Lab 8

Alyssa Andrichik

11/17/2019

Lab 8: Ransom notes keep falling

One of the most useful applications to come out of classification models has been character (i.e. letter) recognition. In this lab, we build our 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.

```
## -- Attaching packages ---------------------------------------------------------------------------------
## v ggplot2 3.2.1       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
## -- Conflicts ------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## 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
```

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.

In addition, each row/image was labeled with the letter that it corresponds to.

You will want to build your model on a training data set and evaluate its performance on a separate test data set. Please use the following indices to subset out the training data set, leaving the remaining as test.

```
#Training and Testing Data
set.seed(1)
train <- sample(1:nrow(lettersdf), nrow(lettersdf) * .75)
traindata <- lettersdf[train, ]
testdata <- lettersdf[-train, ]
```

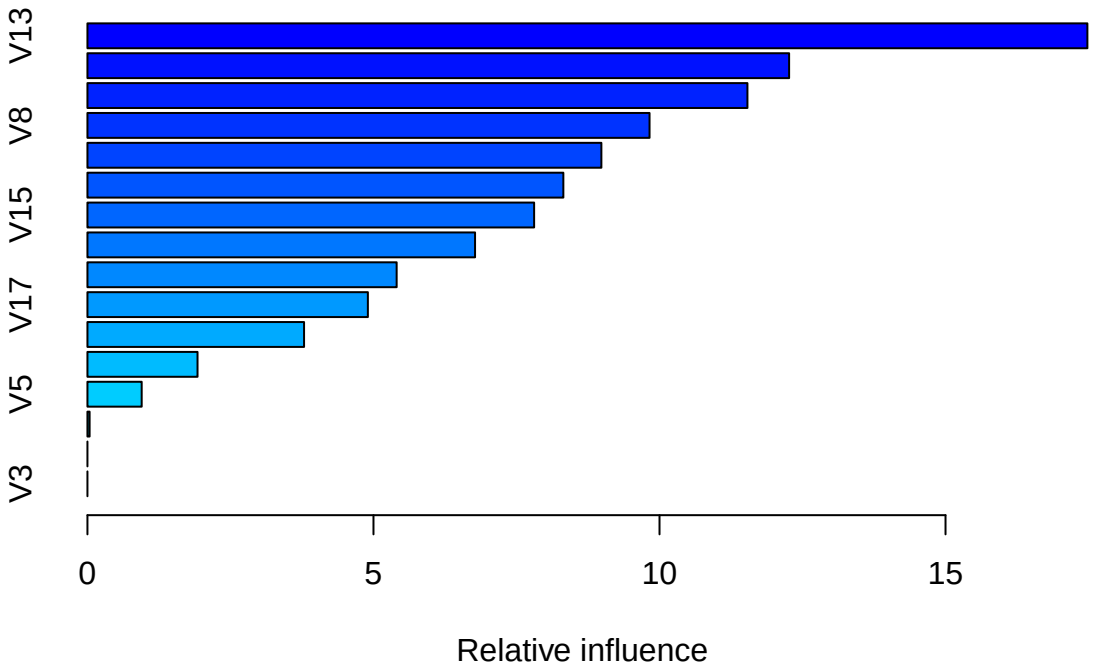
Building a boosted tree

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. Please 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. Which variable is found to be the most important?

```
#Boosted Model
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
boost.letters <- gbm(formula = V1 ~., data = traindata,
                     distribution = "multinomial", n.trees = 50,
                     shrinkage = 0.1, interaction.depth = 1)
summary(boost.letters)
```




```
##     var     rel.inf
## V13 V13 17.47975897
## V12 V12 12.26575586
## V14 V14 11.53730240
## V8   V8  9.82580683
## V10 V10  8.98367860
## V11 V11  8.31923619
## V15 V15  7.80833554
## V9   V9  6.77590463
## V16 V16  5.40537511
## V17 V17  4.90210556
## V7   V7  3.78671390
## V4   V4  1.92421542
## V5   V5  0.94803360
## V6   V6  0.03777739
## V2   V2  0.00000000
## V3   V3  0.00000000
```

V13, or 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), is the most important variable.

