Lab 7: When a guest arrives they will count how many sides it has on

Ben Thomas

11/11/2019

Data

Today, we'll be exploring regression trees. First, we should access our dataset. In this case, we'll use a self-generated data set.

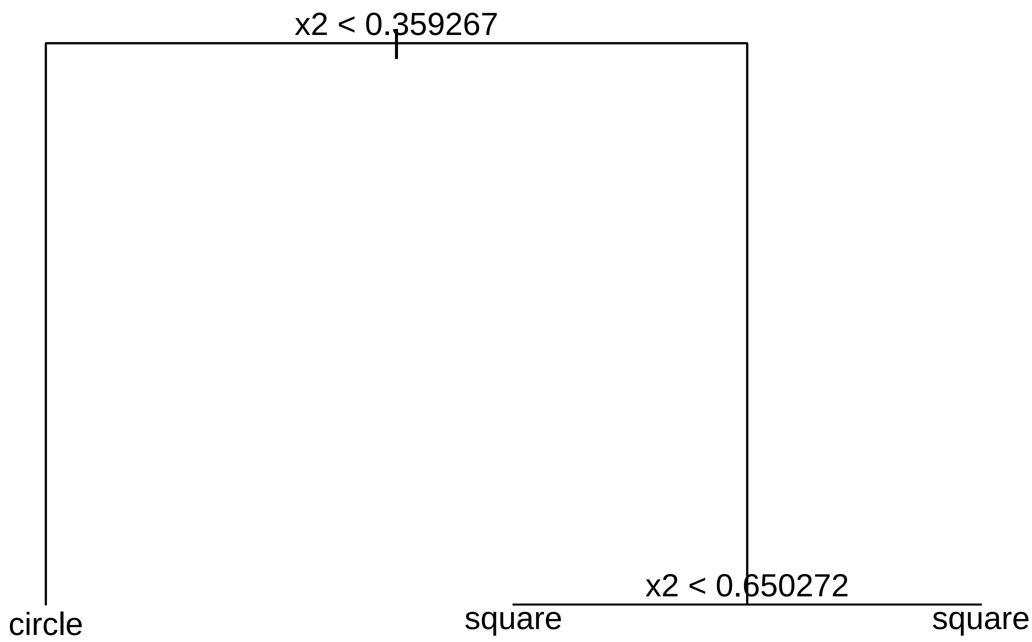
```
set.seed(75)
n <- 16
x1 <- runif(n)
x2 <- runif(n)
group <- as.factor(sample(1:3, n, replace = TRUE))
levels(group) <- c("circle", "triangle", "square")
df <- data.frame(x1, x2, group)
df[1, 2] <- .765 # tweaks to make a more interesting configuration
df[9, 1] <- .741
df <- df[-7, ]
```

Part 1: Growing the full classification tree

Let's start by using the `tree` package in R to create a regression tree on this dataset making splits based on the *Gini index*. The resulting tree is displayed below.

```
library(tree)
t1 <- tree(group ~.,
           data = df,
           split = "gini")
```

```
plot(t1)
text(t1, pretty = 0)
```



a. Neither of the most common splits created in class were the first one made by the regression tree package.

b. The benefit of the second split in the tree is not easy to understand at first; after all, the same category is applied to both sides of the split. However, this split allows for increased nodal purity, thus increasing the gini index.

