512: Assignment #8

Brody Vogel

4/9/2018

Preparation

```
set.seed(1234)
library(ISLR)
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.4.4
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.
```

```
library(gbm)
```

```
## Loading required package: survival

## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

Problem 8.1

R1, R2, etc. refer to the regions; T1, T2, etc. refer to the cutpoints.

Problem 8.4

a)
b)

Problem 8.9

a)

```
attach(OJ)

train <- sample(1:nrow(OJ), 800)
OJ.test <- OJ[-train, ]
```

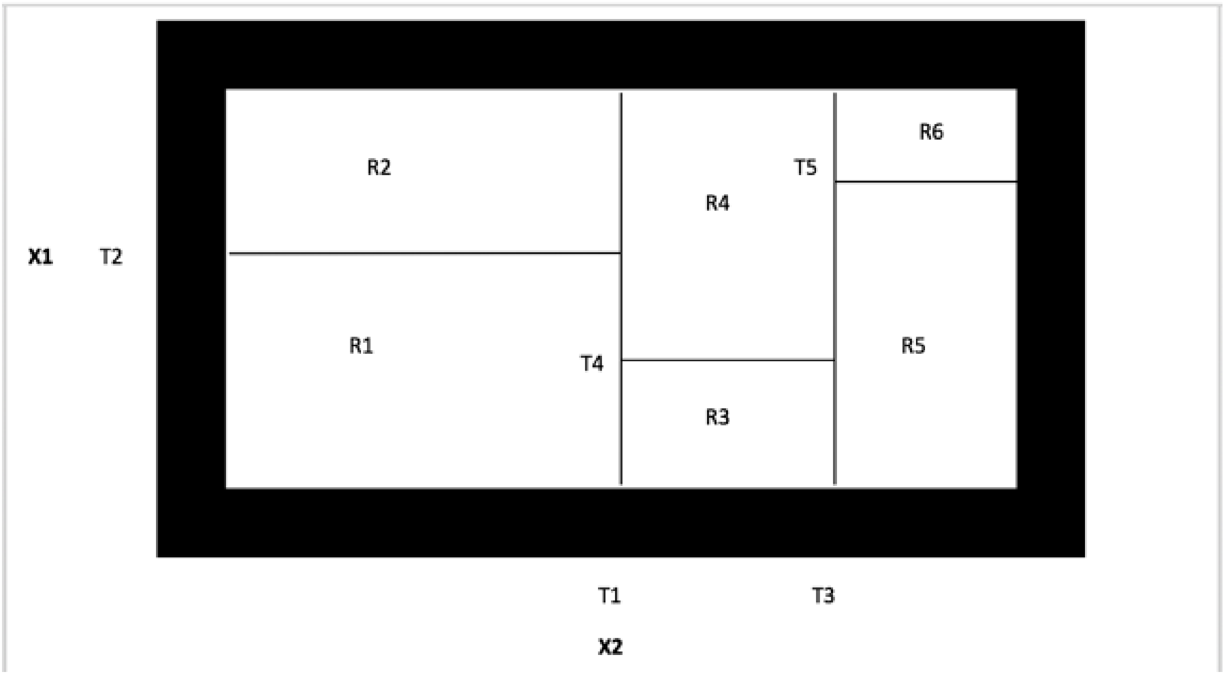