Assessing predictions

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.

```
#Prediciton
yhat.boost <- predict(boost.letters, newdata = testdata, n.trees = 50, type = "response", shrinkage = 0
predicted <- LETTERS[apply(yhat.boost, 1, which.max)]
```

1. Build a cross-tabulation of the predicted and actual letters (a 26 X 26 confusion matrix).

```
#Confusion Matrix
tab <- as.matrix(table(predicted, testdata$V1))
tab
```

```
##
## predicted   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P
##         A 176   0   0   0   0   0   1   0   1   0   2  10   5   0   0   0
##         B   0 129   0  26   5  15   3   7  12  17   2   3   1   5   1   6
##         C   3   0 130   0  26   0  15   0   1   3   7   7   1   3   1   0
##         D   0  20   0 131   0  13   6  10   6   6   4   0   1   4  10  13
##         E   0   0  11   1  72   1   3   0   0   0   5   1   0   0   0   1
##         F   0   0   3   0   0 119   0   1   2   3   0   0   0   0   0  16
##         G   1   2   6   0  22   6 112   4   1   0   4   8   0   0   5   3
##         H   0   0   0   1   0   0   1  82   0   0   4   0   1   1   0   0
##         I   0   0   0   0   0   4   0   0 148   2   0   0   0   0   0   1
##         J   3   0   0   7   0   2   0   1   9 131   0   0   0   1   0   2
##         K   0   1  17   2  10   0   4  13   0   0 108   3   3   0   1   0
##         L   2   0   0   0   0   0   2   0   0   0   1 146   0   0   3   0
##         M   3   7   0   1   0   1   0   3   0   3   6   0 178   8   0   0
##         N   0   2   0   4   1   0   0   5   0   0   5   0   5 157   1   0
##         O   5   1   9   7   0   0   1  28   0   4   0   0   5   8 147  10
##         P   0   0   0   8   0  14   0   0   3   3   0   0   0   9   0 134
##         Q   1   1   1   0   8   0  19   6   1   5   0   2   1   0   5   1
##         R   0  12   0   7   7   3  13   9   2   7  17   2   1   2   2   0
##         S   2   5   4   6   9   5   6   2   2   7   0   3   1   0   0   0
##         T   0   0   0   2   0   6   0   2   0   0   0   0   0   0   0   0
##         U   0   0   2   0   2   1   0  12   0   0   0   0   0   1   2   0
##         V   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         W   0   1   2   0   0   0   6   6   0   0   2   0   4   4  10   9
##         X   5   3   0   0  31   2   0   3   5   0  10   1   0   0   0   0
##         Y   2   0   0   0   2   5   0   0   0   0   2   8   0   5   0   4
##         Z   1   0   1   0  11   0   2   0   0   0   0   0   0   0   0   0
##
## predicted   Q   R   S   T   U   V   W   X   Y   Z
##         A   0   0   8   0   0   0   0   0   0   0
##         B   8  16  13   2   1   0   0   8   1   4
##         C   6   0   0   0   1   0   0   0   0   0
##         D   0   8   5   0   1   0   0   4   4   1
##         E   1   5   2   4   3   0   0   2   0   7
##         F   0   0   2  15   0   4   4   0   5   0
##         G  14   1   1   0   1   0   0   0   0   1
##         H   0   0   2   0   0   0   0  12   0   0
##         I   0   0   4   5   0   0   0   0   2   0
##         J   2   0   1   0   0   0   0   0   0   2
##         K   0   2   0   2   2   0   0   8   0   1
##         L  11   0   1   0   0   0   0   0   0   0
##         M   1   9   0   0  13   2  13   1   1   0
##         N   0   2   0   5  15   7   4   0   0   0
##         O  16   3   3   3  11   1   4   4   1   0
##         P   0   0   0   1   0   2   0   0   1   0
##         Q  98   1   3   0   3   4   0   0   8   1
##         R   1 152  11   0   0   0   1   1   0   4
```

```
##         S   6   0 122   0   0   1   0   4   4  17
##         T   0   0   2 137   2   5   0   0  11   3
##         U   0   0   1   5 145   3   0   0   3   0
##         V   0   0   0  10   5 127   2   0  15   0
##         W   1   3   0   0   1  14 142   0   2   0
##         X   0   3  15   8   0   0   0 128   0   3
##         Y   1   0   1  11   2   4   0   9 120   0
##         Z   0   0   3   2   0   0   0   5   0 133
```

