Lab 8: Ransom notes keep falling

Ben Thomas

11/17/2019

Letter classification

Data

Today, I'll be using boosted trees to do some character regression. To start, I'll download our dataset, which includes a catalog of features extracted from 20,000 different characters.

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

To move forward, I'm going to separate out this dataset into training and test data. I've done so below.

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

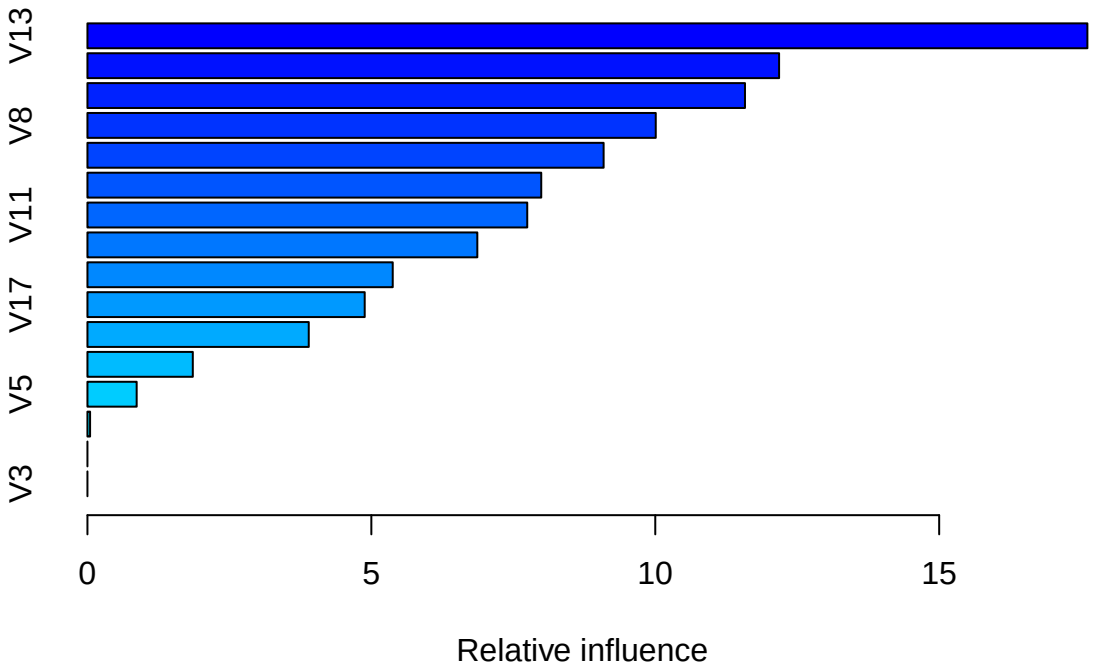
Building a boosted tree

I'll now move forward and build out a boosted classification tree to predict the class (letter) of each of these observations. To do so, I'm using the `gbm` package.

```
library (gbm)
set.seed(1)
boost.characters=gbm(V1 ~ .,
                     data=df_train,
                     n.trees=50,
                     interaction.depth=1,
                     shrinkage = 0.1)
```

```
## Distribution not specified, assuming multinomial ...
```

```
head(summary(boost.characters))
```




```
##     var   rel.inf
## V13 V13 17.610283
## V12 V12 12.180597
## V14 V14 11.580058
## V8   V8 10.008523
## V10 V10  9.090569
## V15 V15  7.991759
```

As we see here, V13 is the most important variable for predicting the letter, followed by V12 and V14.

Assessing predictions

Now that we've got this boosted model, let's use it to predict letters in our test dataset.

First, let's generate the predictions.

```
yhat <- predict(boost.characters,
                df_test,
                n.trees = 50,
                type = "response")
predicted <- LETTERS[apply(yhat, 1, which.max)]
actual <- df_test$V1
```

Now, let's compare these to the actual values in a large "confusion matrix" and find the misclassification rate.

```
conf_matrix <- table(actual, predicted)
conf_matrix
```