Figure 1: Decision Tree Diagram

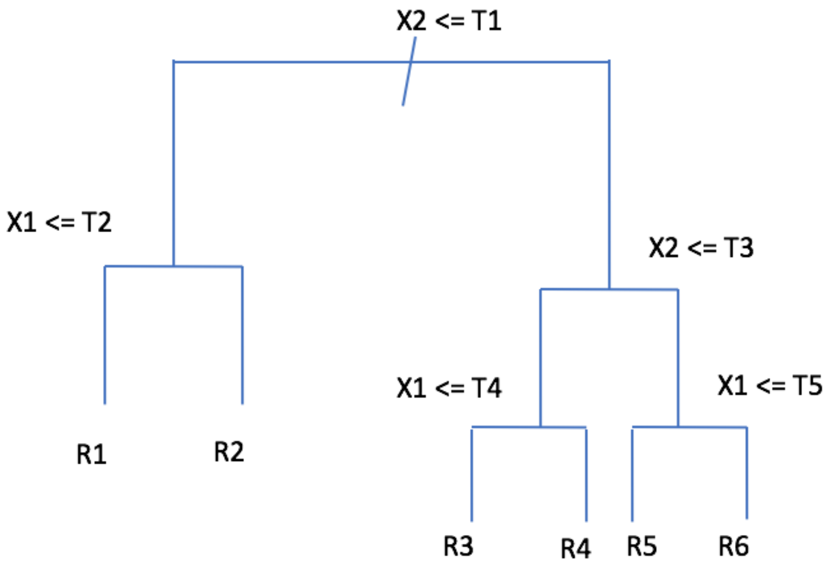


Figure 2: Decision Tree Diagram

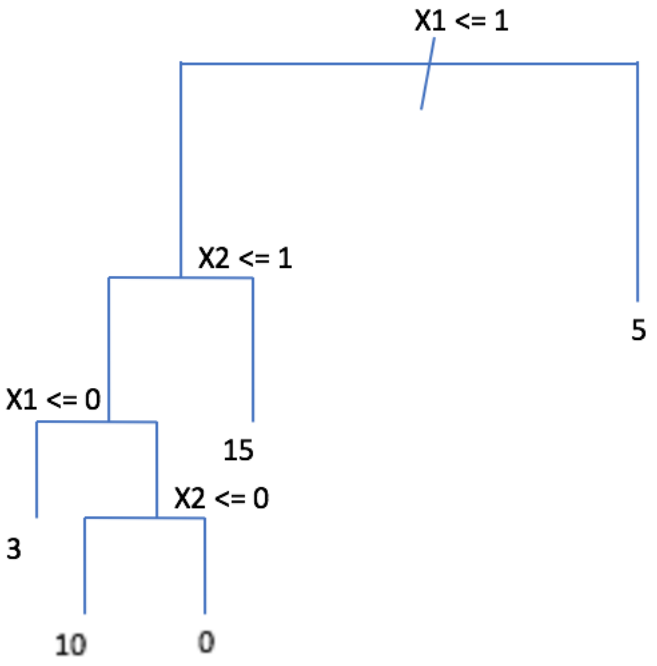


Figure 3: Decision Tree Diagram

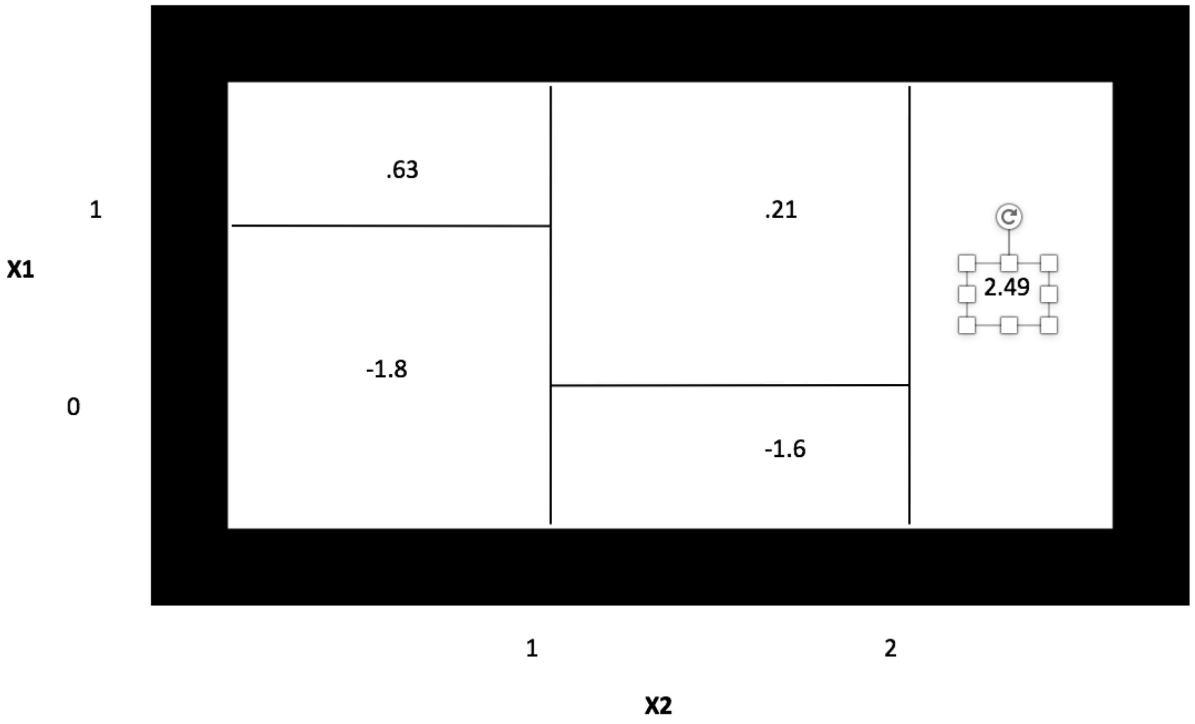


Figure 4: Decision Tree Diagram

b)

The method produced an error rate of .1538 with 7 terminal nodes. The tree was constructed using only the variables "LoyalCH", "PriceDiff", and "ListPriceDiff", which is a bit surprising.

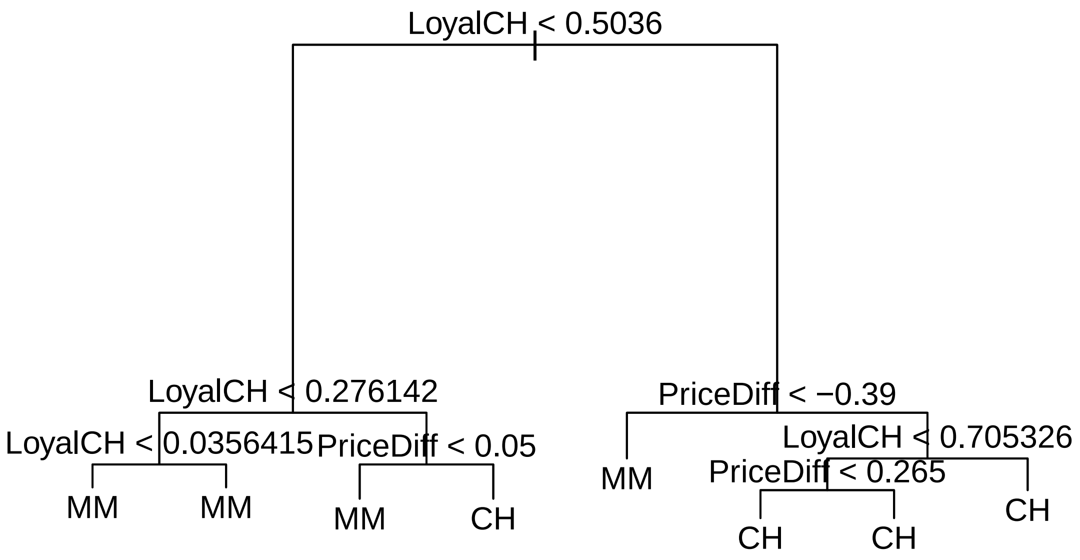
```r
OJTree <- tree(Purchase~., data = OJ[train, ])
summary(OJTree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ[train, ])
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7573 = 599.8 / 792
## Misclassification error rate: 0.1688 = 135 / 800
```

d)

From the plot, it looks like "LoyalCH" is by far the most important variable; it accounts for the top three splits. For values of "LoyalCH" less than .276142, the tree predicts class MM; for values above .764572, class CH. When the value of LoyalCH is between those two values, the tree splits on values of "PriceDiff" and "ListPriceDiff" to decide whether to classify each observation as MM or CH.

