00 0.02 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00
##      P    Q    R    S    T    U    V    W    X    Y    Z
## A 0.00 0.00 0.01 0.06 0.00 0.00 0.00 0.00 0.00 0.00 0.01
## B 0.03 0.06 0.04 0.06 0.00 0.00 0.01 0.00 0.05 0.00 0.04
## C 0.00 0.02 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00
## D 0.05 0.00 0.04 0.01 0.00 0.00 0.00 0.00 0.02 0.03 0.01
## E 0.00 0.03 0.03 0.01 0.04 0.01 0.00 0.00 0.02 0.00 0.04
## F 0.06 0.00 0.00 0.01 0.07 0.00 0.01 0.01 0.00 0.01 0.00
## G 0.03 0.05 0.00 0.01 0.00 0.01 0.01 0.01 0.00 0.00 0.00
## H 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.08 0.00 0.00
## I 0.00 0.00 0.00 0.04 0.02 0.00 0.00 0.00 0.01 0.00 0.00
## J 0.01 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.02
## K 0.00 0.00 0.01 0.01 0.00 0.00 0.00 0.00 0.03 0.00 0.00
## L 0.00 0.07 0.00 0.02 0.00 0.00 0.00 0.00 0.00 0.01 0.01
## M 0.00 0.02 0.03 0.00 0.00 0.05 0.00 0.08 0.00 0.01 0.00
## N 0.00 0.00 0.02 0.00 0.02 0.07 0.02 0.09 0.00 0.00 0.00
## O 0.05 0.09 0.02 0.02 0.01 0.03 0.00 0.01 0.01 0.02 0.00
## P 0.00 0.00 0.00 0.00 0.02 0.00 0.01 0.00 0.00 0.01 0.00
## Q 0.02 0.00 0.01 0.02 0.00 0.04 0.00 0.01 0.00 0.03 0.00
## R 0.00 0.00 0.00 0.02 0.00 0.00 0.01 0.01 0.02 0.00 0.01
## S 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.04 0.03 0.08
## T 0.00 0.00 0.00 0.01 0.00 0.02 0.00 0.00 0.00 0.04 0.00
## U 0.00 0.00 0.00 0.00 0.02 0.00 0.02 0.00 0.00 0.02 0.00
## V 0.00 0.01 0.00 0.00 0.06 0.01 0.00 0.01 0.00 0.06 0.00
## W 0.03 0.01 0.01 0.00 0.00 0.02 0.08 0.00 0.00 0.02 0.00
## X 0.00 0.00 0.02 0.09 0.02 0.00 0.00 0.00 0.00 0.01 0.01
## Y 0.02 0.00 0.01 0.02 0.08 0.02 0.02 0.00 0.05 0.00 0.00
## Z 0.00 0.00 0.00 0.05 0.02 0.00 0.00 0.00 0.05 0.00 0.00
```

```r
#highest
which(letters_conf_prop == max(letters_conf_prop), arr.ind=TRUE)
```

```
##   row col
## X  24   5
```

```r
max(letters_conf_prop)
```

```
## [1] 0.1371429
```

```r
#second highest
which(letters_conf_prop == max( letters_conf_prop[letters_conf_prop!=max(letters_conf_prop)] ), arr.ind=TRUE)
```

```
##   row col
## R  18  11
```

```r
max( letters_conf_prop[letters_conf_prop!=max(letters_conf_prop)] )
```

```
## [1] 0.1185567
```

```r
mixup_mat <- ( letters_conf_prop + t(letters_conf_prop) )/2
mixup_mat %>% round(digits = 2)
```