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

```
##      F   4   5   6   1   0   1   2   5   1
##      G  13   4   0   0   1   7   0   0   3
##      H   9   1   2  12   0   7   2   0   0
##      I   2   2   0   0   0   0   5   0   0
##      J   7   7   0   0   0   0   0   0   0
##      K  15   0   0   0   0   2  10   2   0
##      L   2   4   0   0   0   0   1   8   0
##      M   1   1   0   0   0   4   0   0   0
##      N   3   0   0   3   1   3   0   5   0
##      O   3   0   0   1   0  10   0   0   0
##      P   0   0   0   0   0   9   0   4   0
##      Q   1   6   0   0   0   1   0   1   0
##      R 151   0   0   0   0   4   3   0   0
##      S  11 121   2   1   0   0  14   1   6
##      T   0   0 137   6  10   0   9  11   2
##      U   0   0   2 145   5   1   1   1   0
##      V   0   1   2   4 130  13   0   4   0
##      W   1   0   0   1   3 139   0   0   0
##      X   1   3   0   0   0   0 126   9   7
##      Y   0   4  11   2  16   3   0 122   0
##      Z   4  17   3   0   0   0   3   0 133
```

```
miss.rate <- mean(actual != predicted)
miss.rate
```

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

The mislassification rate from this model on our test data is 0.3198 – not great. We can actually get a little more in depth with this, and calculate the misclassification rate of our for each letter. I've done so below.

```
miss.rates <- rep(NA, 26)
for (i in 1:26) {
  miss.rates[i] = (
    sum(conf_matrix[i,]) - conf_matrix[i,i])/
    sum(conf_matrix[i,])
}
miss.rates
```

```
##  [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
```

From this, we can see that the letters which are misclassified most frequently are the 5th and 8th letters of the alphabet: E and H. E was misclassified nearly 65% of the time, and most commonly mistaken for X, C, and G. Distinguishing between E and X was the hardest of all combinations for our model, as the 35 "E" observations predicted as "X" was the highest mistake rate of any pair of letters. Following E, H was misclassified 59% of the time, and most commonly mistaken for O.

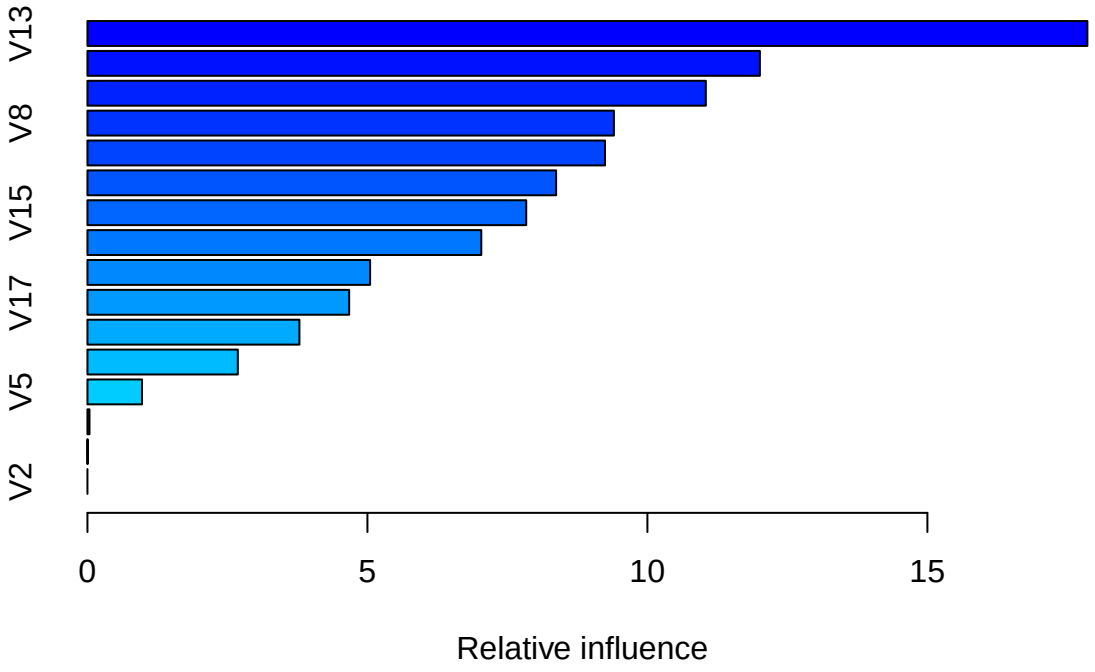
Slow the learning

Let's now change up the parameters of our boosted model to try to lower this misclassification rate. I'll change lambda to 0.033 and increase B to 150.

```
slow.boost.characters=gbm(V1 ~ .,
                          data=df_train,
                          n.trees=150,
                          interaction.depth=1,
                          shrinkage = 0.033)
```

```
## Distribution not specified, assuming multinomial ...
```

```
head(summary(slow.boost.characters))
```




```
##     var   rel.inf
## V13 V13 17.857415
## V12 V12 12.008527
## V14 V14 11.043429
## V8   V8  9.401656
## V11 V11  9.242542
## V10 V10  8.369841
```