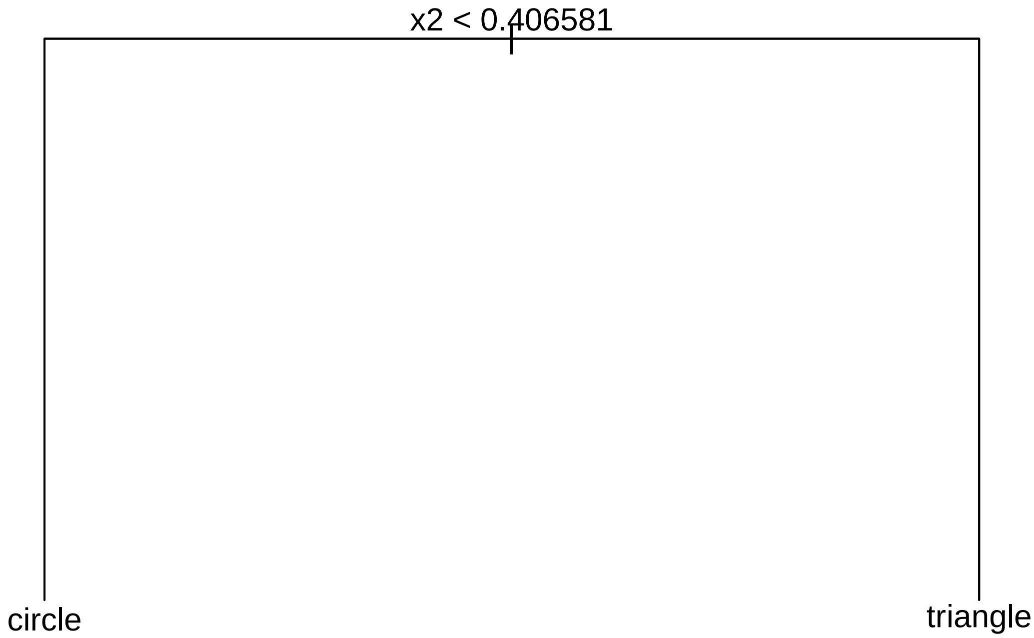
c. This model would predict that an observation at (0.21, 0.56) would be a square.

Part 2: An alternate metric

Let's try creating a new regression tree using *deviance* as the deciding factor for splitting, rather than the *Gini index*.

```
library(tree)
t2 <- tree(group ~.,
           data = df,
           split = "deviance")
```

```
plot(t2)
text(t2, pretty = 0)
```



This regression tree is significantly different from the first one, and actually does not predict that any observations would be squares. This is because there is a different metric determining where the optimal split is.

Crime and Communities, revisited

In lab 3, we used linear regression to predict crime in a given community. Let's try to use regression trees instead!

Part 3: Growing a pruned regression tree

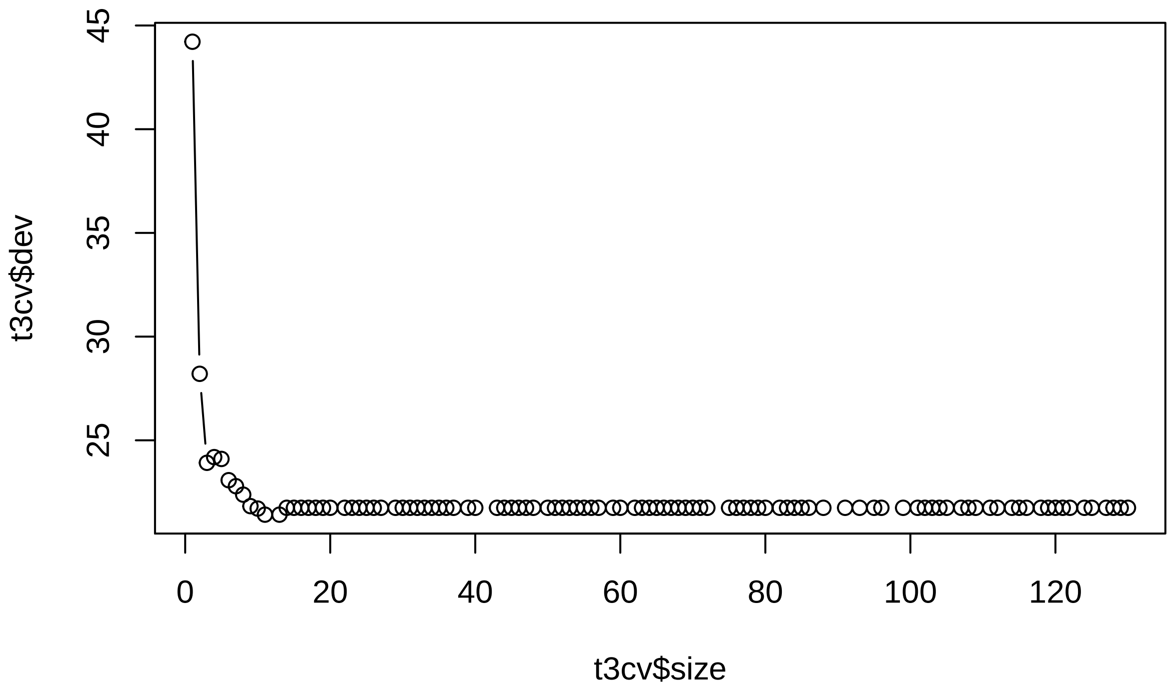
First, let's get the training data and edit it into a useable format.

```
# First, let's download the training data
training_data <- read.csv("http://andrewpbray.github.io/data/crime-train.csv")

# Now let's change the data in the same way we did in Lab 3
library(tidyverse)
training_data <- training_data %>%
    mutate(sqrInvInc = (100*pctWInvInc)^2,
           sqrPop = population^2)

# Let's drop non-numeric variables
training_data_section <- select_if(training_data, is.numeric)
```

