Homework 3

Tim Mastny

February 21, 2018

1

The outside summation means that the total variance by class is scaled by the proportion observations within that region. This means that regions with a large number of observations are weighted more heavily. This is useful for growing trees, as it biases towards regions with many observations. This is likely to generalize better, and makes the trees easier to interpret.

2

Chapter 8.4: 4

(a)

(b)

Chapter 8.4: 5

First, the majority vote says the class of X is red.

```
probs <- c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
length(probs[probs > 0.5]) > length(probs)/2
```

```
## [1] TRUE
```

However, taking the average vote, we conclude the class of X is not red.

```
mean(probs) > 0.5
```

```
## [1] FALSE
```

```
mean(probs)
```

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

Chpater 8.4: 9

(a)

```
library(ISLR)
oj <- ISLR::OJ

set.seed(73)
n <- nrow(oj)
rand_samples <- sample(n, 800, replace = FALSE)
oj_training <- oj[rand_samples,]
oj_test <- oj[-rand_samples,]
```

(b)

```
library(tree)
tree_mod <- tree(Purchase ~ ., data = oj_training)
summary(tree_mod)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = oj_training)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff" "STORE"
## Number of terminal nodes:  10
## Residual mean deviance:  0.6963 = 550.1 / 790
## Misclassification error rate: 0.15 = 120 / 800
```

We have the 10 nodes with error rate of 0.15.

(c)

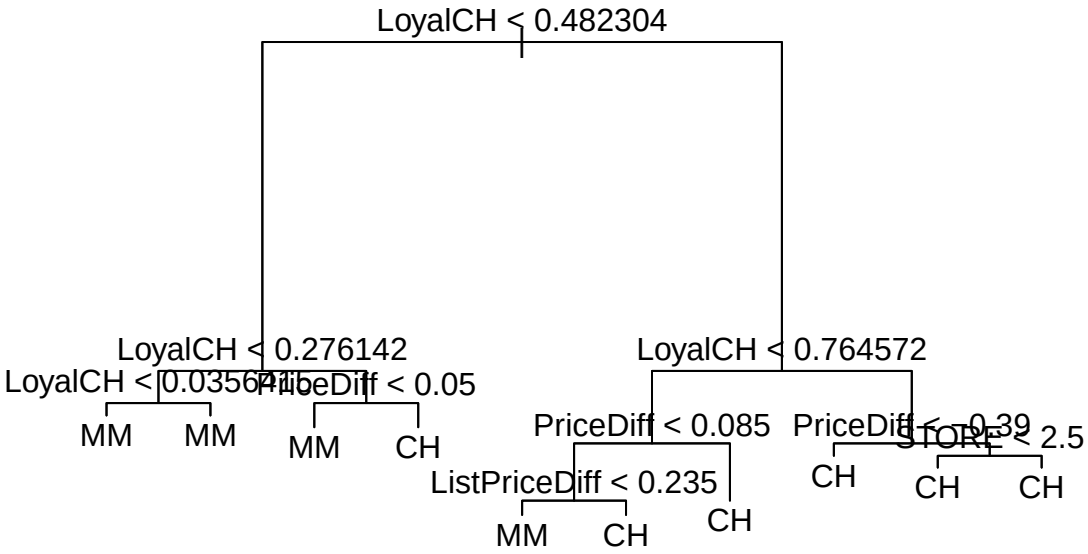
```
tree_mod
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1071.00 CH ( 0.60875 0.39125 )
##    2) LoyalCH < 0.482304 306  333.90 MM ( 0.23529 0.76471 )
##      4) LoyalCH < 0.276142 174  128.20 MM ( 0.12069 0.87931 )
##        8) LoyalCH < 0.0356415 58   10.10 MM ( 0.01724 0.98276 ) *
##        9) LoyalCH > 0.0356415 116  106.60 MM ( 0.17241 0.82759 ) *
##      5) LoyalCH > 0.276142 132  176.10 MM ( 0.38636 0.61364 )
##       10) PriceDiff < 0.05 56   49.38 MM ( 0.16071 0.83929 ) *
##       11) PriceDiff > 0.05 76  104.50 CH ( 0.55263 0.44737 ) *
##    3) LoyalCH > 0.482304 494  434.30 CH ( 0.84008 0.15992 )
##      6) LoyalCH < 0.764572 248  297.00 CH ( 0.71371 0.28629 )
##       12) PriceDiff < 0.085 94  129.20 MM ( 0.44681 0.55319 )
##         24) ListPriceDiff < 0.235 63   78.74 MM ( 0.31746 0.68254 ) *
##         25) ListPriceDiff > 0.235 31   37.35 CH ( 0.70968 0.29032 ) *
##       13) PriceDiff > 0.085 154  115.10 CH ( 0.87662 0.12338 ) *
##      7) LoyalCH > 0.764572 246   70.55 CH ( 0.96748 0.03252 )
##       14) PriceDiff < -0.39 8   10.59 CH ( 0.62500 0.37500 ) *
##       15) PriceDiff > -0.39 238   48.52 CH ( 0.97899 0.02101 )
##         30) STORE < 2.5 156    0.00 CH ( 1.00000 0.00000 ) *
##         31) STORE > 2.5 82   37.66 CH ( 0.93902 0.06098 ) *
```