2. What is your misclassification rate?

```
#Number of instances
n <- sum(tab)
n
```

```
## [1] 5000
```

```
#Number of classes
nc <- nrow(tab)
nc
```

```
## [1] 26
```

```
#Number of correctly classified instances per class
diag <- diag(tab)
diag
```

```
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 176 129 130 131  72 119 112  82 148 131 108 146 178 157 147 134  98 152
##   S   T   U   V   W   X   Y   Z
## 122 137 145 127 142 128 120 133
```

```
#Number of instances per class
rowsums <- apply(tab, 1, sum)
rowsums
```

```
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 203 285 204 247 119 174 192 104 166 161 177 166 250 213 271 175 169 254
##   S   T   U   V   W   X   Y   Z
## 206 170 177 159 207 217 176 158
```

```
#Number of predictions per class
colsums <- apply(tab, 2, sum)
colsums
```

```
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 204 184 186 203 206 197 194 194 193 191 179 194 207 208 188 200 166 205
##   S   T   U   V   W   X   Y   Z
## 200 210 206 174 170 186 178 177
```

```
#Distribution of instances over the actual classes
p <- rowsums / n
p
```

```
##      A      B      C      D      E      F      G      H      I      J
## 0.0406 0.0570 0.0408 0.0494 0.0238 0.0348 0.0384 0.0208 0.0332 0.0322
##      K      L      M      N      O      P      Q      R      S      T
## 0.0354 0.0332 0.0500 0.0426 0.0542 0.0350 0.0338 0.0508 0.0412 0.0340
```

```
##      U      V      W      X      Y      Z
## 0.0354 0.0318 0.0414 0.0434 0.0352 0.0316
```

```
#Distribution of instances over the predicted classes
q <- colsums / n
q
```

```
##      A      B      C      D      E      F      G      H      I      J
## 0.0408 0.0368 0.0372 0.0406 0.0412 0.0394 0.0388 0.0388 0.0386 0.0382
##      K      L      M      N      O      P      Q      R      S      T
## 0.0358 0.0388 0.0414 0.0416 0.0376 0.0400 0.0332 0.0410 0.0400 0.0420
##      U      V      W      X      Y      Z
## 0.0412 0.0348 0.0340 0.0372 0.0356 0.0354
```

```
#Misclassification Rate
Misclass <- 1-sum(diag/n)
Misclass
```

```
## [1] 0.3192
```

The misclassification rate is 0.3192.

3. What letter was most difficult to predict?

```
#Precision is defined as the fraction of correct predictions for a certain class
precision <- diag / colsums
precision
```

```
##         A         B         C         D         E         F         G
## 0.8627451 0.7010870 0.6989247 0.6453202 0.3495146 0.6040609 0.5773196
##         H         I         J         K         L         M         N
## 0.4226804 0.7668394 0.6858639 0.6033520 0.7525773 0.8599034 0.7548077
##         O         P         Q         R         S         T         U
## 0.7819149 0.6700000 0.5903614 0.7414634 0.6100000 0.6523810 0.7038835
##         V         W         X         Y         Z
## 0.7298851 0.8352941 0.6881720 0.6741573 0.7514124
```

Looking at the precision of the classification of every letter, the letter E was the most difficult letter to predict since the prediction was only correct 34.95 percent of the time.

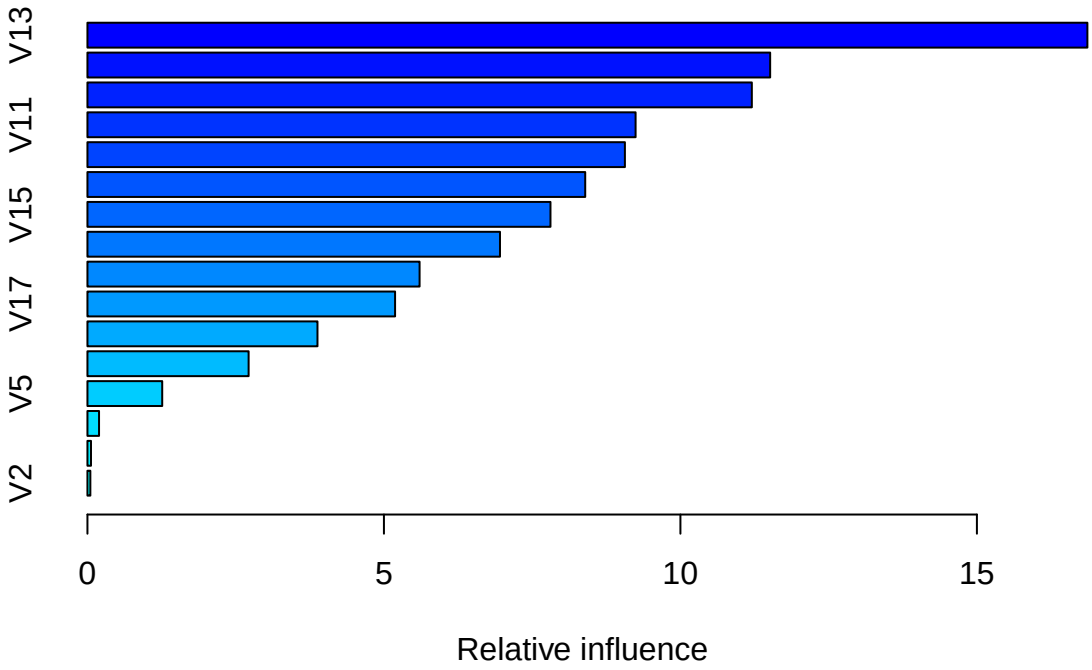
4. Are there any letter pairs that are particularly difficult to distinguish?