```r
plot(OJTree)
text(OJTree, pretty = 0)
```



e)

I got a test error rate of about .193 percent.

```r
tree.pred <- predict(OJTree, OJ.test, type = "class")
table(tree.pred, OJ.test$Purchase)
```

```
##
## tree.pred  CH  MM
##        CH 152  35
##        MM  12  71
```

f)

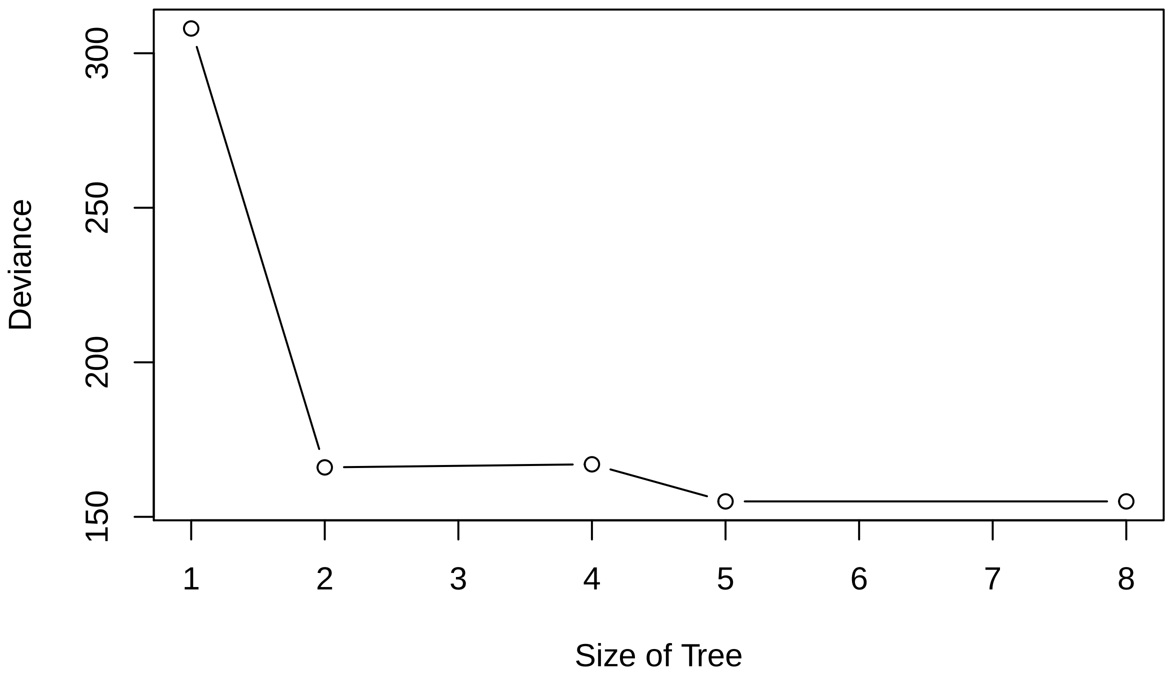
Looks like 5 is the best size of the tree.

```
cv.OJ <- cv.tree(OJTree, FUN = prune.misclass)
cv.OJ
```

```
## $size
## [1] 8 5 4 2 1
##
## $dev
## [1] 155 155 167 166 308
##
## $k
## [1] -Inf    0    6   10  150
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

g)

```
plot(cv.OJ$size, cv.OJ$dev, type = 'b', xlab = 'Size of Tree', ylab = 'Deviance')
```



h)

Trees with both 7 and 5 nodes produced a deviance of 138, so trees of those sizes corrsponded to the lowest cross-validated classification error rate.

i)

```
prune.OJ <- prune.misclass(OJTree, best = 5)
```

j)

From above, the regular tree produced a misclassification error rate of .1538. The pruned tree, here, produced a misclassification error rate of .1575. So, actually, the pruned tree produced a higher misclassification rate in this case.

```r
#summary(OJTree)
summary(prune.OJ)
```

```
##
## Classification tree:
## snip.tree(tree = OJTree, nodes = c(4L, 7L))
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  5
## Residual mean deviance:  0.8381 = 666.3 / 795
## Misclassification error rate: 0.1688 = 135 / 800
```

k)