The first temrinal node we see is number 8. Purchases of class `MM` are binned here, whenever `LoyalCH < 0.0356415` (and all the proceeding criteria) are true.

(d)

```
plot(tree_mod)
text(tree_mod)
```



Seeing the tree plot like this is very useful: most importantly, we see that the first branch almost entirely seperates the `MM` and `CH` factors. There is only one terminal node after the first branch for both classes. Thus, `LoyalCH < 0.482304` seems like a very useful criteria for classification.

(e)

```
pred <- predict(tree_mod, oj_test, type = 'class')
t <- table(pred, oj_test$Purchase)
t
```

```
##
## pred  CH  MM
##   CH 147  37
##   MM  19  67
```

Which gives us a test error rate of:

```
sum(c(t[1,1], t[2,2]))/sum(t)
```

```
## [1] 0.7925926
```

(f)

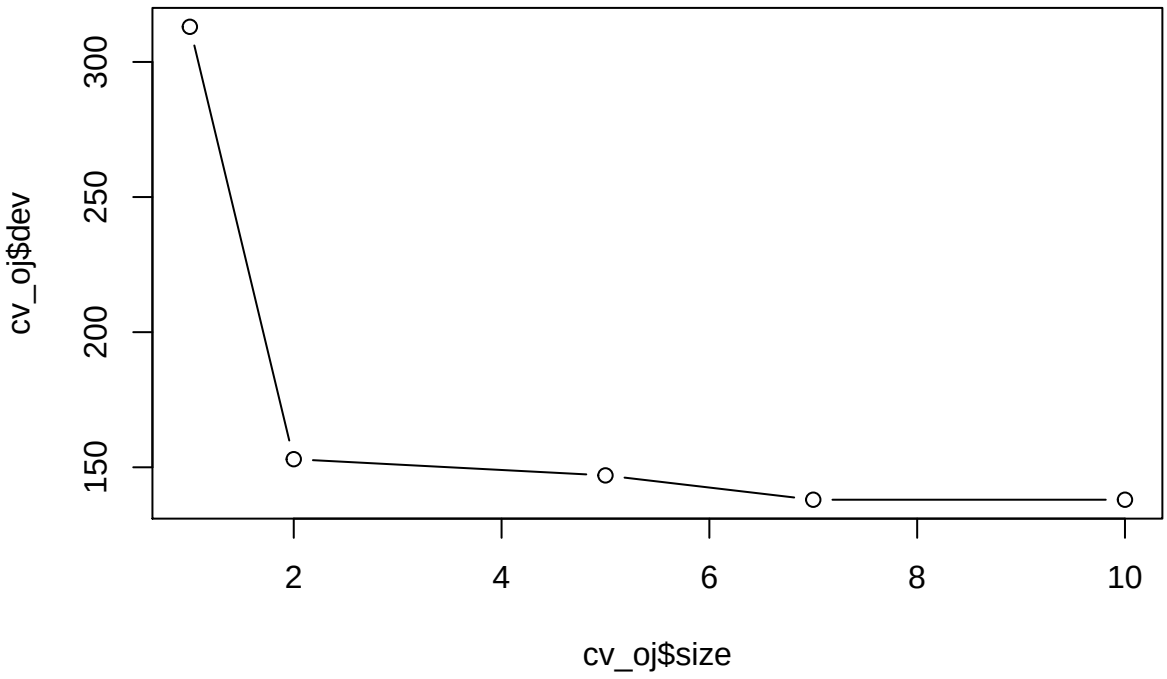
```
cv_oj <- cv.tree(tree_mod, FUN = prune.misclass)
cv_oj
```

```
## $size
## [1] 10  7  5  2  1
##
## $dev
## [1] 138 138 147 153 313
##
## $k
## [1]       -Inf   0.000000   4.000000   7.666667 162.000000
##
## $method
```

```
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

(g)