D & B seems to be pretty difficult for the model to distinguish from one another. B was misclassified as D 20 times and D was misclassified as B 26 times. Q & G also seems to be difficult for the model to distinguish since G was misclassified as Q 19 times and Q misclassified as G 14 times.

Slow the learning

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.

```
#Model
boost2.letters <- gbm(formula = V1 ~., data = traindata,
                      distribution = "multinomial", n.trees = 1000,
                      shrinkage = 0.01, interaction.depth = 1)
summary(boost2.letters)
```




```
##     var     rel.inf
## V13 V13 16.86298812
## V12 V12 11.51314932
## V14 V14 11.20406534
## V11 V11  9.24405516
## V8   V8  9.06374704
## V10 V10  8.39433035
## V15 V15  7.80840617
## V9   V9  6.95727428
## V16 V16  5.60131971
## V17 V17  5.18769101
## V7   V7  3.87929974
## V4   V4  2.71791872
## V5   V5  1.26078187
## V6   V6  0.19578260
## V3   V3  0.05998762
## V2   V2  0.04920296
```

```
#Prediciton
yhat2.boost <- predict(boost2.letters, newdata = testdata, n.trees = 1000, type = "response", shrinkage
predicted2 <- LETTERS[apply(yhat2.boost, 1, which.max)]
```

```
#Confusion Matrix
tab2 <- as.matrix(table(predicted2, testdata$V1))
tab2
```

```
##
## predicted2   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P
```

```
##           A 178   0   0   0   0   0   1   1   1   0   2   5   3   0   0   0
##           B   0 138   1  18   0  11   2   8  12   9   5   1   2   3   1   8
##           C   3   0 131   0  15   0  14   0   2   0   1   3   0   2   1   0
##           D   0  11   0 136   0   4   5  10   4   5   5   0   1   3   8   7
##           E   0   0   8   0 111   3   2   0   0   0   3   0   0   0   0   2
##           F   0   1   0   3   0 128   0   3   2   5   0   0   0   0   0  16
##           G   0   2   7   0  13   7 122   3   1   0   3   6   1   0   3   3
##           H   0   0   0   6   0   0   3 109   1   0   6   0   2   0   1   0
##           I   0   0   0   0   0  10   0   0 151   1   0   0   0   0   0   1
##           J   0   0   0   4   0   1   0   0   6 145   0   0   0   1   0   1
##           K   1   1  19   5  13   0   1  13   0   1 125   3   3   1   0   0
##           L   3   0   2   1   0   0   1   0   0   0   0 157   0   0   4   0
##           M   2   5   0   0   0   0   0   2   0   1   1   0 180   7   0   0
##           N   0   3   0   4   0   0   0   1   0   1   5   0   6 166   1   0
##           O   3   1   8   5   0   0   1  16   0   3   0   0   4   7 153   4
##           P   0   0   0   7   0  14   0   1   4   6   0   0   0   5   0 144
##           Q   5   2   1   0   9   0  22   3   0   3   1   3   0   0   4   1
##           R   0  13   0   5   0   1   9   9   0   1  11   2   2   4   3   0
##           S   4   4   5   4   9   5   4   1   2   5   0   3   0   0   0   0
##           T   0   0   0   5   2   4   0   1   0   0   1   0   0   0   0   0
##           U   0   0   2   0   0   0   0   7   1   1   0   0   0   1   0   0
##           V   0   0   0   0   0   1   0   0   0   0   1   0   0   0   0   0
##           W   0   1   1   0   0   1   5   4   0   0   1   0   3   3   9   7
##           X   3   2   0   0  26   2   0   2   6   2   8   5   0   0   0   0
##           Y   2   0   0   0   0   5   0   0   0   0   0   6   0   5   0   6
##           Z   0   0   1   0   8   0   2   0   0   2   0   0   0   0   0   0
##
## predicted2   Q   R   S   T   U   V   W   X   Y   Z
##           A   0   0   9   0   0   1   0   0   0   0
##           B   1  16  10   3   1   0   1   4   0   4
##           