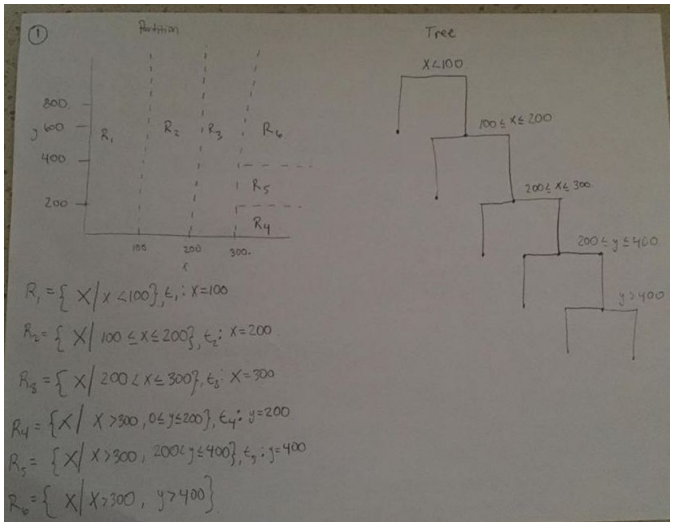
Introduction to Statistical Learning 7/26/2017

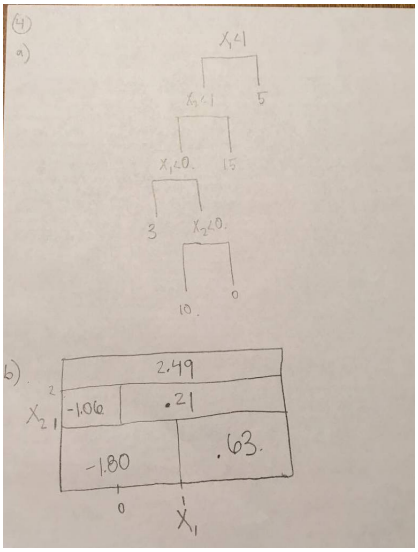
Chapter 8: Trees

Name: roninlaw
Email: roninlaw

1



4



5

1st Method: 4 votes $< .5$, and 6 votes $> .5$. So this method tells us the class is red.

2nd Method: $\frac{.10+.15+.20+.20+.55+.60+.60+.65+.70+.75}{10} = .45$. So this method tells us the class is not red.

6

Start by using recursive binary splitting to grow a large tree on the training data. Do this by picking a predictor along with a cut-point. This point is picked such that the RSS is minimized. Continue this process until we reach some stopping criterion. Now we apply cost-complexity pruning to the large tree to obtain a sequence of subtrees. For each value of α there is a subtree T that corresponds to (8.4). Use $k-$fold cross validation to choose α. Choose the α that minimizes the error.

7

```r
# ntree Analysis
library(randomForest)
library(MASS)
set.seed(1)
train = sample(1:nrow(Boston), nrow(Boston) / 2)
Boston.Test = Boston[-train, "medv"]
nTCE = rep("NA", 20)
for (i in 1:20) {
  rf.boston = randomForest(medv ~ ., data = Boston,
   subset = train, mtry = 6, ntree = i * 100)
  yhat = predict(rf.boston, newdata = Boston[-train,])
  nTCE[i] = mean((yhat - Boston.Test) ^ 2)
}

plot(1:20, nTCE, xlab = "ntrees", ylab = "MSE", type = "l")
which.min(nTCE) # 4 =:> the best number of trees to use is 400

#mtry Analysis
trees = 400
rf.boston.p = rf.boston = randomForest(medv ~ ., data = Boston, subset = train, mtry = 14,
    ntree = trees)
rf.boston.p2 = rf.boston = randomForest(medv ~ ., data = Boston, subset = train, mtry = 7,
    ntree = trees)
rf.boston.p3 = randomForest(medv ~ ., data = Boston, subset = train, ntree = trees)
plot(rf.boston.p$mse, col = "red", type = "l")
lines(1:trees, rf.boston.p2$mse, col = "green")
lines(1:trees, rf.boston.p3$mse, col = "blue")
```

The first plot suggests that it is best to build the model using half of the predictors as candidates at each split. The data also suggests that it is best to use 400 trees to grow.