```
##      A    B    C    D    E    F    G    H    I    J    K    L    M    N    O
## A 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.02 0.01 0.00 0.02
## B 0.00 0.00 0.01 0.09 0.02 0.02 0.01 0.04 0.02 0.02 0.01 0.01 0.02 0.01 0.01
## C 0.00 0.01 0.00 0.00 0.08 0.00 0.05 0.00 0.01 0.01 0.06 0.04 0.00 0.01 0.02
## D 0.00 0.09 0.00 0.00 0.00 0.02 0.01 0.02 0.02 0.02 0.01 0.00 0.01 0.03 0.04
## E 0.00 0.02 0.08 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.03 0.01 0.00 0.00 0.00
## F 0.00 0.02 0.00 0.02 0.00 0.00 0.01 0.00 0.01 0.01 0.00 0.00 0.00 0.01 0.00
## G 0.00 0.01 0.05 0.01 0.04 0.01 0.00 0.01 0.00 0.00 0.03 0.02 0.00 0.00 0.02
## H 0.00 0.04 0.00 0.02 0.00 0.00 0.01 0.00 0.00 0.01 0.02 0.00 0.00 0.03 0.03
## I 0.00 0.02 0.01 0.02 0.00 0.01 0.00 0.00 0.00 0.02 0.00 0.01 0.00 0.00 0.00
## J 0.00 0.02 0.01 0.02 0.00 0.01 0.00 0.01 0.02 0.00 0.00 0.00 0.01 0.01 0.01
## K 0.00 0.01 0.06 0.01 0.03 0.00 0.03 0.02 0.00 0.00 0.00 0.00 0.04 0.04 0.00
## L 0.02 0.01 0.04 0.00 0.01 0.00 0.02 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00
## M 0.01 0.02 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.01 0.04 0.00 0.00 0.04 0.01
## N 0.00 0.01 0.01 0.03 0.00 0.01 0.00 0.03 0.00 0.01 0.04 0.00 0.04 0.00 0.02
## O 0.02 0.01 0.02 0.04 0.00 0.00 0.02 0.03 0.00 0.01 0.00 0.00 0.01 0.02 0.00
## P 0.00 0.02 0.00 0.05 0.00 0.06 0.02 0.00 0.01 0.00 0.00 0.00 0.00 0.02 0.03
## Q 0.01 0.03 0.02 0.00 0.04 0.00 0.05 0.01 0.01 0.01 0.01 0.04 0.01 0.00 0.06
## R 0.01 0.04 0.00 0.03 0.02 0.00 0.03 0.05 0.01 0.01 0.06 0.00 0.02 0.02 0.02
## S 0.04 0.05 0.00 0.01 0.03 0.01 0.01 0.00 0.03 0.02 0.01 0.02 0.00 0.00 0.01
## T 0.00 0.00 0.00 0.01 0.02 0.06 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.00
## U 0.00 0.00 0.02 0.00 0.00 0.00 0.01 0.02 0.00 0.00 0.00 0.00 0.03 0.04 0.02
## V 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.00
## W 0.00 0.01 0.01 0.00 0.01 0.00 0.03 0.02 0.00 0.00 0.01 0.00 0.06 0.06 0.04
## X 0.02 0.03 0.00 0.01 0.08 0.01 0.00 0.05 0.01 0.00 0.05 0.01 0.00 0.00 0.01
## Y 0.01 0.01 0.00 0.01 0.01 0.02 0.00 0.00 0.00 0.00 0.01 0.02 0.01 0.02 0.01
## Z 0.00 0.02 0.01 0.00 0.04 0.00 0.01 0.00 0.00 0.01 0.00 0.01 0.00 0.00 0.00
##      P    Q    R    S    T    U    V    W    X    Y    Z
## A 0.00 0.01 0.01 0.04 0.00 0.00 0.00 0.00 0.02 0.01 0.00
## B 0.02 0.03 0.04 0.05 0.00 0.00 0.00 0.01 0.03 0.01 0.02
## C 0.00 0.02 0.00 0.00 0.00 0.02 0.00 0.01 0.00 0.00 0.01
## D 0.05 0.00 0.03 0.01 0.01 0.00 0.00 0.00 0.01 0.01 0.00
## E 0.00 0.04 0.02 0.03 0.02 0.00 0.00 0.01 0.08 0.01 0.04
## F 0.06 0.00 0.00 0.01 0.06 0.00 0.00 0.00 0.01 0.02 0.00
## G 0.02 0.05 0.03 0.01 0.00 0.01 0.01 0.03 0.00 0.00 0.01
## H 0.00 0.01 0.05 0.00 0.00 0.02 0.00 0.02 0.05 0.00 0.00
## I 0.01 0.01 0.01 0.03 0.01 0.00 0.00 0.00 0.01 0.00 0.00
## J 0.00 0.01 0.01 0.02 0.00 0.00 0.00 0.00 0.00 0.00 0.01
```

