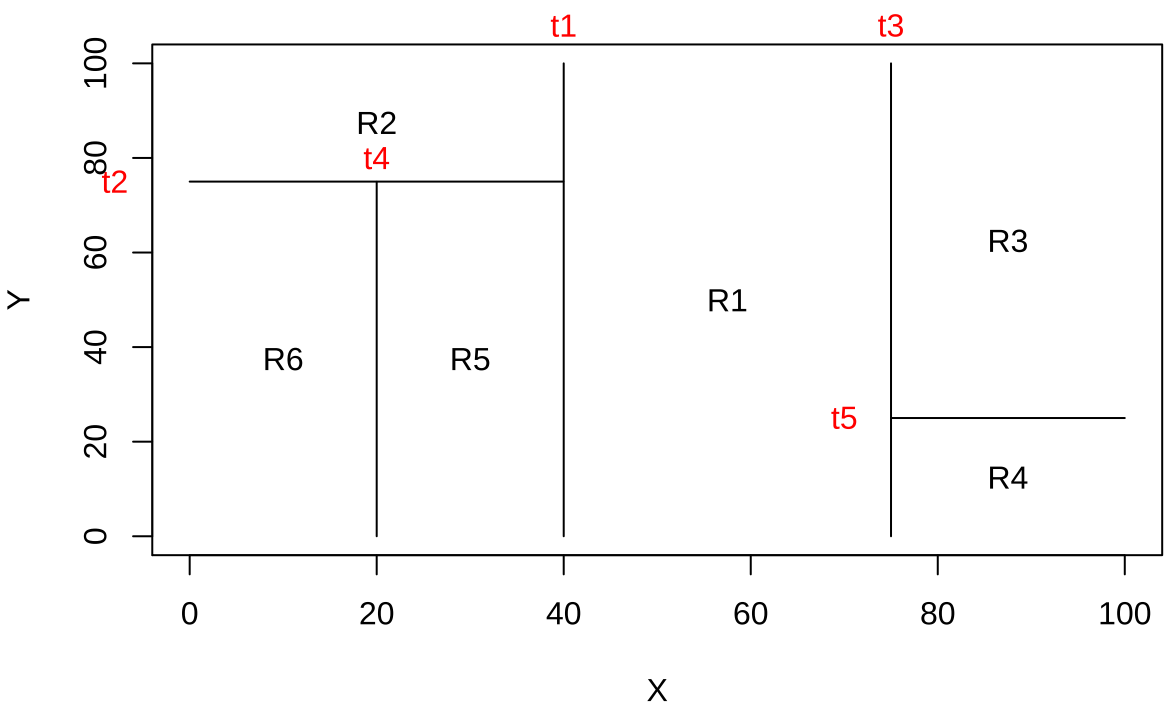
Math642_HW8_FyonaSun

Fyona Sun

3/23/2020

8.1

```r
par(xpd = NA)
plot(NA, NA, type = "n", xlim = c(0,100), ylim = c(0,100), xlab = "X", ylab = "Y")
lines(x = c(40,40), y = c(0,100))
text(x = 40, y = 108, labels = c("t1"), col = "red")
lines(x = c(0,40), y = c(75,75))
text(x = -8, y = 75, labels = c("t2"), col = "red")
lines(x = c(75,75), y = c(0,100))
text(x = 75, y = 108, labels = c("t3"), col = "red")
lines(x = c(20,20), y = c(0,75))
text(x = 20, y = 80, labels = c("t4"), col = "red")
lines(x = c(75,100), y = c(25,25))
text(x = 70, y = 25, labels = c("t5"), col = "red")

text(x = (40+75)/2, y = 50, labels = c("R1"))
text(x = 20, y = (100+75)/2, labels = c("R2"))
text(x = (75+100)/2, y = (100+25)/2, labels = c("R3"))
text(x = (75+100)/2, y = 25/2, labels = c("R4"))
text(x = 30, y = 75/2, labels = c("R5"))
text(x = 10, y = 75/2, labels = c("R6"))
```



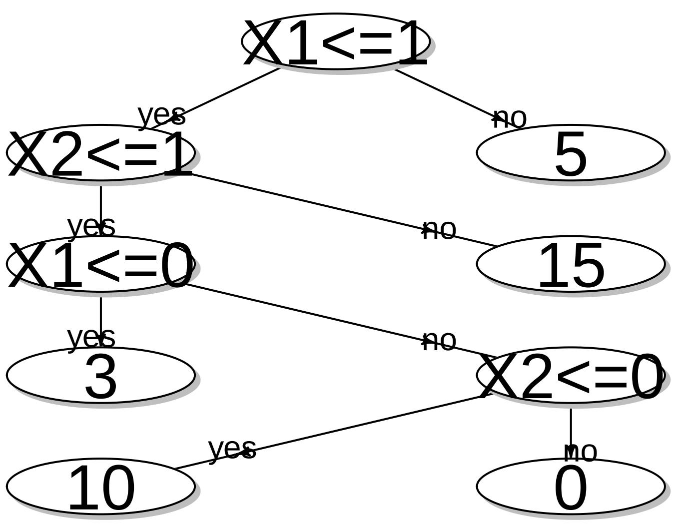
8.4

This question relates to the plots in Figure 8.12. (a) Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of Figure 8.12. The numbers inside the boxes indicate the mean of Y within each region.

```r
library(diagram)
```

