Lab 8: Ransom notes keep falling

Claire Jellison

11/16/2019

```
lettersdf <- read.csv("https://raw.githubusercontent.com/stat-learning/course-materials/master/data/let
                      header = FALSE)
#summary(lettersdf)
```

```
set.seed(1)
train <- sample(1:nrow(lettersdf), nrow(lettersdf) * .75)
```

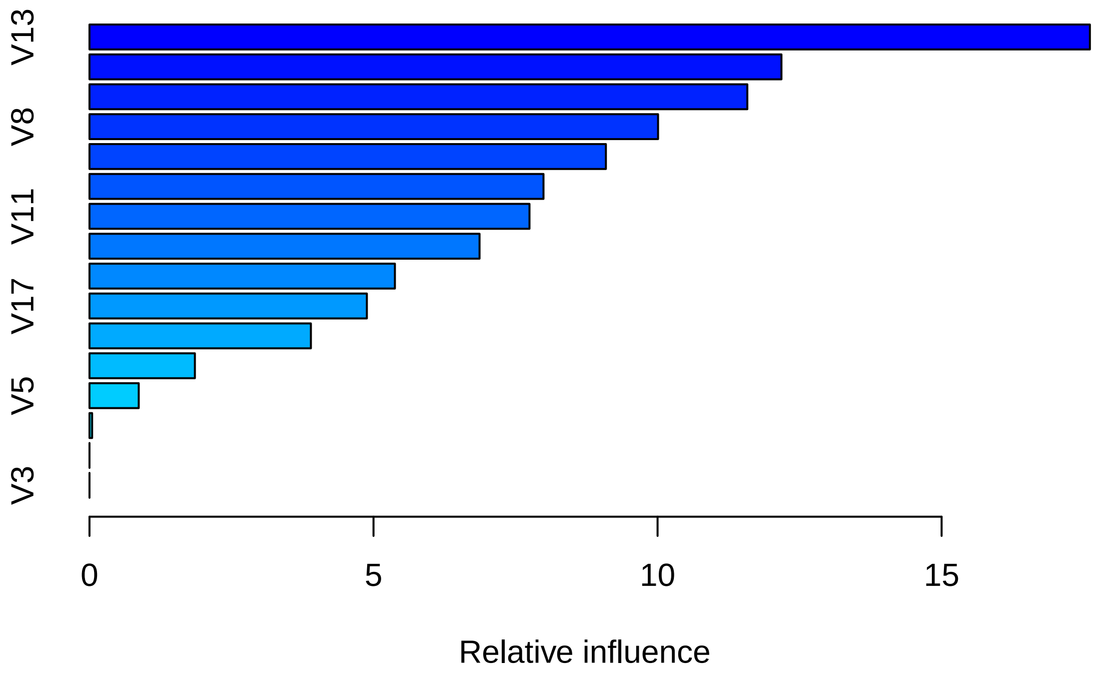
Building a boosted tree

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. 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?

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
set.seed (1)
boostwrite=gbm(V1 ~.,data=lettersdf[train,],distribution=
"multinomial", n.trees=50, interaction.depth=1, shrinkage = 0.1)
summary(boostwrite)
```



```
##      var     rel.inf
## V13 V13 17.61028348
## V12 V12 12.18059722
## V14 V14 11.58005849
## V8   V8 10.00852252
## V10 V10  9.09056884
## V15 V15  7.99175949
## V11 V11  7.74553026
## V9   V9  6.86643170
## V16 V16  5.37664184
## V17 V17  4.88307484
## V7   V7  3.89738592
## V4   V4  1.85545293
## V5   V5  0.86698917
## V6   V6  0.04670331
## V2   V2  0.00000000
## V3   V3  0.00000000
```

The most important variable appears to be V13 by a decent margin.

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.