```
## K 0.00 0.01 0.06 0.01 0.00 0.00 0.00 0.01 0.05 0.01 0.00
## L 0.00 0.04 0.00 0.02 0.00 0.00 0.00 0.00 0.01 0.02 0.01
## M 0.00 0.01 0.02 0.00 0.00 0.03 0.00 0.06 0.00 0.01 0.00
## N 0.02 0.00 0.02 0.00 0.01 0.04 0.01 0.06 0.00 0.02 0.00
## O 0.03 0.06 0.02 0.01 0.00 0.02 0.00 0.04 0.01 0.01 0.00
## P 0.00 0.01 0.00 0.00 0.01 0.00 0.01 0.02 0.00 0.01 0.00
## Q 0.01 0.00 0.00 0.01 0.00 0.02 0.01 0.01 0.00 0.02 0.00
## R 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.01 0.02 0.00 0.01
## S 0.00 0.01 0.01 0.00 0.01 0.00 0.00 0.00 0.06 0.03 0.06
## T 0.01 0.00 0.00 0.01 0.00 0.02 0.03 0.00 0.01 0.06 0.01
## U 0.00 0.02 0.00 0.00 0.02 0.00 0.02 0.01 0.00 0.02 0.00
## V 0.01 0.01 0.00 0.00 0.03 0.02 0.00 0.04 0.00 0.04 0.00
## W 0.02 0.01 0.01 0.00 0.00 0.01 0.04 0.00 0.00 0.01 0.00
## X 0.00 0.00 0.02 0.06 0.01 0.00 0.00 0.00 0.00 0.03 0.03
## Y 0.01 0.02 0.00 0.03 0.06 0.02 0.04 0.01 0.03 0.00 0.00
## Z 0.00 0.00 0.01 0.06 0.01 0.00 0.00 0.00 0.03 0.00 0.00
```

```
#highest rate
which(mixup_mat == max(mixup_mat), arr.ind=TRUE)
```

```
##   row col
## D   4   2
## B   2   4
```

```
max(mixup_mat)
```

```
## [1] 0.09216609
```

```
#second highest rate
which(mixup_mat == max( mixup_mat[mixup_mat!=max(mixup_mat)] ), arr.ind=TRUE)
```

```
##   row col
## X  24   5
## E   5  24
```

```
max( mixup_mat[mixup_mat!=max(mixup_mat)] )
```

```
## [1] 0.07904263
```

e. We reduce our learning rate by a factor of 2 to 0.05 and increase our number of trees by a factor of 4 to 200.

```
boost_letters_slow <-gbm(V1~., letters_trn,
                         distribution = "multinomial",
                         n.trees = 200,
                         shrinkage = 0.05,
                         interaction.depth = 1)
```

```
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

f. In changing the learning rate, we reduced our overall misclassification rate from 0.314 to 0.252.

g. Moreover, the 3 most misclassified letters were S, K, and H, with misclassification rates of 41.8%, 38.7%, and 38.2%. The formerly most misclassified letter (E) had misclassification rate reduced from 52.6% to 36.0%.

Looking at the conditional confusion matrix, we see that

- H was mistaken for R 9% of the time
- E was mistaken for X 9% of the time
- X was mistaken for O 9% of the time (this was new!)