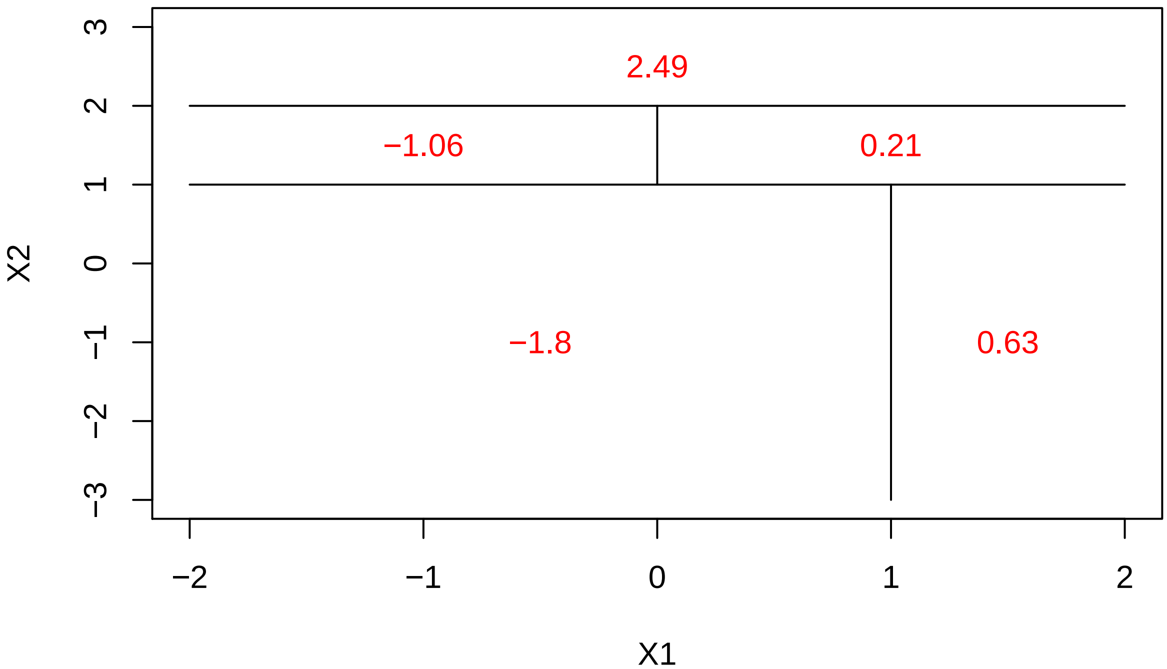
```
## Loading required package: shape
```

```r
par(mar = c(4, 4, 4, 4))
openplotmat()
elpos <- coordinates(c(1, 2, 2, 2,2))
fromto <- matrix(ncol = 2, byrow = TRUE, data = c(1, 2, 1, 3,
                                                   2, 4, 2, 5,
                                                   4, 6, 4, 7,
                                                   7, 8, 7, 9))
nr <- nrow(fromto)
arrpos <- matrix(ncol = 2, nrow = nr)
for (i in 1:nr) arrpos[i, ] <- straightarrow (to = elpos[fromto[i, 2], ],
                                from = elpos[fromto[i, 1], ],
                                lwd = 1, arr.pos = 0.7, arr.length = 0.15)



labels <- c("X1<=1","X2<=1","5","X1<=0","15","3","X2<=0","10","0")

for (i in 1:length(labels)) {
  textellipse(elpos[i,], 0.1, 0.05, lab = labels[i], box.col = "white", shadow.size = 0.005, cex = 2)
}

text(arrpos[1, 1]  - .01 , arrpos[1, 2], "yes")
text(arrpos[2, 1]  + .01, arrpos[2, 2], "no")
text(arrpos[3, 1]  - .01 , arrpos[3, 2], "yes")
text(arrpos[4, 1]  + .01, arrpos[4, 2], "no")
text(arrpos[5, 1]  - .01 , arrpos[5, 2], "yes")
text(arrpos[6, 1]  + .01, arrpos[6, 2], "no")
text(arrpos[7, 1]  - .01, arrpos[7, 2], "yes")
text(arrpos[8, 1]  + .01, arrpos[8, 2], "no")
```



(b) Create a diagram similar to the left-hand panel of Figure 8.12, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.

```r
par(xpd = NA)
plot(NA, NA, type = "n", xlim = c(-2, 2), ylim = c(-3, 3), xlab = "X1", ylab = "X2")
lines(x = c(-2, 2), y = c(1, 1))
lines(x = c(1, 1), y = c(-3, 1))
text(x = (-2 + 1)/2, y = -1, labels = c(-1.8),col = "red")
text(x = 1.5, y = -1, labels = c(0.63),col = "red")
lines(x = c(-2, 2), y = c(2, 2))
text(x = 0, y = 2.5, labels = c(2.49),col = "red")
lines(x = c(0, 0), y = c(1, 2))
text(x = -1, y = 1.5, labels = c(-1.06),col = "red")
text(x = 1, y = 1.5, labels = c(0.21),col = "red")
```