C   5   0   0   0   1   0   0   0   0   0
##           D   0   6   5   0   1   0   0   2   2   0
##           E   3   4   3   5   0   1   0   2   0   7
##           F   0   0   2  12   0   1   0   0   5   0
##           G   6   1   2   1   2   0   0   0   0   0
##           H   0   0   2   2   0   0   0   0   0   0
##           I   0   0   4   2   0   0   0   0   1   1
##           J   1   0   0   0   0   0   0   0   0   4
##           K   0   2   1   2   2   0   0  12   0   0
##           L  10   0   1   0   0   0   0   0   0   2
##           M   1   8   0   0  12   1  12   0   0   0
##           N   0   0   0   0  11   5   3   0   0   0
##           O  16   0   0   1   9   0   4  16   1   0
##           P   0   0   0   0   0   5   0   0   4   0
##           Q 113   0   3   0   3   2   0   0   4   1
##           R   2 160   7   0   0   0   1   1   0   3
##           S   5   1 139   1   0   1   0   4   3  15
##           T   0   0   1 160   3   2   0   0   5   3
##           U   0   0   0   7 152   3   0   0   5   0
##           V   0   0   1   0   6 138   2   1  11   0
##           W   0   0   0   0   1  10 147   0   0   0
##           X   0   6   5   9   0   1   0 139   0   2
##           Y   1   1   1   4   2   3   0   5 137   0
##           Z   2   0   4   1   0   0   0   0   0 135
```

```r
#Number of instances
n2 <- sum(tab2)
n2
```

```
## [1] 5000
```

```r
#Number of classes
nc2 <- nrow(tab2)
nc2
```

```
## [1] 26
```

```r
#Number of correctly classified instances per class
diag2 <- diag(tab2)
diag2
```

```
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 178 138 131 136 111 128 122 109 151 145 125 157 180 166 153 144 113 160
##   S   T   U   V   W   X   Y   Z
## 139 160 152 138 147 139 137 135
```

```r
#Number of instances per class
rowsums2 <- apply(tab2, 1, sum)
rowsums2
```

```
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 201 259 178 215 154 178 183 132 171 163 205 181 232 206 252 190 180 234
##   S   T   U   V   W   X   Y   Z
## 215 187 179 161 193 218 178 155
```

```r
#Number of predictions per class
colsums2 <- apply(tab2, 2, sum)
colsums2
```

```
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 204 184 186 203 206 197 194 194 193 191 179 194 207 208 188 200 166 205
##   S   T   U   V   W   X   Y   Z
## 200 210 206 174 170 186 178 177
```

```r
#Distribution of instances over the actual classes
p2 <- rowsums2 / n2
p2
```

```
##      A      B      C      D      E      F      G      H      I      J
## 0.0402 0.0518 0.0356 0.0430 0.0308 0.0356 0.0366 0.0264 0.0342 0.0326
##      K      L      M      N      O      P      Q      R      S      T
## 0.0410 0.0362 0.0464 0.0412 0.0504 0.0380 0.0360 0.0468 0.0430 0.0374
##      U      V      W      X      Y      Z
## 0.0358 0.0322 0.0386 0.0436 0.0356 0.0310
```

```r
#Distribution of instances over the predicted classes
q2 <- colsums2 / n2
q2
```

```
##      A      B      C      D      E      F      G      H      I      J
## 0.0408 0.0368 0.0372 0.0406 0.0412 0.0394 0.0388 0.0388 0.0386 0.0382
##      K      L      M      N      O      P      Q      R      S      T
## 0.0358 0.0388 0.0414 0.0416 0.0376 0.0400 0.0332 0.0410 0.0400 0.0420
##      U      V      W      X      Y      Z
```

```
## 0.0412 0.0348 0.0340 0.0372 0.0356 0.0354
```

```
#Misclassification Rate
Misclass2 <- 1-sum(diag2/n2)
Misclass2
```

```
## [1] 0.2612
```

1. How does the misclassification rate compare to the rate from your original model?

The Misclassification rate decreased from 0.3192 to 0.2618! Less of the letters were misclassified in the new model.