```
plot(cv_oj$size, cv_oj$dev ,type="b")
```



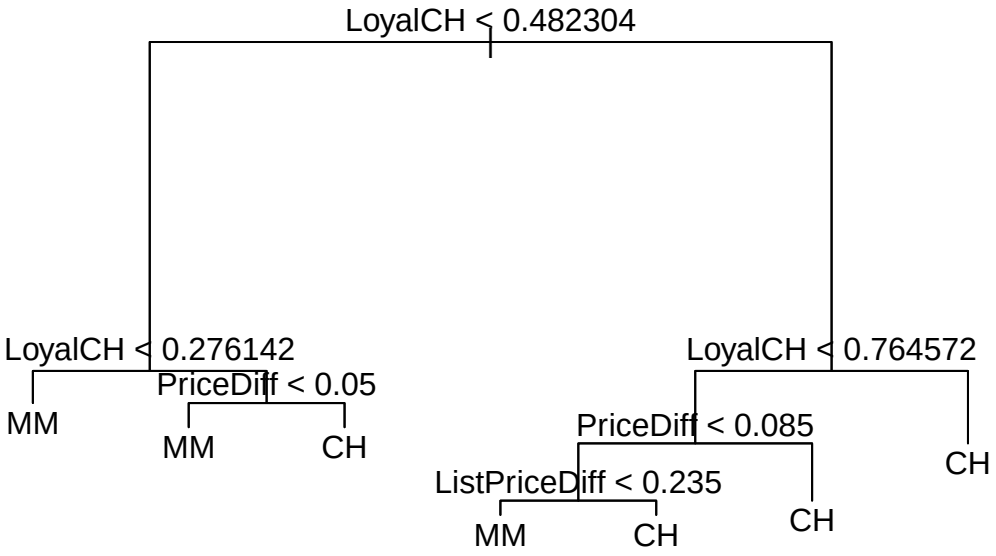
(h)

So the initial model with 10 nodes is tied for the best cross-validation error with a 7 node tree. To reduce complexity, it may be preferrable to choose the 7 node tree model.

(i)

```
pruned_tree <- prune.misclass(tree_mod, best = 7)
```

```
plot(pruned_tree)
text(pruned_tree)
```



(j)

```
summary(pruned_tree)
```

```
##
## Classification tree:
## snip.tree(tree = tree_mod, nodes = c(4L, 7L))
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7362 = 583.8 / 793
## Misclassification error rate: 0.15 = 120 / 800
```

Here the training error rates are equivalent.

(k)

```
pred <- predict(pruned_tree, oj_test, type = 'class')
table(pred, oj_test$Purchase)
```

```
##
## pred  CH  MM
##   CH 147  37
##   MM  19  67
```

Which gives us a test error rate of:

```
(147 + 67)/(147 + 19 + 37 + 67)
```

```
## [1] 0.7925926
```

The error rate is exactly the same, which is not surprisingly considering the cross-validation error was the same.

However, a smaller tree is easier to interpret and since it performs well on the test set, it makes sense to prefer the pruned tree.

3

```
spam = read.csv("http://thinktostart.com/data/data.csv",header=FALSE,sep=";")
names(spam) = read.csv("http://thinktostart.com/data/names.csv",
                       header=FALSE,sep=";",stringsAsFactors=FALSE)$V1
spam$y = factor(spam$y)
names(spam)[49:54] = paste0("char_freq_", 1:6)

set.seed(98533795)
```

```
library(rsample)
library(purrr)
library(magrittr)
spam_split <- initial_split(spam, prop = 3/4)
```

rpart