8.9

9. This problem involves the OJ data set which is part of the ISLR package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```r
library(ISLR)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.5.2

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.5.2
```

```r
attach(OJ)
set.seed(1)
Train <- createDataPartition(OJ$Purchase, p = 800/1070, list = FALSE)
training <- OJ[Train,]
testing <- OJ[-Train,]
```

(b) Fit a tree to the training data, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```r
library(rpart)
rpart_model <- rpart(Purchase ~ ., data = training, method = 'class',
                     control = rpart.control(cp = 0))
summary(rpart_model, cp = 1)
```

```
## Call:
## rpart(formula = Purchase ~ ., data = training, method = "class",
##     control = rpart.control(cp = 0))
##   n= 801
##
##            CP nsplit rel error    xerror       xstd
## 1 0.522435897      0 1.0000000 1.0000000 0.04423447
## 2 0.023504274      1 0.4775641 0.5128205 0.03626754
## 3 0.016025641      4 0.4070513 0.4583333 0.03473842
## 4 0.009615385      5 0.3910256 0.4583333 0.03473842
## 5 0.006410256      9 0.3493590 0.4358974 0.03405724
## 6 0.001068376     10 0.3429487 0.4679487 0.03502081
## 7 0.000000000     13 0.3397436 0.4903846 0.03565844
##
## Variable importance
##        LoyalCH      PriceDiff    SalePriceMM  ListPriceDiff        PriceMM
##             53              8              7              7              4
##        PriceCH         DiscMM      PctDiscMM          STORE        StoreID
##              3              3              3              2              2
## WeekofPurchase         DiscCH    SalePriceCH      PctDiscCH
##              2              2              2              1
##
## Node number 1: 801 observations
##   predicted class=CH  expected loss=0.3895131  P(node) =1
##     class counts:   489   312
##    probabilities: 0.610 0.390
```

```r
postResample(predict(rpart_model, training, type = 'class'), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8676654 0.7217437
```

The model summary shows that the variable LoyalCH is the most important for determining which orange juice a customer will buy. The tree model has an accuracy of 86.76654% and 1 node.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

```r
rpart_model
```

```
## n= 801
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 801 312 CH (0.61048689 0.38951311)
##     2) LoyalCH>=0.48285 500  80 CH (0.84000000 0.16000000)
##       4) LoyalCH>=0.7645725 255  10 CH (0.96078431 0.03921569) *
##       5) LoyalCH< 0.7645725 245  70 CH (0.71428571 0.28571429)
##        10) ListPriceDiff>=0.235 141  17 CH (0.87943262 0.12056738) *
```

```
##        11) ListPriceDiff< 0.235 104  51 MM (0.49038462 0.50961538)
##          22) PriceDiff>=0.085 42  11 CH (0.73809524 0.26190476) *
##          23) PriceDiff< 0.085 62  20 MM (0.32258065 0.67741935)
##            46) STORE>=3.5 11   3 CH (0.72727273 0.27272727) *
##            47) STORE< 3.5 51  12 MM (0.23529412 0.76470588) *
##     3) LoyalCH< 0.48285 301  69 MM (0.22923588 0.77076412)
##       6) LoyalCH>=0.282272 126  49 MM (0.38888889 0.61111111)
##        12) PriceDiff>=0.195 68  31 CH (0.54411765 0.45588235)
##          24) LoyalCH< 0.3084325 7   0 CH (1.00000000 0.00000000) *
##          25) LoyalCH>=0.3084325 61  30 MM (0.49180328 0.50819672)
##            50) WeekofPurchase>=248.5 40  17 CH (0.57500000 0.42500000)
##             100) STORE< 1.5 26   9 CH (0.65384615 0.34615385) *
##             101) STORE>=1.5 14   6 MM (0.42857143 0.57142857) *
##            51) WeekofPurchase< 248.5 21   7 MM (0.33333333 0.66666667) *
##        13) PriceDiff< 0.195 58  12 MM (0.20689655 0.79310345) *
##       7) LoyalCH< 0.282272 175  20 MM (0.11428571 0.88571429)
##        14) LoyalCH>=0.051325 110  19 MM (0.17272727 0.82727273)
##          28) LoyalCH< 0.203377 65  15 MM (0.23076923 0.76923077)
##            56) LoyalCH>=0.180654 7   3 CH (0.57142857 0.42857143) *
##            57) LoyalCH< 0.180654 58  11 MM (0.18965517 0.81034483) *
##          29) LoyalCH>=0.203377 45   4 MM (0.08888889 0.91111111) *
##        15) LoyalCH< 0.051325 65   1 MM (0.01538462 0.98461538) *
```

The root is split into nodes using the varable LoyalCH. If a customer scored LoyalCH is larger than 0.48 they are predicted to be in the class CH, which means we expect them to buy CH over MM. In CH class if the LoyalCH score is less than 0.76 then we will continue to examine whether the ListPriceDiff is greater than 0.24. If the ListPriceDiff is less than 0.24, the observation would be in the class MM. Then we check the number of stores. If the number of stores is greater than 4 then the observation is in class MM, otherwise it's in class CH.