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

The new model distinguished the letters B & D more correctly. Only 13 cases of B being misclassified as D and 17 cases of D being misclassified as B. However, F was misclassified as P 15 times and P was misclassified as F 18 times in the new model which is up from the original where F was only misclassified as P 14 times and P was only misclassified as F 16 times. So, distinguishing P from F became harder in the new model. G was misclassified at Q 25 times and Q misclassified as G 6 times, which is quite different from my original model where G was only misclassified as Q 19 times and Q misclassified as G 14 times.

Communities and Crime

Return to the Communities and Crime data set.

One last boost

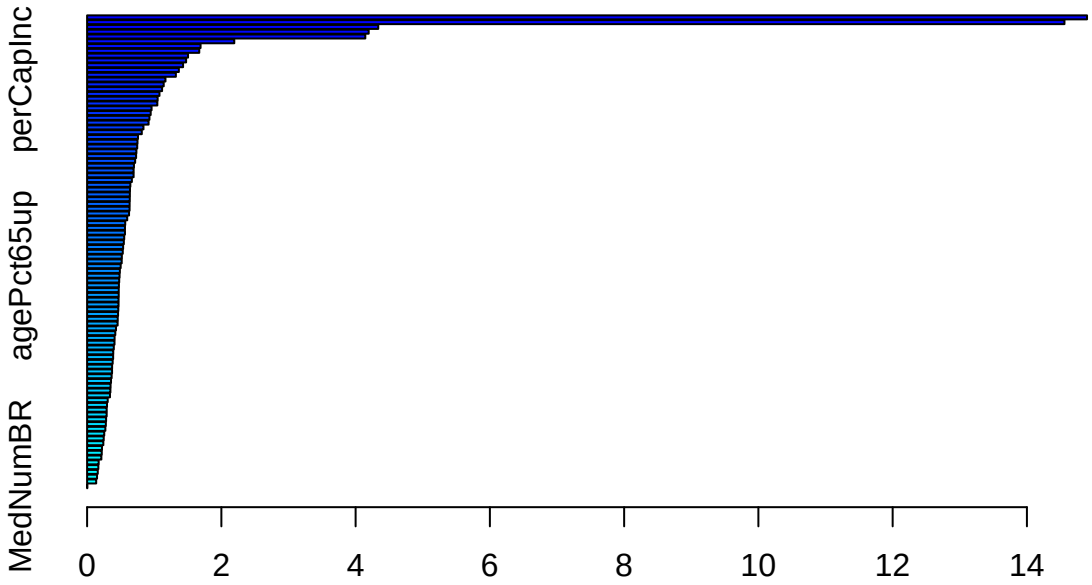
Construct a model based on a boosted tree with parameters of your choosing. How does the test MSE compare to your existing models (Bagged Trees, Random Forests, etc.)?

```
d <- read.csv("http://andrewpbray.github.io/data/crime-train.csv")
d[d=="?"]<-NA
d[d==""]<-NA

newd <- d %>%
  select(-c(state, county, community, communityname, population, LemasSwornFT, LemasSwFTPerPop,
            LemasSwFTFieldOps, LemasSwFTFieldPerPop, LemasTotalReq, LemasTotReqPerPop,
            PolicReqPerOffic, PolicPerPop, RacialMatchCommPol, PctPolicWhite, PctPolicBlack,
            PctPolicHisp, PctPolicAsian, PctPolicMinor, OfficAssgnDrugUnits, NumKindsDrugsSeiz,
            PolicAveOTWorked, PolicCars, PolicOperBudg, LemasPctPolicOnPatr, LemasGangUnitDeploy,
            PolicBudgPerPop)) %>%
  drop_na()
```

```
#Training and Testing Data
set.seed(1)
traincrime <- sample(1:nrow(newd), nrow(newd) * .75)
critraindata <- newd[traincrime, ]
critestdata <- newd[-traincrime, ]
```

```r
#Boosted Model
boost.crime <- gbm(formula = ViolentCrimesPerPop ~., data = critraindata,
                   distribution = "gaussian", n.trees = 100000,
                   shrinkage = 0.01, interaction.depth = 3)
summary(boost.crime)
```




```
##                                          var      rel.inf
## PctKids2Par                       PctKids2Par 14.895542565
## PctIlleg                             PctIlleg 14.559599814
## racePctWhite                     racePctWhite  4.337246202
## NumIlleg                             NumIlleg  4.194730389
## racepctblack                     racepctblack  4.144320122
## MalePctDivorce                 MalePctDivorce  2.192593526
## TotalPctDiv                       TotalPctDiv  1.690418520
## PctVacantBoarded             PctVacantBoarded  1.670939027
## NumStreet                           NumStreet  1.503563979
## PctPersDenseHous             PctPersDenseHous  1.474865677
## PctLargHouseFam               PctLargHouseFam  1.429721945
## pctWInvInc                         pctWInvInc  1.366821276
## PctHousLess3BR                 PctHousLess3BR  1.322596427
## perCapInc                           perCapInc  1.167559489
## blackPerCap                       blackPerCap  1.141277590
## PctImmigRec5                     PctImmigRec5  1.116969455
## FemalePctDiv                     FemalePctDiv  1.078297137
## PctImmigRec8                     PctImmigRec8  1.051049931
## PctFam2Par                         PctFam2Par  1.046463656
## PctOccupMgmtProf             PctOccupMgmtProf  0.960737989
## PctBSorMore                       PctBSorMore  0.947522160
## PctEmplProfServ                 PctEmplProfServ  0.928794588
## PctHousNoPhone                   PctHousNoPhone  0.916065303
## PctNotHSGrad                       PctNotHSGrad  0.839634787
## MedOwnCostPctIncNoMtg MedOwnCostPctIncNoMtg  0.814797367
```

```