Now, let's make a pruned regression tree, and see what size of pruned tree results in the least deviance.

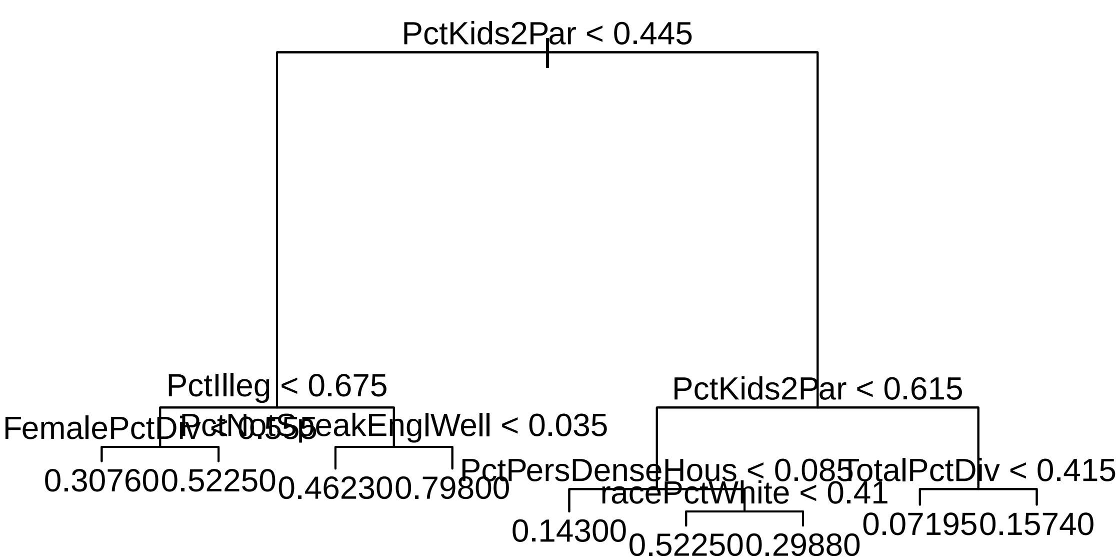
```
# Fit a regression tree
t3 <- tree(ViolentCrimesPerPop ~ .,
           data = training_data_section,
           split = "gini")
```

```r
# Make a pruned tree
t3cv <- cv.tree(t3, FUN = prune.tree)

# Plot the deviance against the size of the tree
plot(t3cv$size, t3cv$dev, type = "b")
```



As we can see here, the size of the tree that results in the least deviance is 9. This tree (with size 9) is shown below, though it is quite unappealing (sorry!).

```r
t3prune <- prune.tree(t3, best = 9)
plot(t3prune)
text(t3prune, pretty = 0)
```