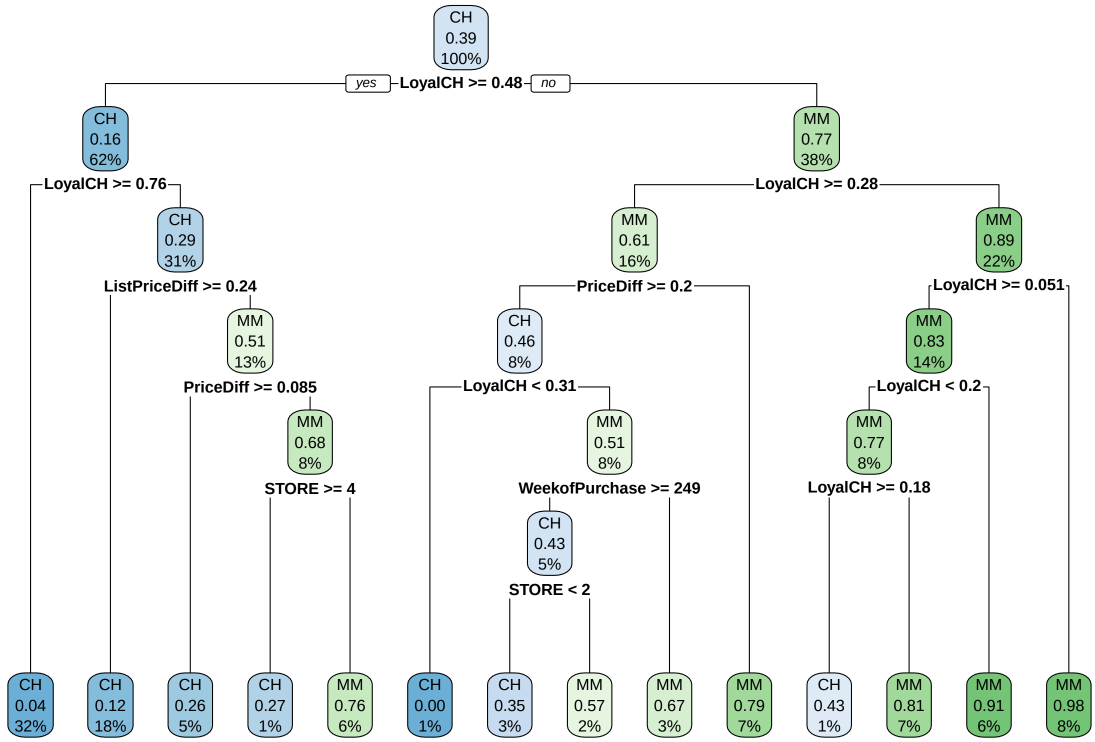
We can see from the output that the accuracy of that node is 81.9%

(d) Create a plot of the tree, and interpret the results.

```r
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.5.2
```

```r
rpart.plot(rpart_model)
```



(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
pred <- predict(rpart_model, testing, type = 'class')

caret::confusionMatrix(pred, testing$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 140  25
##         MM  24  80
##
##                Accuracy : 0.8178
##                  95% CI : (0.7664, 0.8621)
##     No Information Rate : 0.6097
##     P-Value [Acc > NIR] : 1.354e-13
##
##                   Kappa : 0.6166
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8537
##             Specificity : 0.7619
##          Pos Pred Value : 0.8485
##          Neg Pred Value : 0.7692
##              Prevalence : 0.6097
##          Detection Rate : 0.5204
##    Detection Prevalence : 0.6134
##       Balanced Accuracy : 0.8078
```