Again, V13, V12, and V14 are the most important variables for predicting a character. Let's now find the new misclassification rate and compare it to our original model.

```
slow.yhat <- predict(slow.boost.characters,
                     df_test,
                     n.trees = 150,
                     type = "response")
```

```
slow.predicted <- LETTERS[apply(slow.yhat, 1, which.max)]
slow.miss.rate <- mean(actual != slow.predicted)
slow.miss.rate
```

```
## [1] 0.3232
```

This new misclassification rate, 0.3236, is actually a little worse than the original misclassification rate. We can (again) get a little more granular by calculating the miscalculation rate by letter. I've done so below.

```
slow.conf.matrix <- table(actual, slow.predicted)
slow.miss.rates <- rep(NA, 26)
for (i in 1:26) {
  slow.miss.rates[i] = (
    sum(slow.conf.matrix[i,]) - slow.conf.matrix[i,i])/
    sum(slow.conf.matrix[i,])
}
slow.miss.rates
```

```
##  [1] 0.1372549 0.3043478 0.3225806 0.3349754 0.6553398 0.3807107 0.4226804
##  [8] 0.5876289 0.2331606 0.3193717 0.4022346 0.2525773 0.1594203 0.2403846
## [15] 0.2180851 0.3400000 0.3975904 0.2682927 0.4400000 0.3619048 0.3106796
## [22] 0.2471264 0.1764706 0.3010753 0.3202247 0.2485876
```

Here again, E (the 5th letter) and H (the 8th) are the hardest letters for our model to predict. To dive in more, we can see which letters this slower model is better (or worse) at predicting by finding the change in misclassification rates by letter. Letters where the change is negative imply a lower (better) misclassification rate in the slower model, while those where the change is positive imply that the slower model is worse for that letter.

```
miss.rate.changes <- rep(NA, 26)
for (i in 1:26) {
  miss.rate.changes[i] = (
    slow.miss.rates[i] - miss.rates[i])
}
miss.rate.changes
```

```
##  [1] -0.009803922  0.010869565 -0.010752688 -0.019704433  0.014563107
##  [6] -0.015228426  0.000000000  0.000000000  0.000000000  0.015706806
## [11]  0.016759777  0.005154639  0.014492754 -0.009615385  0.015957447
## [16]  0.005000000  0.000000000  0.004878049  0.045000000  0.014285714
## [21]  0.014563107 -0.005747126 -0.005882353 -0.021505376  0.005617978
## [26]  0.000000000
```

As we'd expect from the overall higher misclassification rate, the new model is worse for the majority of the letters. Some notable highlights: the slower model misclassifies about 2% more "J"s, 1.5% more "E"s, and 2% more "P"s than the original model did.

Communities and crime

Now, I'll return once more to the crime dataset, which has been put to very good use so far this semester.

Data

First, let's download the data (both training and test).

```
crime.train <- read.csv("http://andrewpbray.github.io/data/crime-train.csv")
crime.test <- read.csv("https://bit.ly/2PYS8Ap")
```