```r
yhats <- predict(boostwrite, newdata = lettersdf[-train,], type = "response", n.trees = 50)
predictedboost <- LETTERS[apply(yhats, 1, which.max)]
#predictedboost
```

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

```r
testdata <- lettersdf[-train,]
confusionm <- table(predictedboost, testdata$V1 )
confusionm
```

```
##
## predictedboost   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
##              A 174   0   0   0   0   0   1   0   1   0   2  10   5   0   0
##              B   0 130   0  26   2  13   3  10  12  17   2   3   1   5   1
##              C   0   0 124   0  24   0  15   0   1   1   6   4   1   3   0
##              D   0  19   0 131   0  11   6  10   6   6   4   0   1   4   7
##              E   2   0  11   1  74   1   3   0   0   0   5   2   0   0   0
##              F   0   0   3   1   0 119   0   1   2   3   0   0   0   0   0
##              G   1   2   8   0  23   6 112   3   1   0   4   7   0   0   3
##              H   0   1   0   2   0   0   1  80   0   0   3   0   1   1   0
##              I   0   0   0   0   0   4   0   0 148   2   0   0   0   0   0
##              J   5   0   0   7   0   3   0   1   9 133   0   0   0   1   0
##              K   2   1  21   2  10   0   4  15   0   0 110   4   4   0   1
##              L   2   0   0   0   0   0   2   0   0   0   1 146   0   0   3
##              M   2   7   0   1   0   0   0   3   0   3   6   0 177   7   0
##              N   0   2   0   4   0   0   0   4   0   0   5   0   8 156   2
##              O   4   1   9   7   0   0   2  28   0   4   0   0   3   7 150
##              P   0   0   0   7   0  15   0   0   3   3   0   0   0   9   0
##              Q   1   1   1   0   9   0  17   6   1   5   2   3   0   0   7
##              R   2  11   0   6   7   4  13   9   2   7  15   2   1   3   3
##              S   3   5   4   4   7   5   4   1   2   7   0   4   1   0   0
##              T   0   0   0   2   0   6   0   2   0   0   0   0   0   0   0
##              U   0   1   2   0   2   1   0  12   0   0   0   0   0   3   1
##              V   0   0   0   0   0   0   1   0   0   0   0   0   0   1   0
##              W   0   1   2   0   1   1   7   7   0   0   2   0   4   3  10
##              X   4   2   0   0  35   2   0   2   5   0  10   1   0   0   0
##              Y   1   0   0   0   2   5   0   0   0   0   2   8   0   5   0
##              Z   1   0   1   2  10   1   3   0   0   0   0   0   0   0   0
##
## predictedboost   P   Q   R   S   T   U   V   W   X   Y   Z
##              A   0   0   0   8   0   0   0   0   0   0   1
##              B   6   7  17  13   2   1   0   0   8   1   4
##              C   0   5   0   0   0   1   0   0   0   0   0
##              D  11   0   8   5   0   1   0   0   4   4   1
##              E   1   2   5   2   6   2   0   0   1   0   7
##              F  18   0   0   2  14   0   3   4   0   4   0
##              G   3  11   0   1   0   1   0   0   0   0   1
##              H   0   0   0   2   0   0   0   0  12   0   0
##              I   1   0   0   5   5   0   0   0   0   1   0
##              J   3   4   0   0   0   0   0   0   0   0   1
##              K   0   0   3   0   2   3   0   0  10   0   1
##              L   0   9   0   0   0   0   0   0   0   0   0
##              M   0   1   9   0   0  13   2  13   1   1   0
##              N   0   0   1   0   3  14   8   5   0   0   0
##              O  10  18   2   3   2  11   1   4   4   1   0
##              P 133   0   0   0   1   0   3   0   0   1   0
##              Q   1 100   2   3   0   4   3   0   0   7   1
##              R   0   1 151  11   0   0   0   1   1   0   4
##              S   0   6   0 121   0   0   1   0   3   4  17
##              T   0   0   0   2 137   2   2   0   0  11   3
##              U   0   0   0   1   6 145   4   1   0   2   0
##              V   0   0   0   0  10   5 130   3   0  16   0
##              W   9   1   4   0   0   1  13 139   0   3   0
##              X   0   0   3  14   9   1   0   0 126   0   3
##              Y   4   1   0   1  11   1   4   0   9 122   0
##              Z   0   0   0   6   2   0   0   0   7   0 133
```

https://github.com/stat-learning/course-materials/blob/master/slides/week-06/lda.Rmd

2)What is your misclassification rate? (the function diag() might be helpful)

```r
diag <- diag(confusionm)
missclass <-  1 - ((1/nrow(testdata)) * (sum(diag)))
missclass
```

```
## [1] 0.3198
```

The missclassification rate is 0.3198.

3)What letter was most difficult to predict?