```
##
##        'Positive' Class : CH
##
```

The test accuracy is 81.78%

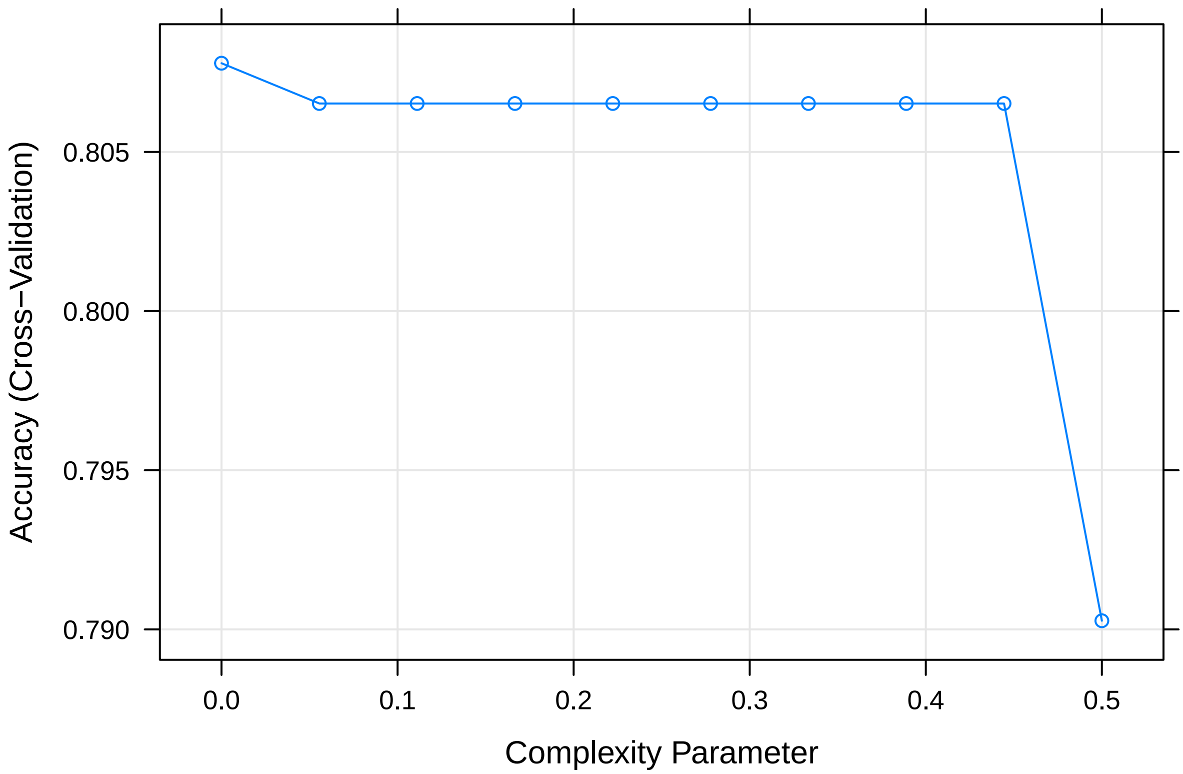
(f) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

```r
rpart_cv_model <- train(training[,-1], training[,1],
                        method = 'rpart',
                        trControl = trainControl(method = 'cv', number = 10),
                        tuneGrid = expand.grid(cp = seq(0, 0.5, length.out = 10)))

rpart_cv_model
```

```
## CART
##
## 801 samples
##  17 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 721, 721, 720, 722, 721, 721, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.00000000  0.8077877  0.5948491
##   0.05555556  0.8065227  0.5922651
##   0.11111111  0.8065227  0.5922651
##   0.16666667  0.8065227  0.5922651
##   0.22222222  0.8065227  0.5922651
##   0.27777778  0.8065227  0.5922651
##   0.33333333  0.8065227  0.5922651
##   0.38888889  0.8065227  0.5922651
##   0.44444444  0.8065227  0.5922651
##   0.50000000  0.7902727  0.5375197
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```r
plot(rpart_cv_model)
```



(h)

Which tree size corresponds to the lowest cross-validated classi- fication error rate?

```r
rpart_cv_model$bestTune
```

```
##   cp
## 1  0
```

```r
rpart_cv_model$results
```

```
##            cp  Accuracy     Kappa AccuracySD    KappaSD
## 1  0.00000000 0.8077877 0.5948491 0.03200645 0.07006905
## 2  0.05555556 0.8065227 0.5922651 0.04313488 0.09147587
## 3  0.11111111 0.8065227 0.5922651 0.04313488 0.09147587
## 4  0.16666667 0.8065227 0.5922651 0.04313488 0.09147587
## 5  0.22222222 0.8065227 0.5922651 0.04313488 0.09147587
## 6  0.27777778 0.8065227 0.5922651 0.04313488 0.09147587
## 7  0.33333333 0.8065227 0.5922651 0.04313488 0.09147587
## 8  0.38888889 0.8065227 0.5922651 0.04313488 0.09147587
## 9  0.44444444 0.8065227 0.5922651 0.04313488 0.09147587
## 10 0.50000000 0.7902727 0.5375197 0.07509702 0.20926057
```

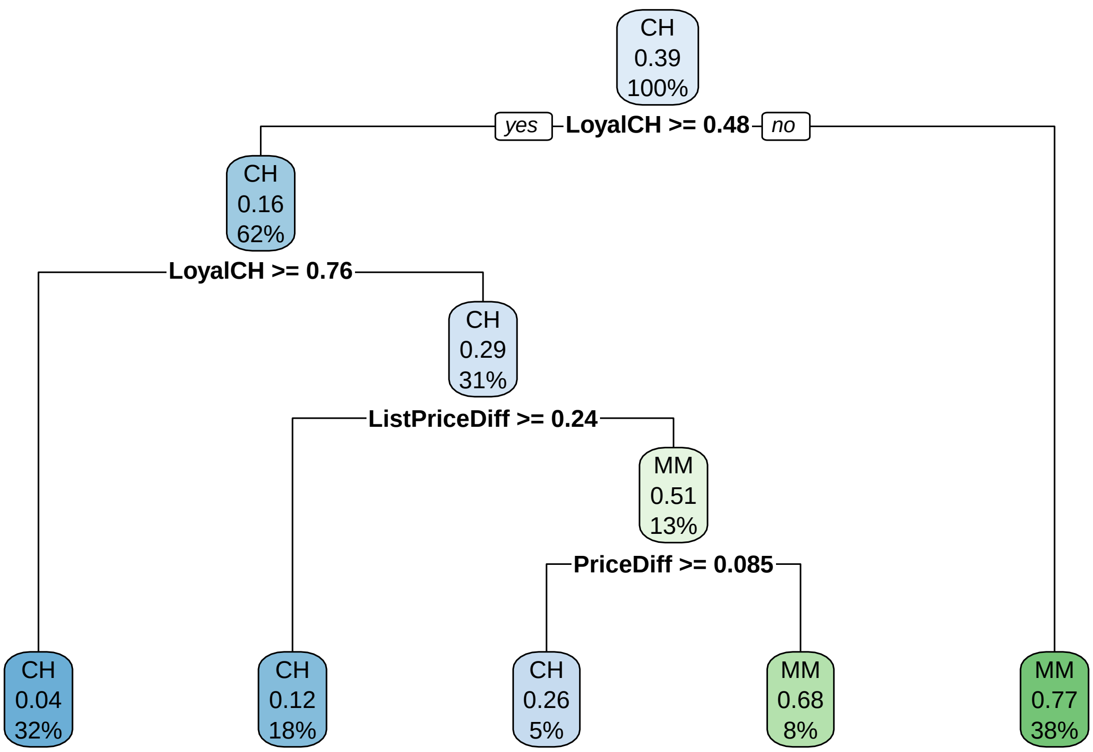
(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```r
set.seed(1)
rpart_tuned <- rpart(Purchase ~ ., data = training, method = 'class',
                     control = rpart.control(cp = 0.02))
rpart_tuned
```

```
## n= 801
##
```

```
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 801 312 CH (0.61048689 0.38951311)
##    2) LoyalCH>=0.48285 500  80 CH (0.84000000 0.16000000)
##      4) LoyalCH>=0.7645725 255  10 CH (0.96078431 0.03921569) *
##      5) LoyalCH< 0.7645725 245  70 CH (0.71428571 0.28571429)
##       10) ListPriceDiff>=0.235 141  17 CH (0.87943262 0.12056738) *
##       11) ListPriceDiff< 0.235 104  51 MM (0.49038462 0.50961538)
##         22) PriceDiff>=0.085 42  11 CH (0.73809524 0.26190476) *
##         23) PriceDiff< 0.085 62  20 MM (0.32258065 0.67741935) *
##    3) LoyalCH< 0.48285 301  69 MM (0.22923588 0.77076412) *
```

```r
rpart.plot(rpart_tuned)
```