The letters that were most frequently mixed up were:

- E and C were mixed up at average rate of 0.06
- Z and S were mixed up at average rate of 0.06

```
preds_matrix_slow<-predict(boost_letters_slow, letters_tst, type = "response")
```

```
## Using 200 trees...
```

```
my_preds_slow<-my_predictions(preds_matrix_slow)
```

```
#Confusion Matrix
letters_conf_slow <- table(my_preds_slow, letters_tst$V1)
 letters_conf_slow
```

```
##
## my_preds_slow   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P
##             A 161   0   0   1   0   0   0   1   0   0   0   1   1   0   0   0
##             B   0 144   3   9   2   7   1  12   6   7   4   4   1   3   0   5
##             C   0   0 127   0  10   0  11   0   2   0   2   5   0   0   1   0
##             D   0  10   0 146   0   5   5   7   4   5   2   0   0   7   7   6
##             E   0   0   9   1 113   0   0   0   0   0   5   4   0   0   0   4
##             F   0   3   0   1   0 139   0   1   2   6   0   0   0   0   0  11
##             G   1   1   6   0   5   3 129   3   0   0   3   4   1   0   1   6
##             H   0   1   0   3   0   1   1 127   1   1   6   0   1   1   1   0
##             I   0   0   0   1   0   8   0   0 161   1   0   1   0   0   0   0
##             J   0   0   0   1   0   0   1   1   6 134   1   0   0   3   0   0
##             K   1   1  10   1   4   0   1   5   0   0 117   1   4   2   0   2
##             L   0   0   0   1   0   0   0   0   3   0   0 150   0   0   0   0
##             M   3   3   0   0   0   1   1   1   0   0   5   0 186   2   1   0
##             N   2   2   3   4   0   3   1   5   0   0   9   0   5 157   4   0
##             O   3   0  10   5   0   0   1  10   0   4   0   0   1   3 159   3
##             P   0   1   0   5   0   9   3   2   3   0   0   0   0   7   1 160
##             Q   4   0   2   0   5   0  15   3   2   3   1   1   0   0   6   3
##             R   1  10   0   3   1   2   6  20   1   2  17   1   2   7   4   1
##             S   3   6   1   1  10   4   3   1   1   4   1   2   1   1   1   0
##             T   0   0   1   5   1  10   0   0   0   0   0   0   0   0   0   1
##             U   0   0   3   0   0   1   0   6   0   0   0   0   0   2   0   0
##             V   0   2   0   0   0   0   1   1   0   0   0   0   0   1   0   1
##             W   0   1   3   0   1   0   9   3   0   0   2   0   8   2  11   3
##             X   9   3   0   0  18   5   0   2   3   3  14   2   0   0   0   0
##             Y   1   3   0   0   1   4   0   1   0   0   4   6   0   8   0   4
##             Z   1   0   2   0   4   1   1   0   0   0   1   0   0   0   0   0
##
## my_preds_slow   Q   R   S   T   U   V   W   X   Y   Z
##             A   2   1  12   0   0   0   0   1   0   0
##             B   7   6  10   1   1   2   1   7   0   4
##             C   3   0   0   0   4   0   0   0   0   0
##             D   0  10   1   0   1   0   0   0   3   1
##             E   6   5   5  10   1   1   0   0   1   8
##             