Part 4: Comparing predictive performance

Let's use this best tree to compute the test MSE.

```r
# First, let's download, modify, and clean the test data
test_data <- read.csv("https://bit.ly/2PYS8Ap")
test_data <- test_data %>%
  mutate(sqrInvInc = (100*pctWInvInc)^2,
         sqrPop = population^2)

test_data_section <- select_if(test_data, is.numeric)

# Now, let's compute the test MSE
MSE_tree <- mean((test_data_section$ViolentCrimesPerPop - predict(t3prune, test_data_section)) ^ 2)
MSE_tree
```

```
## [1] 0.01885504
```

We see here that the test MSE of this model is 0.01885504. Let's compare this to the test MSE of the model I used in Lab 3.

```r
m1 <- lm(ViolentCrimesPerPop ~ population + sqrPop + log(medIncome) + PctHousOccup +
             NumInShelters + PctKids2Par + pctWInvInc + sqrInvInc + PctPersDenseHous +
             racePctWhite + PctWorkMomYoungKids, data = training_data_section)

MSE_lm <- mean((test_data_section$ViolentCrimesPerPop - predict.lm(m1, test_data_section)) ^ 2)
MSE_lm
```

```
## [1] 0.01927444
```

The MSE of the linear model I created in Lab 3 is 0.01927444, ever-so-slightly higher than that of the best regression tree model.

Part 5: Growing a random forest

Let's now try to decrease the variance of this prediction. First, let's use bagging (with the random forests package, where m = p).

```r
library(randomForest)
set.seed(1)
bag.train = randomForest(ViolentCrimesPerPop ~ .,
           data = training_data_section,
           mtry = 103,
           importance = TRUE)
```

```r
yhat.bag <- predict(bag.train, newdata = test_data_section)
MSE_bag <- mean((test_data_section$ViolentCrimesPerPop - yhat.bag) ^ 2)
MSE_bag
```

```
## [1] 0.003130023
```

Wow! The MSE of the bagging method is 0.003130023, *significantly* lower than that of the linear model or of the best tree.

Now, let's try a true random forests model, where m = 1/3 p (or roughly 34).

```
rf.train = randomForest(ViolentCrimesPerPop ~ .,
            data = training_data_section,
            mtry = 34,
            importance = TRUE)
```

```
yhat.rf <- predict(rf.train, newdata = test_data_section)
MSE_rf <- mean((test_data_section$ViolentCrimesPerPop - yhat.rf) ^ 2)
MSE_rf
```