(j) Compare the training error rates between the pruned and un- pruned trees. Which is higher?

```r
postResample(predict(rpart_model,
                     training,
                     type = 'class'), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8676654 0.7217437
```

```r
postResample(predict(rpart_tuned,
                     training,
                     type = 'class'), training$Purchase)
```

```
##  Accuracy     Kappa
## 0.8414482 0.6761968
```

The unpruned model has higher training accuracy. This does not mean we should never prune. It just means that the training set is well representative of the testing set.

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

```
postResample(predict(rpart_model,
                     testing,
                     type = 'class'), testing$Purchase)
```

```
##  Accuracy     Kappa
## 0.8178439 0.6166196
```

```
postResample(predict(rpart_tuned,
                     testing,
                     type = 'class'), testing$Purchase)
```

```
##  Accuracy     Kappa
## 0.8104089 0.6090228
```

The unpruned model has higher accuracy.

8.10

10. We now use boosting to predict Salary in the Hitters data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

```
library(data.table)
Hitters<- read.csv('~/Math642_FyonaSun/Hitters.csv')
setDT(Hitters)
Hitters <-  Hitters[-which(is.na(Hitters$Salary)), ]
sum(is.na(Hitters$Salary))
```

```
## [1] 0
```

```
Hitters$Salary <-  log(Hitters$Salary)
```

(b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

```
train <-  1:200
train.hitters <-  Hitters[train, ]
test.hitters <-  Hitters[-train, ]
```

(c) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter lambda. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

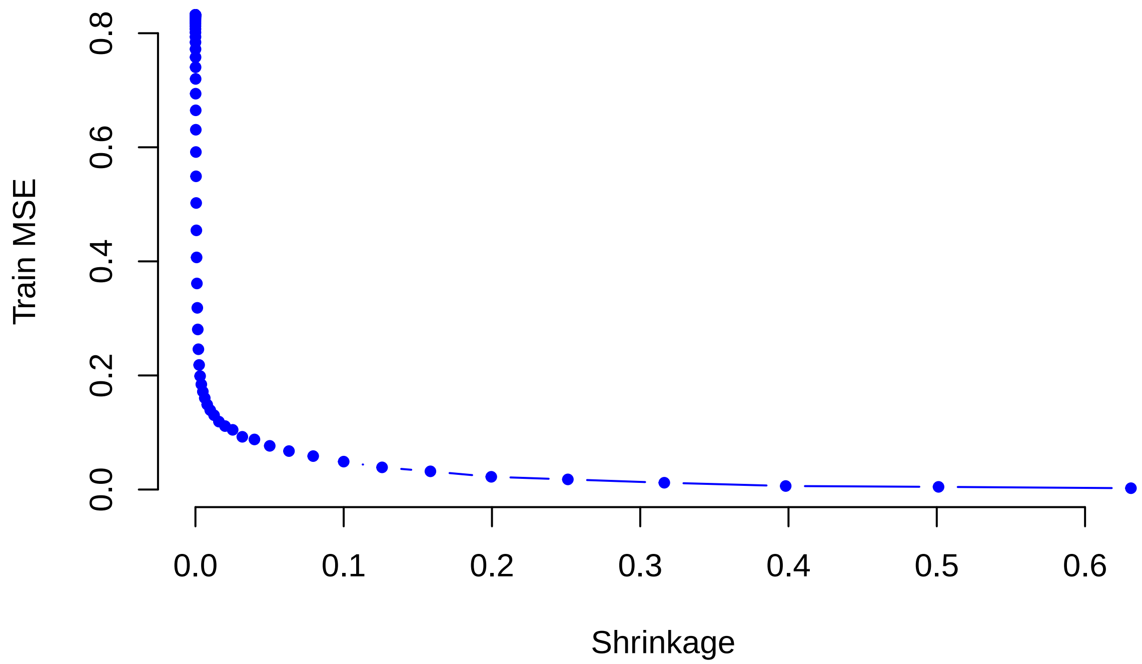
```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.5.2
```

```
## Loaded gbm 2.1.5
```