F   0   0   0  10   0   0   1   0   1   1
##             G   6   0   3   0   2   2   1   0   0   0
##             H   0   1   0   0   1   3   0   1   0   0
##             I   0   0   6   0   0   0   0   1   0   0
##             J   0   1   1   0   0   0   0   0   0   3
##             K   1   3   1   1   1   0   0   8   0   0
##             L  17   0   2   0   0   0   0   0   0   2
##             M   2   2   0   1   8   0  13   0   0   0
##             N   0   4   0   0  12   5   7   0   0   0
##             O  13   2   0   2   7   0   2  17   4   0
##             P   0   0   0   1   0   6   0   0   1   0
##             Q 138   0   5   0   7   1   3   0   4   2
```

```
##             R   0 157   3   0   0   1   1   2   0   1
##             S   4   1  98   1   0   1   0   7   5  10
##             T   0   0   2 142   1   0   0   0   3   0
##             U   1   0   1   5 152   3   0   0   5   0
##             V   2   0   1  10   5 170   1   1  19   0
##             W   0   0   0   0   5   9 143   0   3   0
##             X   1   4   5   6   0   0   0 138   1   0
##             Y   1   1   4   7   4   1   0   8 135   0
##             Z   4   0  10   3   0   0   0   0   0 132
```

```
#Overall Misclassification rate
letters_error_slow <- (sum(letters_conf_slow) - sum(diag(letters_conf_slow)))/sum(letters_conf_slow)
letters_error_slow
```

```
## [1] 0.257
```

```
## Misclassification by letter
misclass_slow<-c()
for (i in 1:26){
  n <- sum(letters_conf_slow[,i])
  misclass_slow[i]<-(n - letters_conf_slow[i,i])/n
}

data.frame(LETTERS, misclass_slow) %>% arrange(desc(misclass_slow))
```

```
##    LETTERS misclass_slow
## 1        S     0.4235294
## 2        H     0.4009434
## 3        K     0.3969072
## 4        E     0.3542857
## 5        Q     0.3365385
## 6        G     0.3210526
## 7        F     0.3152709
## 8        C     0.2944444
## 9        T     0.2900000
## 10       U     0.2830189
## 11       X     0.2774869
## 12       Y     0.2702703
## 13       B     0.2460733
## 14       P     0.2380952
## 15       N     0.2378641
## 16       D     0.2234043
## 17       J     0.2117647
## 18       R     0.2070707
## 19       Z     0.1951220
## 20       O     0.1928934
## 21       L     0.1758242
## 22       I     0.1743590
## 23       W     0.1734104
## 24       V     0.1707317
## 25       A     0.1526316
## 26       M     0.1184834
```

```
#mixup rate
letters_conf_prop_slow<-letters_conf_slow %>% as.data.frame.matrix() %>%
  mutate_all( funs(./sum(.))) %>% as.matrix()

diag(letters_conf_prop_slow)<-rep(0, times = 26)

letters_conf_prop_slow %>% round(digits = 2)
```

```