## MedRentPctHousInc         MedRentPctHousInc  0.757887503
## PctBornSameState           PctBornSameState  0.750615001
## MedYrHousBuilt               MedYrHousBuilt  0.746792633
## MalePctNevMarr               MalePctNevMarr  0.734536385
## PopDens                             PopDens  0.730307680
## pctWSocSec                       pctWSocSec  0.715991445
## AsianPerCap                     AsianPerCap  0.698084755
## PctSameState85               PctSameState85  0.693045157
## PctWorkMomYoungKids     PctWorkMomYoungKids  0.692621991
## pctWPubAsst                     pctWPubAsst  0.666937883
## PctWOFullPlumb               PctWOFullPlumb  0.644505268
## PctOccupManu                   PctOccupManu  0.638819893
## racePctHisp                     racePctHisp  0.637079099
## PctTeen2Par                     PctTeen2Par  0.636035682
## PctUnemployed                 PctUnemployed  0.633634921
## PctImmigRecent               PctImmigRecent  0.632903552
## whitePerCap                     whitePerCap  0.624694492
## agePct12t21                     agePct12t21  0.597962935
## RentLowQ                           RentLowQ  0.567874024
## PersPerFam                       PersPerFam  0.564254789
## PctNotSpeakEnglWell     PctNotSpeakEnglWell  0.564222925
## agePct16t24                     agePct16t24  0.553072404
## PctSameCity85                 PctSameCity85  0.547319850
## PctVacMore6Mos               PctVacMore6Mos  0.536653065
## PersPerRentOccHous       PersPerRentOccHous  0.531479699
## PctEmplManu                     PctEmplManu  0.517776697
## RentMedian                       RentMedian  0.515803854
## PctImmigRec10                 PctImmigRec10  0.500258011
## PctYoungKids2Par           PctYoungKids2Par  0.488164145
## agePct65up                       agePct65up  0.483721594
## pctWFarmSelf                   pctWFarmSelf  0.478288836
## PctHousOccup                   PctHousOccup  0.473911427
## PctSameHouse85               PctSameHouse85  0.471560470
## HispPerCap                       HispPerCap  0.471325812
## RentHighQ                         RentHighQ  0.469030850
## PctSpeakEnglOnly           PctSpeakEnglOnly  0.468799817
## numbUrban                         numbUrban  0.465036931
## PctPopUnderPov               PctPopUnderPov  0.461656945
## PctWorkMom                       PctWorkMom  0.456285794
## PctForeignBborn             PctForeignBborn  0.454796626
## HousVacant                       HousVacant  0.430604249
## PctLess9thGrade             PctLess9thGrade  0.420151761
## MedOwnCostPctInc           MedOwnCostPctInc  0.409467434
## pctWWage                           pctWWage  0.408302829
## medIncome                         medIncome  0.398140205
## PctUsePubTrans               PctUsePubTrans  0.389506809
## pctWRetire                       pctWRetire  0.388048798
## householdsize                 householdsize  0.380798401
## PctLargHouseOccup         PctLargHouseOccup  0.372517114
## medFamInc                         medFamInc  0.371975384
## OtherPerCap                     OtherPerCap  0.365588768
## NumInShelters                 NumInShelters  0.356260601
## racePctAsian                   racePctAsian  0.349478341
## PctPersOwnOccup             PctPersOwnOccup  0.347197603
```

```
## NumImmig                          NumImmig  0.340685974
## PctEmploy                        PctEmploy  0.307012118
## OwnOccHiQuart                OwnOccHiQuart  0.296904934
## PctRecImmig10                PctRecImmig10  0.291878386
## PctHousOwnOcc                PctHousOwnOcc  0.291636956
## NumUnderPov                    NumUnderPov  0.283105684
## OwnOccLowQuart              OwnOccLowQuart  0.279317549
## indianPerCap                  indianPerCap  0.272192156
## PersPerOwnOccHous        PersPerOwnOccHous  0.255464458
## MedRent                            MedRent  0.247707637
## LandArea                          LandArea  0.241972530
## agePct12t29                  agePct12t29  0.221696774
## OwnOccMedVal                OwnOccMedVal  0.218670677
## PctRecImmig5                PctRecImmig5  0.212711023
## LemasPctOfficDrugUn    LemasPctOfficDrugUn  0.172256643
## PersPerOccupHous          PersPerOccupHous  0.167656183
## pctUrban                          pctUrban  0.158875586
## PctRecentImmig              PctRecentImmig  0.148114528
## PctRecImmig8                  PctRecImmig8  0.135336464
## MedNumBR                          MedNumBR  0.002885684
```

```
#Prediciton
yhat3.boost <- predict(boost.crime, newdata = critestdata, n.trees = 100000, shrinkage = 0.01, interacti
```

```
crime.test <- newd[-traincrime, "ViolentCrimesPerPop"]
testMSE <- mean((yhat3.boost-crime.test)^2)
testMSE
```

```
## [1] 0.022093
```