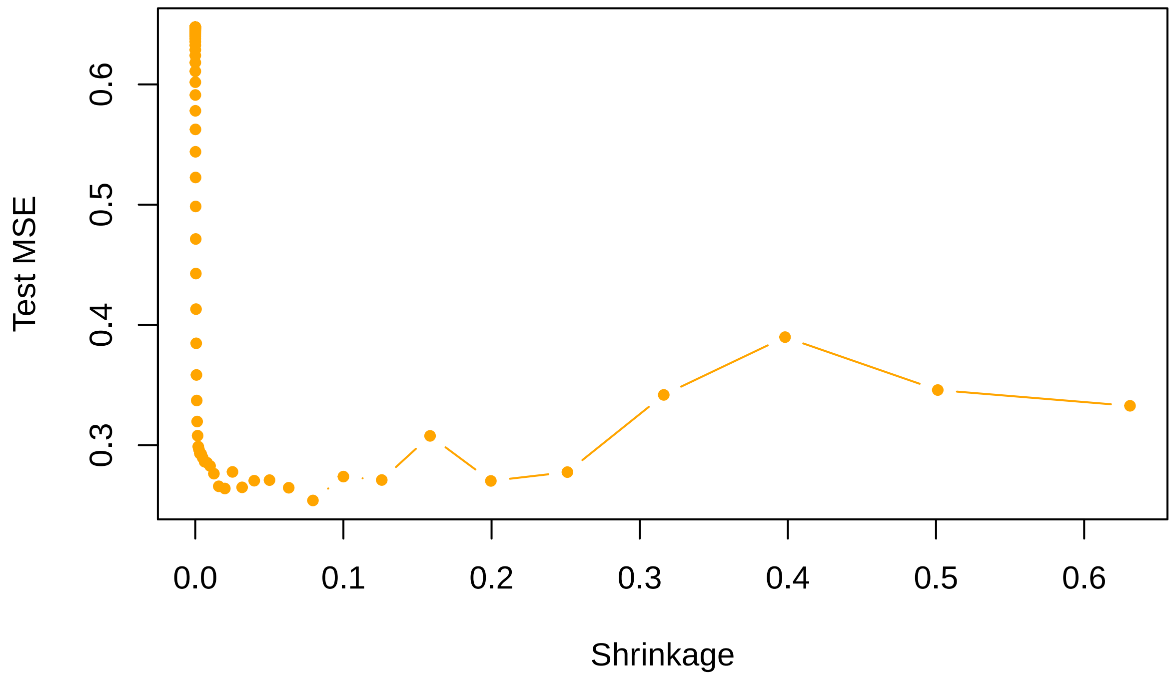
```
set.seed(1)

pows <-  seq(-10, -0.2, by=0.1)
lambdas <-  10 ^ pows
train.errors <-  rep(NA, length(lambdas))
test.errors <-  rep(NA, length(lambdas))

for (i in 1:length(lambdas)) {
```

```r
  boost.hitters <-  gbm(Salary ~ AtBat+Hits+HmRun+Runs+RBI+Walks+Years+CAtBat+CHits+CHmRun +CRuns+CRBI+C
                        distribution="gaussian",
                        n.trees=1000,
                        shrinkage=lambdas[i])
  train.pred <-  predict(boost.hitters, train.hitters, n.trees=1000)
  test.pred <-  predict(boost.hitters, test.hitters, n.trees=1000)
  train.errors[i] <-  mean((train.hitters$Salary - train.pred)^2)
  test.errors[i] <-  mean((test.hitters$Salary - test.pred)^2)
}

plot(lambdas, train.errors, type="b",
     xlab="Shrinkage", ylab="Train MSE",
     col="blue", pch=20, bty = "n")
```



(d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```r
plot(lambdas, test.errors, type="b",
     xlab="Shrinkage", ylab="Test MSE",
     col="orange", pch=20)
```