##      A    B    C    D    E    F    G    H    I    J    K    L    M    N    O
## A 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00
## B 0.00 0.00 0.02 0.05 0.01 0.03 0.01 0.06 0.03 0.04 0.02 0.02 0.00 0.01 0.00
## C 0.00 0.00 0.00 0.00 0.06 0.00 0.06 0.00 0.01 0.00 0.01 0.03 0.00 0.00 0.01
## D 0.00 0.05 0.00 0.00 0.00 0.02 0.03 0.03 0.02 0.03 0.01 0.00 0.00 0.03 0.04
## E 0.00 0.00 0.05 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.03 0.02 0.00 0.00 0.00
## F 0.00 0.02 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.04 0.00 0.00 0.00 0.00 0.00
## G 0.01 0.01 0.03 0.00 0.03 0.01 0.00 0.01 0.00 0.00 0.02 0.02 0.00 0.00 0.01
## H 0.00 0.01 0.00 0.02 0.00 0.00 0.01 0.00 0.01 0.01 0.03 0.00 0.00 0.00 0.01
## I 0.00 0.00 0.00 0.01 0.00 0.04 0.00 0.00 0.00 0.01 0.00 0.01 0.00 0.00 0.00
## J 0.00 0.00 0.00 0.01 0.00 0.00 0.01 0.00 0.03 0.00 0.01 0.00 0.00 0.01 0.00
## K 0.01 0.01 0.06 0.01 0.02 0.00 0.01 0.02 0.00 0.00 0.00 0.01 0.02 0.01 0.00
## L 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.00 0.00
## M 0.02 0.02 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.03 0.00 0.00 0.01 0.01
## N 0.01 0.01 0.02 0.02 0.00 0.01 0.01 0.02 0.00 0.00 0.05 0.00 0.02 0.00 0.02
## O 0.02 0.00 0.06 0.03 0.00 0.00 0.01 0.05 0.00 0.02 0.00 0.00 0.00 0.01 0.00
## P 0.00 0.01 0.00 0.03 0.00 0.04 0.02 0.01 0.02 0.00 0.00 0.00 0.00 0.03 0.01
## Q 0.02 0.00 0.01 0.00 0.03 0.00 0.08 0.01 0.01 0.02 0.01 0.01 0.00 0.00 0.03
## R 0.01 0.05 0.00 0.02 0.01 0.01 0.03 0.09 0.01 0.01 0.09 0.01 0.01 0.03 0.02
## S 0.02 0.03 0.01 0.01 0.06 0.02 0.02 0.00 0.01 0.02 0.01 0.01 0.00 0.00 0.01
## T 0.00 0.00 0.01 0.03 0.01 0.05 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
## U 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.03 0.00 0.00 0.00 0.00 0.00 0.01 0.00
## V 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
## W 0.00 0.01 0.02 0.00 0.01 0.00 0.05 0.01 0.00 0.00 0.01 0.00 0.04 0.01 0.06
## X 0.05 0.02 0.00 0.00 0.10 0.02 0.00 0.01 0.02 0.02 0.07 0.01 0.00 0.00 0.00
## Y 0.01 0.02 0.00 0.00 0.01 0.02 0.00 0.00 0.00 0.00 0.02 0.03 0.00 0.04 0.00
## Z 0.01 0.00 0.01 0.00 0.02 0.00 0.01 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00
##      P    Q    R    S    T    U    V    W    X    Y    Z
## A 0.00 0.01 0.01 0.07 0.00 0.00 0.00 0.00 0.01 0.00 0.00
## B 0.02 0.03 0.03 0.06 0.00 0.00 0.01 0.01 0.04 0.00 0.02
## C 0.00 0.01 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.00
## D 0.03 0.00 0.05 0.01 0.00 0.00 0.00 0.00 0.00 0.02 0.01
## E 0.02 0.03 0.03 0.03 0.05 0.00 0.00 0.00 0.00 0.01 0.05
## F 0.05 0.00 0.00 0.00 0.05 0.00 0.00 0.01 0.00 0.01 0.01
## G 0.03 0.03 0.00 0.02 0.00 0.01 0.01 0.01 0.00 0.00 0.00
## H 0.00 0.00 0.01 0.00 0.00 0.00 0.01 0.00 0.01 0.00 0.00
## I 0.00 0.00 0.00 0.04 0.00 0.00 0.00 0.00 0.01 0.00 0.00
## J 0.00 0.00 0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.02
## K 0.01 0.00 0.02 0.01 0.00 0.00 0.00 0.00 0.04 0.00 0.00
## L 0.00 0.08 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00 0.01
## M 0.00 0.01 0.01 0.00 0.00 0.04 0.00 0.08 0.00 0.00 0.00
## N 0.00 0.00 0.02 0.00 0.00 0.06 0.02 0.04 0.00 0.00 0.00
## O 0.01 0.06 0.01 0.00 0.01 0.03 0.00 0.01 0.09 0.02 0.00
## P 0.00 0.00 0.00 0.00 0.00 0.00 0.03 0.00 0.00 0.01 0.00
## Q 0.01 0.00 0.00 0.03 0.00 0.03 0.00 0.02 0.00 0.02 0.01
## R 