8

(a)

```r
# Split Train/Test data
library(ISLR)
library(tree)
attach(Carseats)
set.seed(1)
High = ifelse(Sales <= 8, "No", "Yes")
Carseats = data.frame(Carseats, High)
train = sample(1:nrow(Carseats), nrow(Carseats) / 2)
Carseats.Train = Carseats[train,]
Carseats.Test = Carseats[-train,]
```

(b)

```r
# Make predictions
```

```
tree.carseats = tree(Sales ~ . , data = Carseats.Train)
yhat = predict(tree.carseats, newdata = Carseats.Test)
tmse = mean((yhat - Carseats.Test) ^ 2)
plot(tree.carseats)
text(pretty = 0)
sales.test = Carseats[-train,"Sales"]
tmse = mean((yhat - sales.test) ^ 2) # 4.148
```

(c)

```
# Optimal Level of Tree Complexity
set.seed(1)
cv.carseats.size = cv.tree(tree.carseats)$size # size = 11 corresponds to lowest cv value
cv.carseats.dev = cv.tree(tree.carseats)$dev
cv.carseats.k = cv.tree(tree.carseats)$k # k = 32.74 corresponds to the lowest cv value
plot(cv.carseats.size, cv.carseats.dev, type = "b")
plot(cv.carseats.size, cv.carseats.k, type = "b")

# Pruned Tree Analysis
prune.carseats = prune.tree(tree.carseats, best = 11)
yhatprune = predict(prune.carseats, newdata = Carseats.Test)
tmse = mean((yhatprune - sales.test) ^ 2) # 4.627
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```

Pruning the tree did not help the test MSE (it actually increased)

(d)

```
# Bagging
set.seed(1)
bag.carseats = randomForest(Sales ~ . - High, data = Carseats.Train, mtry = 11,
    importance = TRUE)
yhatbag = predict(bag.carseats, newdata = Carseats.Test)
plot(yhatbag, sales.test)
tmse = mean((yhatbag - sales.test) ^ 2) # 2.554
importance(bag.carseats)
varImpPlot(bag.carseats)
```

Price and ShelveLoc are the two most important variables in determining Sales.

(e)

```
# Random Forests
set.seed(1)
bag.carseats = randomForest(Sales ~ ., data = Carseats.Train, importance = TRUE) #
    default mtry = p/3 for regression
yhatrf = predict(bag.carseats, newdata = Carseats.Test)
plot(yhatrf, sales.test)
tmse = mean((yhatrf - sales.test) ^ 2) # 3.307
importance(bag.carseats)
varImpPlot(bag.carseats)
```

Most important variables are the same ones as the last part, and once we decreased m, the number of variables considered at each split, the MSE increased.

9

(a)

```r
# Split Train/Test Data
library(ISLR)
library(tree)
set.seed(1)
train = sample(1:nrow(OJ), 800)
OJ.Train = OJ[train,]
OJ.Test = OJ[-train,]
purchase.test = OJ[-train, "Purchase"]
```

(b)

```r
# Fit tree
tree.oj = tree(Purchase ~ ., data = OJ.Train)
summary(tree.oj) # .165 train error rate
```

(c)

```r
# Interpret Terminal Node
tree.oj
```

Picked leftmost terminal node on the tree corresponding to LoyalCH $< .508$, LoyalCH $< .264$, LoyalCH $< .034$. The class is MM. This means that points that call into this category will be assigned a Purchase value of MM.

(d)

```r
# Plot Tree
plot(tree.oj)
text(tree.oj, pretty = 0)
```

The tree has eight terminal nodes, and splits on LoyalCH multiple times. This tells us that splitting on LoyalCH reduces the MSE more so than the other predictors.

(e)

```r
# Predict Response
yhat = predict(tree.oj, newdata = OJ.Test, type = "class")
table(yhat, purchase.test) # .226 .... (147 + 62) / 270
```

(f)

```r
# Optimal Tree Size
cv.oj = cv.tree(tree.oj)
plot(cv.oj$size, cv.oj$k, type = "b")
```

(g)

```r
# Tree size vs. Cross Validated Classification Error
plot(cv.oj$size, cv.oj$dev, type = "b")
```

(h) Tree size with 5 terminal nodes is best.

(i)

```r
# Pruned Tree
prune.oj = prune.tree(tree.oj, best = 5) # .1825 train error rate
yhatprune = predict(prune.oj, newdata = OJ.Test, type = "class")
table(yhatprune, purchase.test) # .259....(119 + 81) / 270
```

(j) The unpruned tree has the lower train error rate.

(k) The pruned tree has the lower test error rate.

10

(a)

```
# Omit NAs
Hitters = na.omit(Hitters)
LogSalary = log(Hitters["Salary"])
Hitters = data.frame(Hitters, LogSalary)
```

(b)

```
# Split Train/Test Data
train = sample(1:nrow(Hitters), 200)
Hitters.Train = Hitters[train, ]
Hitters.Test = Hitters[-train, ]
```

(c)

```
# Plot: Shrinkage Value vs. Train MSE
Salary.Train = Hitters[train, "Salary.1"]
shrinks = c(.00000001, .0000001, .000001, .00001, .0001, .001, .01)
trainMSE = rep("NA", 7)
for (i in 1:7) {
  boost.hitters = gbm(Salary.1 ~ . - Salary, data = Hitters.Train, distribution =
      "gaussian",
                    n.trees = 1000, interaction.depth = 4, shrinkage = shrinks[i])
  yhatTrain = predict(boost.hitters, newdata = Hitters.Train, n.trees = 1000)
  trainMSE[i] = mean((yhatTrain - Salary.Train) ^ 2)
  print(trainMSE[i])
}
plot(shrinks, trainMSE, type = "b")
```

(d)