```r
min(test.errors)
```

```
## [1] 0.2540265
```

```r
lambdas[which.min(test.errors)]
```

```
## [1] 0.07943282
```

The smallest test MSE of boosting is 0.2540265.

(e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.5.2
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-18
```

```r
fitlm = lm(Salary ~ AtBat+Hits+HmRun+Runs+RBI+Walks+Years+CAtBat+CHits+CHmRun +CRuns+CRBI+CWalks+League+
pred = predict(fitlm, test.hitters)
mean((pred - test.hitters$Salary)^2)
```

```
## [1] 0.4917959
```

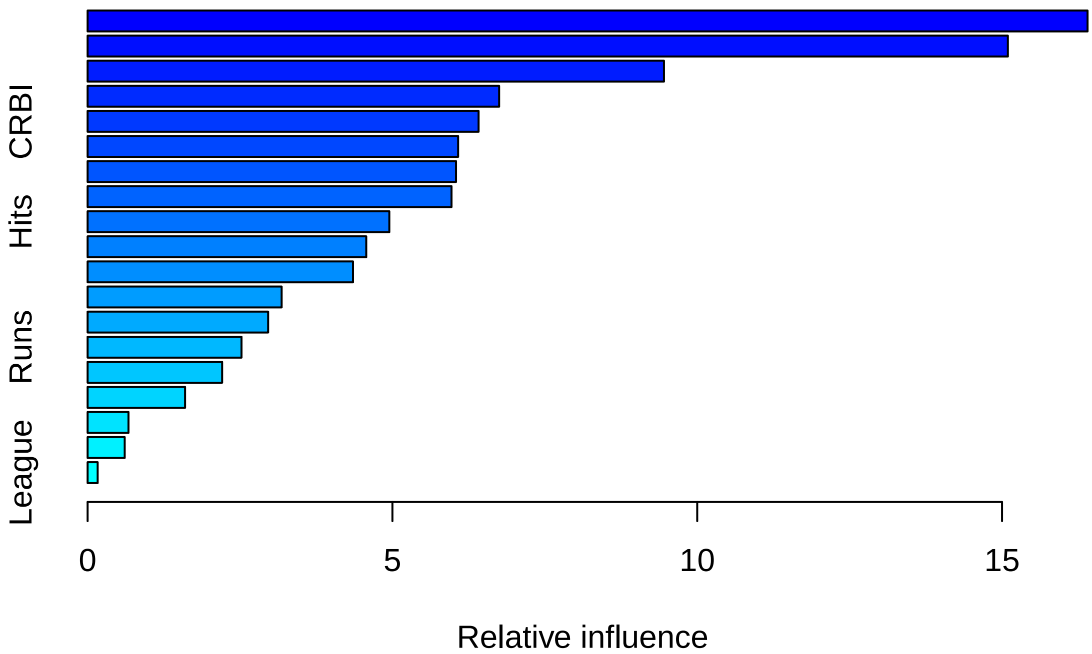
```r
set.seed(1)

x <-  model.matrix(Salary ~ . , data=train.hitters)
y <-  train.hitters$Salary
x.test <-  model.matrix(Salary ~ . , data=test.hitters)
lasso.fit <-  glmnet(x, y, alpha=1)
lasso.pred <-  predict(lasso.fit, s=0.01, newx=x.test)
mean((test.hitters$Salary - lasso.pred)^2)
```

```
## [1] 0.5515612
```

The linear model and the Lasso have higher test MSE than boosting.

(f) Which variables appear to be the most important predictors in the boosted model?

```r
boost.best <-  gbm(Salary ~ AtBat+Hits+HmRun+Runs+RBI+Walks+Years+CAtBat+CHits+CHmRun +CRuns+CRBI+CWalks
                   distribution="gaussian", n.trees=1000,
                   shrinkage=lambdas[which.min(test.errors)])
summary(boost.best)
```




```
##                 var    rel.inf
## CAtBat       CAtBat 16.4028631
## CRuns         CRuns 15.0957847
## PutOuts     PutOuts  9.4543457
## CWalks       CWalks  6.7508573
## CRBI           CRBI  6.4133341
## CHmRun       CHmRun  6.0778920
## Walks         Walks  6.0434466
## Years         Years  5.9694890
## Hits           Hits  4.9492273
## Assists     Assists  4.5706789
## RBI             RBI  4.3533790
## AtBat         AtBat  3.1830285
## HmRun         HmRun  2.9616488
## Runs           Runs  2.5244563
## Errors       Errors  2.2067482
## CHits         CHits  1.5986893
## Division   Division  0.6707407
## NewLeague NewLeague  0.6083337
## League       League  0.1650564
```

CAtBat, CRBI and CWalks are the most important variables.

(g) Now apply bagging to the training set. What is the test set MSE for this approach?

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
set.seed(1)
fit.rf <-  randomForest(Salary ~ AtBat+Hits+HmRun+Runs+RBI+Walks+Years+CAtBat+CHits+CHmRun +CRuns+CRBI+C
                            ntree=500, mtry=19)
pred.rf <-  predict(fit.rf, test.hitters)
mean((test.hitters$Salary - pred.rf)^2)
```

```
## [1] 0.2299324
```

Test MSE for random forest bagging is about 0.2299324, which is slightly better than the smallest test MSE for boosting.