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.01 0.01 0.00 0.01
## S 0.00 0.02 0.01 0.00 0.00 0.00 0.00 0.00 0.04 0.03 0.06
## T 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.02 0.00
## U 0.00 0.00 0.00 0.01 0.02 0.00 0.01 0.00 0.00 0.03 0.00
## V 0.00 0.01 0.00 0.01 0.05 0.02 0.00 0.01 0.01 0.10 0.00
## W 0.01 0.00 0.00 0.00 0.00 0.02 0.04 0.00 0.00 0.02 0.00
## X 0.00 0.00 0.02 0.03 0.03 0.00 0.00 0.00 0.00 0.01 0.00
## Y 0.02 0.00 0.01 0.02 0.04 0.02 0.00 0.00 0.04 0.00 0.00
## Z 0.00 0.02 0.00 0.06 0.02 0.00 0.00 0.00 0.00 0.00 0.00
```

```
which(letters_conf_prop_slow == max(letters_conf_prop_slow), arr.ind=TRUE)
```

```
##   row col
## X  24   5
```

```r
max(letters_conf_prop_slow)
```

```
## [1] 0.1028571
```

```r
mixup_mat_slow <- ( letters_conf_prop_slow + t(letters_conf_prop_slow) )/2
mixup_mat_slow %>% round(digits = 2)
```

```
##      A    B    C    D    E    F    G    H    I    J    K    L    M    N    O
## A 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.01 0.01 0.01
## B 0.00 0.00 0.01 0.05 0.01 0.03 0.01 0.03 0.02 0.02 0.01 0.01 0.01 0.01 0.00
## C 0.00 0.01 0.00 0.00 0.05 0.00 0.05 0.00 0.01 0.00 0.03 0.01 0.00 0.01 0.03
## D 0.00 0.05 0.00 0.00 0.00 0.01 0.01 0.02 0.01 0.02 0.01 0.00 0.00 0.03 0.03
## E 0.00 0.01 0.05 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.02 0.01 0.00 0.00 0.00
## F 0.00 0.03 0.00 0.01 0.00 0.00 0.01 0.00 0.02 0.02 0.00 0.00 0.00 0.01 0.00
## G 0.00 0.01 0.05 0.01 0.01 0.01 0.00 0.01 0.00 0.00 0.01 0.01 0.01 0.00 0.01
## H 0.00 0.03 0.00 0.02 0.00 0.00 0.01 0.00 0.00 0.01 0.03 0.00 0.00 0.01 0.03
## I 0.00 0.02 0.01 0.01 0.00 0.02 0.00 0.00 0.00 0.02 0.00 0.01 0.00 0.00 0.00
## J 0.00 0.02 0.00 0.02 0.00 0.02 0.00 0.01 0.02 0.00 0.00 0.00 0.00 0.01 0.01
## K 0.00 0.01 0.03 0.01 0.02 0.00 0.01 0.03 0.00 0.00 0.00 0.00 0.02 0.03 0.00
## L 0.00 0.01 0.01 0.00 0.01 0.00 0.01 0.00 0.01 0.00 0.00 0.00 0.00 0.00 0.00
## M 0.01 0.01 0.00 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.02 0.00 0.00 0.02 0.00
## N 0.01 0.01 0.01 0.03 0.00 0.01 0.00 0.01 0.00 0.01 0.03 0.00 0.02 0.00 0.02
## O 0.01 0.00 0.03 0.03 0.00 0.00 0.01 0.03 0.00 0.01 0.00 0.00 0.00 0.02 0.00
## P 0.00 0.01 0.00 0.03 0.01 0.05 0.02 0.00 0.01 0.00 0.00 0.00 0.00 0.02 0.01
## Q 0.02 0.02 0.01 0.00 0.03 0.00 0.05 0.01 0.01 0.01 0.00 0.04 0.00 0.00 0.05
## R 0.01 0.04 0.00 0.03 0.02 0.00 0.02 0.05 0.00 0.01 0.05 0.00 0.01 0.03 0.02
## S 0.04 0.05 0.00 0.01 0.04 0.01 0.02 0.00 0.02 0.01 0.01 0.01 0.00 0.00 0.00
## T 0.00 0.00 0.00 0.01 0.03 0.05 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
## U 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.02 0.00 0.00 0.00 0.00 0.02 0.03 0.02
## V 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.01 0.00 0.00 0.00 0.00 0.00 0.01 0.00
## W 0.00 0.01 0.01 0.00 0.00 0.00 0.03 0.01 0.00 0.00 0.01 0.00 0.06 0.03 0.03
## X 0.03 0.03 0.00 0.00 0.05 0.01 0.00 0.01 0.01 0.01 0.06 0.01 0.00 0.00 0.04
## Y 0.00 0.01 0.00 0.01 0.01 0.01 0.00 0.00 0.00 0.00 