```
library(rpart)
growtree = rpart.control(minsplit = 2, minbucket = 0, cp = 0)
rpart_mod = rpart(y ~ ., data = training(spam_split), control = growtree)
```

bagging

We'll use the number of bootstrap sets to be $B = 100$, as per the example on page 317.

replace with `randomForest`

```
library(randomForest)
boot_num <- ncol(training(spam_split)) - 1
bag_mod <- randomForest(y ~ ., data = training(spam_split),
             mtry = boot_num, ntrees = 500)
```

C5.0

```
library(caret)
c50_mod <- train(y ~ ., data = training(spam_split), method = "C5.0")
```

Weighted Sum

Note that I am choosing to evaluating this model (and the models in the next section) based on MSE. Since we are considering a weighted sum, we might get predicted outcome like `0.2` if the individual predictions are `0, 1, 1` with weights `0.8, 0.1, 0.1`. So I think it makes the most sense to use MSE in this instance.

```
vec <- c(.1, .2, .3, .4, .5, .6, .7, .8, .9)
weights <- list()
i = 1
for (v1 in vec) {
  for (v2 in vec) {
    for (v3 in vec) {
      if ((v1 + v2 + v3) == 1) {
```

```
        weights[[i]] <- c(v1, v2, v3)
        i <- i + 1
      }
    }
  }
}
```

```
cv <- vfold_cv(training(spam_split), v = 6)
```

```
weight_pred <- function(data, models, weight) {
  ave_pred =
    (as.numeric(predict(models[[1]], newdata = data, type = 'class')) - 1) * weight[1] +
    (as.numeric(predict(models[[2]], newdata = data)) - 1) * weight[2] +
    (as.numeric(predict(models[[3]], newdata = data)) - 1) * weight[3]
  ave_pred <- ifelse(ave_pred > 0.5, "1", "0")
}
```

```
fold_mse <- function(splits, models, weight) {
  map_dbl(
    splits,
    function(split) {
      test <- assessment(split)
      fit <- weight_pred(test, models, weight)
      t <- table(test$y, fit)
      sum(c(t[1,1], t[2,2]))/sum(t)
      #mean(((as.numeric(test$y) - 1) - fit)^2)
    }
  )
}
```

```
weights_mse <- function (splits, models, weights) {
  map_dbl(weights, ~mean(fold_mse(splits, models, .)))
}
```

```
computed_weights_mse <- weights_mse(cv$splits,
                                    list(rpart_mod, bag_mod, c50_mod),
                                    weights)
```

```
trained_weight <- weights[[which.min(computed_weights_mse)]]
```

```
trained_weight
```

```
## [1] 0.1 0.1 0.8
```

Model Comparison

rpart:

```
test <- testing(spam_split)
fit <- (as.numeric(predict(rpart_mod, newdata = test, type = 'class')) - 1)
t <- table(test$y, fit)
sum(c(t[1,1], t[2,2]))/sum(t)
```

```
## [1] 0.9052174
```

Bagged:

```
fit <- (as.numeric(predict(bag_mod, newdata = test)) - 1)
t <- table(test$y, fit)
sum(c(t[1,1], t[2,2]))/sum(t)
```

```
## [1] 0.9486957
```

C5.0:

```
fit <- (as.numeric(predict(c50_mod, newdata = test)) - 1)
t <- table(test$y, fit)
sum(c(t[1,1], t[2,2]))/sum(t)
```

```
## [1] 0.9643478
```

Weighted Average:

```
fit <- weight_pred(test, list(rpart_mod, bag_mod, c50_mod), trained_weight)
t <- table(test$y, fit)
sum(c(t[1,1], t[2,2]))/sum(t)
```

```
## [1] 0.9643478
```

So as measured by MSE on the test set, the C5.0 model is tied with the linear combination model. This result is exactly what we would expect from the given the weight `0.1, 0.1, 0.8` found in training. That weight means any classification by the C5.0 model basically overrides the selection from the other two models.