```
confusiondf <- as.matrix(confusionm)
missclassbyl <- rep(NA, 26)
for (i in 1: 26){
missclassbyl[i] <- 1 - ((1/sum(confusiondf[, i])) * diag[i])
}
missclassbyl
```

```
##  [1] 0.1470588 0.2934783 0.3333333 0.3546798 0.6407767 0.3959391 0.4226804
##  [8] 0.5876289 0.2331606 0.3036649 0.3854749 0.2474227 0.1449275 0.2500000
## [15] 0.2021277 0.3350000 0.3975904 0.2634146 0.3950000 0.3476190 0.2961165
## [22] 0.2528736 0.1823529 0.3225806 0.3146067 0.2485876
```

```
which.max(missclassbyl)
```

```
## [1] 5
```

The most missclassfied letter appears to be E the fifth letter.

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

```
pairs <- function(matrix) {
max = 0
letter1 = 0
letter2 = 0
for (i in 1: 26){ #iterate through all matrix ijth entries
  for (j in 1: 26)
    if (matrix[i, j] + matrix[j,i] > max & i!=j ){ # look for pairings with largest off diagonal sums
      max = matrix[i, j] + matrix[j,i]
      letter1 = i
      letter2 = j }
}
out <- cbind(letter1, letter2, max)
out
}
```

```
pairs(confusiondf)
```

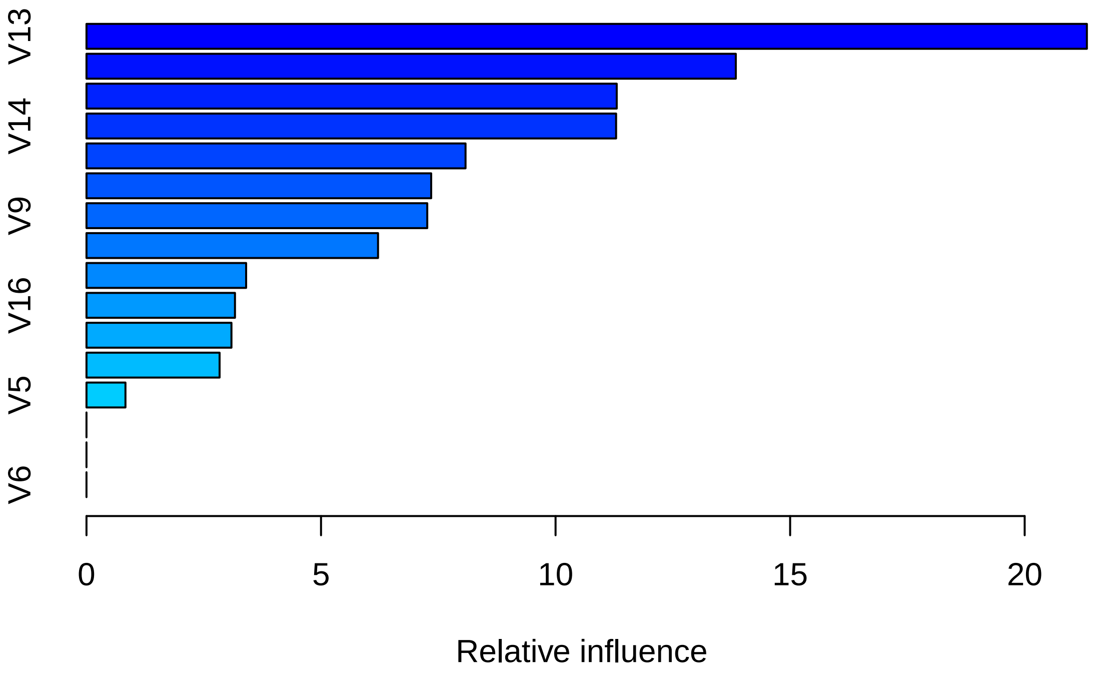
```
##      letter1 letter2 max
## [1,]       2       4  45
```

B and D appear to be particularly difficult to distinguish.

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.