0.01 0.02 0.00 0.02 0.01
## Z 0.00 0.01 0.01 0.00 0.04 0.01 0.00 0.00 0.00 0.01 0.00 0.01 0.00 0.00 0.00
##      P    Q    R    S    T    U    V    W    X    Y    Z
## A 0.00 0.02 0.01 0.04 0.00 0.00 0.00 0.00 0.03 0.00 0.00
## B 0.01 0.02 0.04 0.05 0.00 0.00 0.01 0.01 0.03 0.01 0.01
## C 0.00 0.01 0.00 0.00 0.00 0.02 0.00 0.01 0.00 0.00 0.01
## D 0.03 0.00 0.03 0.01 0.01 0.00 0.00 0.00 0.00 0.01 0.00
## E 0.01 0.03 0.02 0.04 0.03 0.00 0.00 0.00 0.05 0.01 0.04
## F 0.05 0.00 0.00 0.01 0.05 0.00 0.00 0.00 0.01 0.01 0.01
## G 0.02 0.05 0.02 0.02 0.00 0.00 0.01 0.03 0.00 0.00 0.00
## H 0.00 0.01 0.05 0.00 0.00 0.02 0.01 0.01 0.01 0.00 0.00
## I 0.01 0.01 0.00 0.02 0.00 0.00 0.00 0.00 0.01 0.00 0.00
## J 0.00 0.01 0.01 0.01 0.00 0.00 0.00 0.00 0.01 0.00 0.01
## K 0.00 0.00 0.05 0.01 0.00 0.00 0.00 0.01 0.06 0.01 0.00
## L 0.00 0.04 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.02 0.01
## M 0.00 0.00 0.01 0.00 0.00 0.02 0.00 0.06 0.00 0.00 0.00
## N 0.02 0.00 0.03 0.00 0.00 0.03 0.01 0.03 0.00 0.02 0.00
## O 0.01 0.05 0.02 0.00 0.00 0.02 0.00 0.03 0.04 0.01 0.00
## P 0.00 0.01 0.00 0.00 0.00 0.00 0.02 0.01 0.00 0.01 0.00
## Q 0.01 0.00 0.00 0.02 0.00 0.02 0.01 0.01 0.00 0.01 0.02
## R 0.00 0.00 0.00 0.01 0.00 0.00 0.00 0.00 0.02 0.00 0.00
## S 0.00 0.02 0.01 0.00 0.01 0.00 0.01 0.00 0.03 0.03 0.06
## T 0.00 0.00 0.00 0.01 0.00 0.01 0.02 0.00 0.02 0.03 0.01
## U 0.00 0.02 0.00 0.00 0.01 0.00 0.02 0.01 0.00 0.02 0.00
## V 0.02 0.01 0.00 0.01 0.02 0.02 0.00 0.02 0.00 0.05 0.00
## W 0.01 0.01 0.00 0.00 0.00 0.01 0.02 0.00 0.00 0.01 0.00
## X 0.00 0.00 0.02 0.03 0.02 0.00 0.00 0.00 0.00 0.02 0.00
```

```
## Y 0.01 0.01 0.00 0.03 0.03 0.02 0.05 0.01 0.02 0.00 0.00
## Z 0.00 0.02 0.00 0.06 0.01 0.00 0.00 0.00 0.00 0.00 0.00
```

```
#highest rate
which(mixup_mat_slow == max(mixup_mat_slow)[], arr.ind=TRUE)
```

```
##   row col
## Z  26  19
## S  19  26
```

```
max(mixup_mat_slow)
```

```
## [1] 0.05989957
```

```
#second highest rate
which(mixup_mat_slow == max( mixup_mat_slow[mixup_mat_slow!=max(mixup_mat_slow)] ), arr.ind=TRUE)
```

```
##   row col
## X  24  11
## K  11  24
```

```
max( mixup_mat_slow[mixup_mat_slow!=max(mixup_mat_slow)] )
```

```
## [1] 0.05702488
```

Problem 4

Return to the `College` data set from HW 8, which can be loaded with the following code:

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 3.6.2
```

```
data("College")
```

a. Split the data into a test and training set.

b. 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 rMSEs.

c. Construct a model based on a boosted tree with parameters of your choosing, and compute the rMSE.

d. Compare the rMSE of the bagged tree, the Random Forest, and the boosted tree to the rMSE you for the pruned regression tree, as well at the linear regression model, from HW 8. Which model was most accurate?

e. 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, as well as the boosted tree. 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 8?