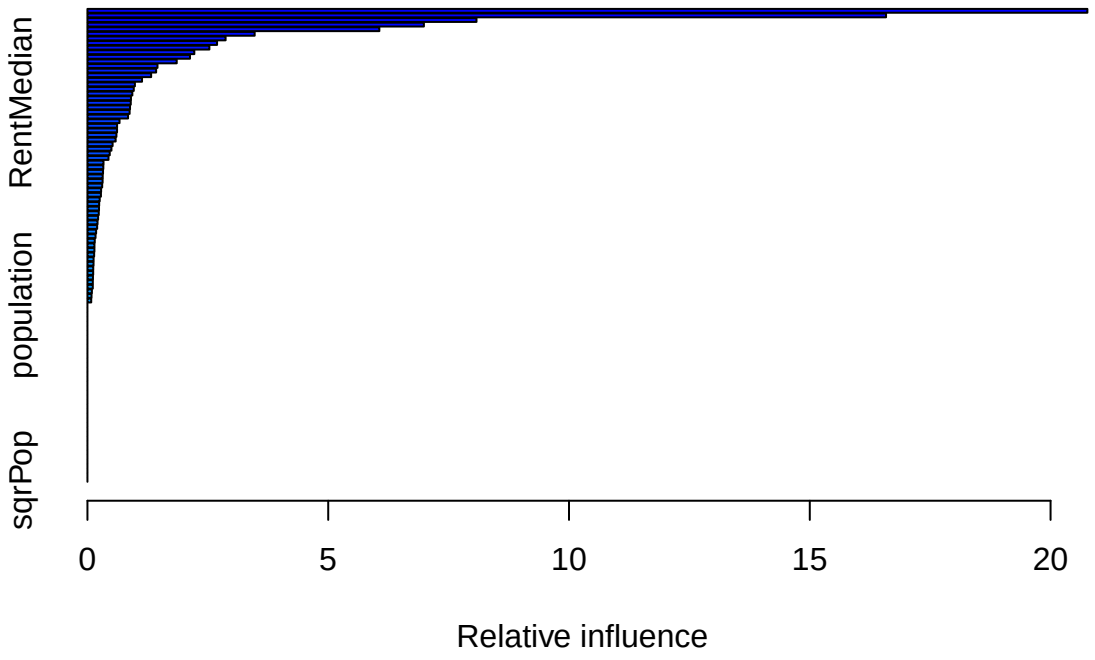
In order to compare the boosting method on this dataset to previous methods (ridge, lasso, bagging, random forests, etc.), I'll first need to get the data in comparable format. I'll therefore transform the data to match my earlier transformations.

```
library(tidyverse)
# Training data transformation
crime.train <- crime.train %>%
  mutate(sqrInvInc = (100*pctWInvInc)^2,
         sqrPop = population^2)

crime.train.section <- select_if(crime.train, is.numeric)

# Test data transformation
crime.test <- crime.test %>%
  mutate(sqrInvInc = (100*pctWInvInc)^2,
         sqrPop = population^2)

crime.test.section <- select_if(crime.test, is.numeric)
```

One last boost

Now, let's create a boosted regression tree to predict crime, and compare its MSE to prior models. Note that I've used the parameters lambda = 0.1 and B = 200.

```
crime.boost <- gbm(ViolentCrimesPerPop ~ .,
                   data= crime.train.section,
                   distribution = "gaussian",
                   n.trees= 200,
                   interaction.depth=1,
                   shrinkage = 0.1)
head(summary(crime.boost))
```




```
##                           var   rel.inf
## PctIlleg             PctIlleg 20.765027
## PctKids2Par       PctKids2Par 16.583966
## racePctWhite     racePctWhite  8.079275
## NumIlleg             NumIlleg  6.986855
## PctFam2Par         PctFam2Par  6.061541
## MalePctDivorce MalePctDivorce  3.472713
```

Now, let's estimate the test MSE of this boosted model using the `crime.test.section`.

```
MSE.crime.boost <- mean((crime.test.section$ViolentCrimesPerPop -
                         predict(crime.boost,
                                 crime.test.section,
                                 n.trees = 200,
                                 type = "response")) ^ 2)
MSE.crime.boost
```

```
## [1] 0.01259784
```

The estimated test MSE of our new model is therefore 0.01253594. While good (better than any of the lasso, ridge, or linear models), this is still significantly higher than that of the random forests model, which had a MSE of 0.003113876.

Chapter 8 exercises

Problem 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, 0.15, 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?

Majority vote

Using the majority vote approached discussed in Chapter 8, we would determine the prediction from each of the probabilities. The final classification would be the value most frequently predicted.

In this case, we have the following predictions (where if P > .5, the prediction is red):

1. green
2. green
3. green
4. green
5. red
6. red
7. red
8. red
9. red
10. red

The majority of the samples generate a predictiont that this specific value is **red**, so this is our final classification.

Average probability

Using the average probability, we would take the mean of the probabilities as our final probability estimate. If the final probability estimate is above 0.5, the final classification would be red; else, it would be green.

The average of the 10 probabilities above is 0.45, so in this case, the final prediction would be **green**.

Problem 6

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

The algorithm that is used to fit a regression tree proceeds as follows.

1. Divide the observations into two sections with different predictions (mean response values), such that the RSS (given by the equation below) is minimized by the split. The two sections are ranges of the X_j space, separated by the cutpoint s.

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

2. Repeat step 1 for one of the new sections, such that RSS is minimized by the new split.
3. Continue splitting the new sections until some stopping criterion is reached.