The test MSE from the boosted model is 0.02141982, the test MSE for my tree model is 0.01609171, the test MSE for my linear regression model is 0.02320641, and the test MSE for the random forest with $m = p$ is 0.001511666. So, my boosted model is slightly more accurate than the lineaar regression model, but not the random forest model or the normal tree model.

Chapter 8 exercises

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 $P(\text{Class is Red}|X)$:

$$0.1, 0.15, 0.2, 0.2, , 0.55, 0.6, 0.6, 0.65, 0.7, 0.75.$$

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

The majority vote approach, looking for the most commonly occurring class among the 10 predictions, would classify X as Red since there are there are more estimates that are >0.5 (0.55, 0.6, 0.65, 0.7, 0.75) than there are estimates that are <0.5 (0.1, 0.15, 0.2, 0.2). Using the average probability approach, X would be classified as Green since the average of the 10 probabilities is 0.45, which is less than 0.5.

6. Provide a detailed explanation of the algorithm that is used to fit a regression tree.

To fit a regression tree, we must first do recursive binary splitting on the full data set to minimize the RSS. This top-down, greedy strategy is applied seperately to each split part until stopping condition when every leaf has a small number of observations (n_j is greater than or equal to 5).

Second, becauses the full T_o might be too complex, we must prune the tree to identify the subtree that has the lowest estimated test MSE. This is cost complexity pruning, where the larger tree (from the originial data set) is pruned to find the sequence of best subtrees for many values of α. For $\alpha > 0$, there is a subtree (T) that gets minimized

$$\sum_{m=i}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|.$$

Here $|T|$ is the number of terminal nodes on the tree. When $\alpha = 0$ we have the original tree, and as α increases we get a more pruned version of the tree.

Third, use K-fold CV to choose α. For each fold, repeat steps 1 and 2, and then compute the test MSE as a function of α on all subtrees. Chose an α that minimizes the average error after averaging the test MSEs of each α.