Again from above, I got a test error rate of about .193 percent for the regular tree. For the pruned tree, as can be seen here, I got a test error rate of about .189 percent. So, as opposed to the misclassification rate in (j), the regular tree produced a slightly higher test error rate in this case (it actually only missed one more than the pruned tree, so they're really equivalent).

```r
#tree.pred <- predict(OJTree, OJ.test, type = "class")
#table(tree.pred, OJ.test$Purchase)

tree.pred <- predict(prune.OJ, OJ.test, type = "class")
table(tree.pred, OJ.test$Purchase)
```

```
##
## tree.pred  CH  MM
##        CH 152  35
##        MM  12  71
```

Problem 12 with the MNIST data

I used 24 random predictors from the mnist68 dataset. I then transformed the response - "labels" - column of the data to a factor (a "6" corresponds to a 1; an "8" to a 0).

When I tried to predict whether a number was a 6 or 8 using logistic regression, I got an error rate of about .078 percent, which I thought was pretty good.

Next I tried random forest. This produced an even lower error rate of .046 percent; really good, I thought.

Finally, I tried boosting with 5000 trees. This produced an error rate, in between the last two, of about .056.

In conclusion, then, random forest produced the best results for me. It improved on the results of logistic regresiion, and performed better than boosting. The lowest error rate, produced by random forest, was about .046. I should note, though, that when I reran the program, the results changed pretty significantly. Based on the random subset of predictors the code chose, sometimes the error rates shot up over 13 percent, and occasionally logistic regression outperformed boosting; random forest, though, always seemed to perform best.

```r
load("/Users/brodyvogel/Downloads/mnist68.RData")

chosen <- sample(1:784, 24, replace = FALSE)

reduced <- mnist68[, c(chosen, 785)]

reduced$labels <- ifelse(reduced$labels == 6, 1, 0)
```

```
train <- sample(1:nrow(reduced), 9000, replace = FALSE)

reduced.test <- reduced[-train, ]

head(reduced.test)
```

```
##   V120 V314 V65 V514 V375 V13 V751 V135 V358 V435 V605 V161 V105 V218 V318
## 1    0    0   0  173  253   0    0    0   27  252    0    0    0    0  140
## 2    0    0   0  253    0   0    0    0    0  254    0  203    0  188    0
## 3    0    0   0   68  152   0    0    0    0  151  217    0    0    0    0
## 5    0    0   0   35  227   0    0    0    0    0  152    0    0    0    0
## 7    0    0   0  217  253   0    0    0   17    0  106    0    0    0    0
## 9    0    0   0   20  195   0    0    0    0  254   49    0    0    0  214
##   V773 V549 V440 V277 V419 V599 V547 V577 V680 labels
## 1    0  233  215    0    0  253  252  148    0      1
## 2    0    0    0    0    0    5  253    0   17      0
## 3    0    0  193    0    0  107    0    5    0      1
## 5    0   73  152    0    0    0    0  254    0      1
## 7    0  253   94    0    0   30  253  253    0      1
## 9    0  100    0    0    0  254    0   23    0      0
```

```
# logistic regression performs pretty well

log.fit <- glm(labels ~., data = reduced[train, ], family = 'binomial')
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
log.probs <- predict(log.fit, newdata = reduced.test, type = 'response')
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
log.preds <- rep(0, length(log.probs))
log.preds[log.probs > .5] <- 1
mean(log.preds != reduced.test$labels)
```

```
## [1] 0.1105092
```

```
# random forest performs even better
                                  # round up the square root of 24, the number or predictors
rf <- randomForest(factor(labels) ~., data = reduced, subset = train, mtry = 5)
rf.preds <- predict(rf, newdata = reduced.test)
mean(rf.preds != reduced.test$labels)
```

```
## [1] 0.06356085
```

```
# boosting falls between logistic regression and random forest

boost <- gbm(labels~., data = reduced[train,], n.trees = 5000)
```

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

```
## Warning in gbm.fit(x, y, offset = offset, distribution = distribution, w =
## w, : variable 16: V773 has no variation.
```

```
boost.preds <- predict(boost, newdata = reduced.test, n.trees = 5000)
boost.preds[boost.preds > 0.5] = 1
boost.preds[boost.preds < .5] = 0
mean(boost.preds != reduced.test$labels)
```

```
## [1] 0.1408451
```