```
# Plot: Shrinkage vs. Test MSE
Salary.Test = Hitters[-train, "Salary.1"]
shrinks = c(.00000001, .0000001, .000001, .00001, .0001, .001, .01)
testMSE = rep("NA", 7)
for (i in 1:7) {
  boost.hitters = gbm(Salary.1 ~ . - Salary, data = Hitters.Train, distribution =
      "gaussian",
                    n.trees = 1000, interaction.depth = 4, shrinkage = shrinks[i])
  yhatTest = predict(boost.hitters, newdata = Hitters.Test, n.trees = 1000)
  testMSE[i] = mean((yhatTest - Salary.Test) ^ 2)
  print(testMSE[i])
}
plot(shrinks, testMSE, type = "b")
```

(e)

```
# Alternative Regression Analysis
quant_vars = c("AtBat", "Hits", "HmRun", "Runs", "RBI", "Walks", "Years",
               "CAtBat", "CHits", "CHmRun", "CRuns", "CRBI", "CWalks",
               "PutOuts", "Assists", "Salary", "Salary.1")

cor(Hitters.Train[quant_vars])
lm.fit = lm(Salary.1 ~ Years + CHits + Division, data = Hitters.Train)
lm.preds = predict(lm.fit, data = Hitters.Test)
lm.testMSE = mean((yhatTest - Salary.Test) ^ 2) # .189
```

```
library(pls)
set.seed(10)
pcr.fit = pcr(Salary.1 ~ . - Salary, data = Hitters.Train, scale = TRUE, validation =
    "CV")
pcr.preds = predict(pcr.fit, newdata = Hitters.Test)
pcr.testMSE = mean((pcr.preds - Salary.Test) ^ 2) # 0.3739779
```

(f) CRuns, CAtBat, CWalks are the most important variables in the boosted model.

(g)

```
# Bagging
library(randomForest)
bag.hitters = randomForest(Salary.1 ~ . - Salary, data = Hitters.Train, mtry = 19,
    importance = TRUE)
bag.preds = predict(bag.hitters, newdata = Hitters.Test)
bag.testMSE = mean((bag.preds - Salary.Test) ^ 2) # 0.1903033
```

11

(a)

```
# Split Train/Test Data
attach(Caravan)
contrasts(Purchase)
train = sample(1:nrow(Caravan), 1000)
Caravan.Train = Caravan[train,]
Caravan.Test = Caravan[-train, ]
train.purchase = Caravan[train, "Purchase"]
test.purchase = Carvan[-train, "Purchase"]
```

(b)

```
# Fit Boost Model
boost.caravan = gbm(Purchase ~ ., data = Caravan.Train, shrinkage = .01,
                    distribution = "gaussian", n.trees = 1000)
summary(boost.caravan)
```

MOSTYPE, PPERSAUT, and MKOOPKLA are the most important predictors

12

```
# Imports
library(randomForest)
library(gbm)

# Split data
train = sample(1 : nrow(Auto), nrow(Auto) / 2)
Auto.Train = Auto[train, ]
Auto.Test = Auto[-train, ]
mpg.test = Auto[-train, "mpg"]

# Analysis for Lin Regression
quant_cols = c("mpg", "cylinders", "displacement", "horsepower", "weight", "year")
cor(Auto[quant_cols])

# Bagging
```

```
bag.auto = randomForest(mpg ~ . - name, data = Auto.Train, mtry = 7,
                        importance = TRUE, ntrees = 1000)
yhat.bag = predict(bag.auto, Auto.Test)
tmse.bag = mean((mpg.test - yhat.bag) ^ 2) # 8.552

# Random Forests
rf.auto = randomForest(mpg ~ . - name, Auto.Train, mtry = 2,
                       importance = TRUE, ntrees = 1000)
yhat.rf = predict(rf.auto, Auto.Test)
tmse.rf = mean((mpg.test - yhat.rf) ^ 2)# 10.008

# Boosting
boost.auto = gbm(mpg ~ . - name, data = Auto.Train, distribution = "gaussian",
                 n.trees = 1000, interaction.depth = 4)
yhat.boost = predict(boost.auto, Auto.Test, n.trees = 1000)
tmse.boost = mean((mpg.test - yhat.boost) ^ 2) # 19.39


# Linear Regression
lin.fit = lm(mpg ~ cylinders, data = Auto.Train)
yhat.lin = predict(lin.fit, Auto.Test)
tmse.lin = mean((mpg.test - yhat.lin) ^ 2) # 27.55

# Multiple Linear Regression
mlin.fit = lm(mpg ~ cylinders + displacement + weight, data = Auto.Train)
yhat.mlin = predict(mlin.fit, Auto.Test)
tmse.mlin = mean((mpg.test - yhat.mlin) ^ 2) # 17.64018
```

The tree methods yielded the best results on the test data. Specifically the bagging approach resulted in the lowest test MSE.