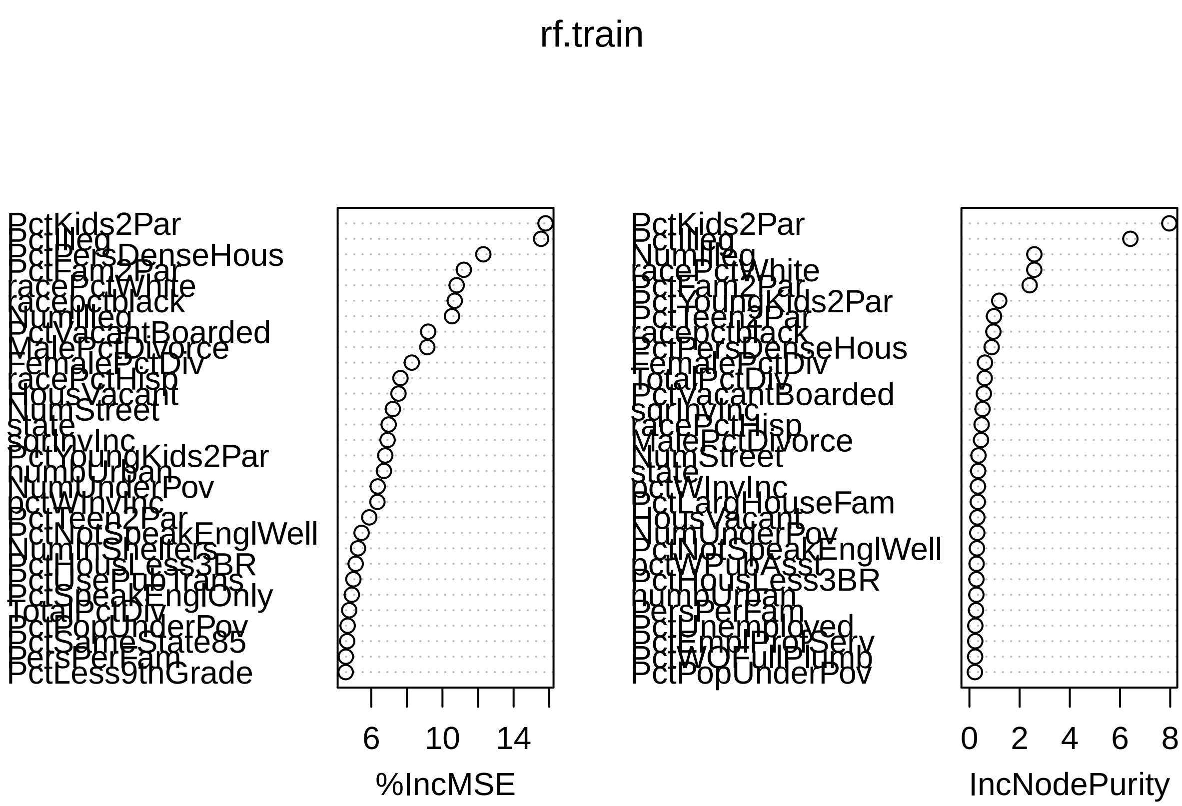
```
## [1] 0.003113876
```

The MSE of the random forests model is even lower than that of the bagged model: 0.003113876.

Part 6: Variance importance

Let's try to salvage some interpretability from the random forest calculation. We'll do this using `importance()` and `varImpPlot()`. For your sake, I'm not displaying the `importance()` term: it's quite large.

```
#importance(rf.train)
varImpPlot(rf.train)
```



As we can see here, the most important variables in this random forest are `PctKids2Par` and `PctIlleg`, with `PctPersDenseHous`, `PctFam2Par`, `racePctWhite`, `racepctblack`, and `numIlleg` following behind closely.

This is a bit different from the results from the model. I've displayed a summary of the first linear model below; note that the most important (read: significant) variables here are `medIncome`, `PctKids2Par`, `pctWInvInc`, and `racePctWhite`. Our model didn't even include `PctIlleg`, `PctFam2Par`, `racepctblack`, or `numIlleg`.

```r
summary(m1)
```

```
##
## Call:
## lm(formula = ViolentCrimesPerPop ~ population + sqrPop + log(medIncome) +
##     PctHousOccup + NumInShelters + PctKids2Par + pctWInvInc +
##     sqrInvInc + PctPersDenseHous + racePctWhite + PctWorkMomYoungKids,
##     data = training_data_section)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46658 -0.07734 -0.01559  0.04443  0.75172
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          1.067e+00  6.689e-02  15.958  < 2e-16 ***
## population           3.186e-01  1.157e-01   2.753 0.006036 **
## sqrPop              -3.289e-01  1.253e-01  -2.624 0.008851 **
## log(medIncome)       7.466e-02  1.518e-02   4.920 1.05e-06 ***
## PctHousOccup        -8.128e-02  2.954e-02  -2.751 0.006069 **
## NumInShelters        2.460e-01  8.033e-02   3.062 0.002271 **
## PctKids2Par         -6.548e-01  5.614e-02 -11.662  < 2e-16 ***
## pctWInvInc          -6.507e-01  1.794e-01  -3.627 0.000305 ***
## sqrInvInc            4.540e-05  1.507e-05   3.013 0.002668 **
## PctPersDenseHous     7.940e-02  3.870e-02   2.051 0.040554 *
## racePctWhite        -1.380e-01  4.067e-02  -3.394 0.000723 ***
## PctWorkMomYoungKids -2.153e-02  3.226e-02  -0.667 0.504650
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1399 on 788 degrees of freedom
## Multiple R-squared:  0.6504, Adjusted R-squared:  0.6455
## F-statistic: 133.2 on 11 and 788 DF,  p-value: < 2.2e-16
```