```
set.seed (1)
boostwriteslow=gbm(V1 ~.,data=lettersdf[train,],distribution=
"multinomial", n.trees=100, interaction.depth=1, shrinkage = 0.01)
summary(boostwriteslow)
```



```
##     var    rel.inf
## V13 V13 21.3264047
## V12 V12 13.8424008
## V11 V11 11.3039101
## V14 V14 11.2904106
## V8   V8  8.0808200
## V10 V10  7.3483450
## V9   V9  7.2646437
## V15 V15  6.2153122
## V4   V4  3.4034863
## V16 V16  3.1655673
## V7   V7  3.0903327
## V17 V17  2.8372837
## V5   V5  0.8310828
## V2   V2  0.0000000
## V3   V3  0.0000000
## V6   V6  0.0000000
```

1)How does the misclassification rate compare to the rate from you original model?

```
yhatslow=predict(boostwriteslow,newdata=lettersdf[-train,], n.trees = 50, type = "response")
predicted2 <- LETTERS[apply(yhatslow, 1, which.max)]
#predicted2

confusionmslow <- table(predicted2, testdata$V1 )
confusiondfslow <- as.matrix(confusionmslow)
#confusionmslow

diagslow <- diag(confusiondfslow)
missclassslow <-  1 - ((1/nrow(testdata)) * (sum(diagslow)))
missclassslow
```

```
## [1] 0.5734
```

The missclassification rate is significantly higher than it was before (rose from 0.3198 to 0.5734). It now missclassifies the majority of observations.

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

```r
pairs(confusiondfslow)
```

```
##      letter1 letter2 max
## [1,]       2       4  98
```

B and D are still some of the more difficult letters to distinguish and now the model is even worse at distinguishing them and missclassified more of them.

Communities and Crime (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.)?

```r
set.seed(9)
dfcrime <- read.csv("http://andrewpbray.github.io/data/crime-train.csv")
trainc <- sample(1:nrow(dfcrime), nrow(dfcrime) * .75)
boostcrime=gbm(ViolentCrimesPerPop ~.,data=dfcrime[trainc,], distribution=
"gaussian", n.trees=100, interaction.depth=2, shrinkage = .01)
```

```r
yhatcrime=predict(boostcrime,newdata=dfcrime[-trainc,], n.trees= 50, type = "response")
#yhatcrime
```

```r
MSEtree <- function(yhats, data){
  n <- nrow(data)
  ys <- data$ViolentCrimesPerPop
  residuals <- yhats - ys
  MSE <- sum(residuals^2)/n
  MSE
}
```

```r
MSEtree(yhatcrime, dfcrime[-trainc,])
```

```
## [1] 0.04878183
```

The test MSE for this model is 0.04878. My random forest model had a test MSE of 0.003587 so it preformed better than this model (looking back at this code though I accidently used the same data to train and test it). The regular regression model had a test MSE of 0.01888 so it also preformed better than this model. The single tree model had an MSE of 0.01708644 which is also better than this 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 $P(ClassisRed|X)$:0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, and 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?

By the majority vote criteria we would classify this as red because there are 6 values of probability that the class is red given X that are greater than .5, meaning that the majority suggest that it is red.

Neverthless, by the average probability criteria we would classify it as green. In the chunk below we see that the mean probability that it is red given the specific value of X is .45 which is less than .50 so by this criteria we would classify it as green.

```r
estimates <- c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
mean(estimates)
```

```
## [1] 0.45
```

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

The first step involves partitioning the predictor space. This is done through recursive binary splitting which is a top down approach since we begin with the whole predictor space and then split the space at some cutoff point that decreases the RSS by the most across all the potential cutoff points of all the predictors. Each time a line is drawn across the predictor space two new regions are formed, we continue the splitting process on one of the newly created regions each time until we hit some stopping criterion (such as none of the regions have more than a certain number of observations).

The next step involves predicting a response for a test observation that falls within any of the regions created by our partition, which we compute as the mean of all the training observations within that region.