HW7

Norman Hong

April 17, 2019

Book 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.

If $X_1 \geq 1$ then 5, else if $X_2 \geq 1$ then 15, else if $X_1 < 0$ then 3, else if $X_2 < 0$ then 10, else 0. Look to the last page for tree diagram.

(b)

Create a diagram similar to the left-hand panel of Figure 8.12, using the t ree 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.

Extra 61

Consider the concrete strength data from Extra 37. There are eight numerical predictors and one numerical response.

(a)

Fit a random forest to the data to predict strength. Estimate the rms prediction error using 10-fold cross validation. you will have to write your own cross validation code to do this.

```r
# Have to pass concrete data set as argument.
cv.randomForest <- function(concrete){
     # Randomly shuffle data
    concrete <- concrete[sample(nrow(concrete)),]

    # Create 10 equally size folds
    folds <- cut(seq(1, nrow(concrete)), breaks=10, labels=FALSE)

    rmse <- c()
    # Perform 10 fold cross validation
    for (i in 1:10){
        # segment data by fold integer
        testIndex <- which(folds==i, arr.ind=TRUE)
        testData <- concrete[testIndex,]
        trainData <- concrete[-testIndex,]
        rf.concrete <- randomForest(y~.,mtry=4, data=trainData)
        pred <- predict(rf.concrete, newdata=testData)
        rmse[i] <- sqrt(mean((pred-testData$y)^2))
    }
    cat(mean(rmse))
}
```

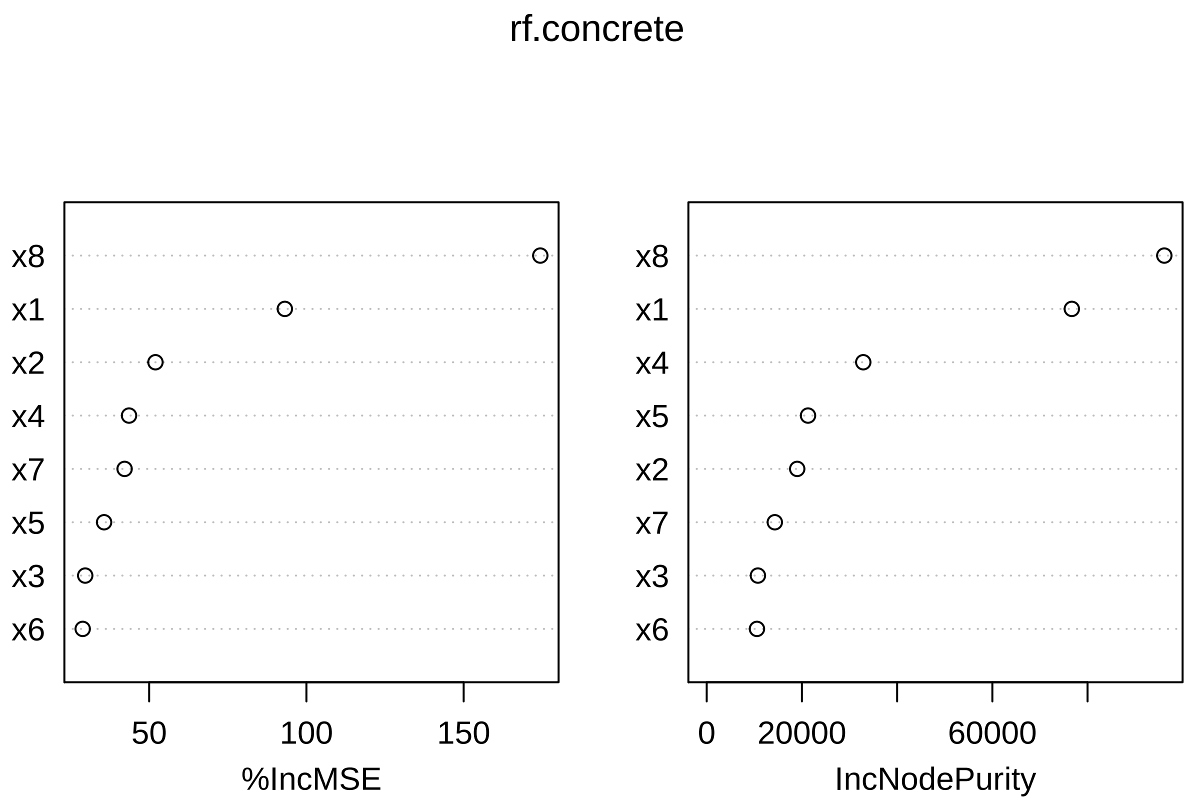
```
set.seed(100)
cv.randomForest(concrete)
```

```
## 4.787499
```

(b)

Make a variable importance plot for a r andom forest model for the full data

```
# %IncMSE is based on the mean decrease of accuracy in predictions on the out of bag samples
# when the variable is excluded from the model.
# IncNodePurity is the total decrease in node impurity (not purity) that results from
# splits over that variable averaged over all trees.
rf.concrete <- randomForest(y~., mtry=4, data=concrete, importance=TRUE)
varImpPlot(rf.concrete)
```



(c)

Leave out the least important predictor and repeat part a. How does the estimated rms prediction error change? Comment on your observation.

The estimated rmse predition error increase slightly from 4.787 to 4.805. This slight increase in rmse is not terrible since the model is simplier.

```
# Have to pass concrete data set as argument.
cv.randomForest <- function(concrete){
     # Randomly shuffle data
    concrete <- concrete[sample(nrow(concrete)),]

    # Create 10 equally size folds
    folds <- cut(seq(1, nrow(concrete)), breaks=10, labels=FALSE)

    rmse <- c()
    # Perform 10 fold cross validation
    for (i in 1:10){
        # segment data by fold integer
        testIndex <- which(folds==i, arr.ind=TRUE)
        testData <- concrete[testIndex,]
        trainData <- concrete[-testIndex,]
        rf.concrete <- randomForest(y~x1+x2+x3+x4+x5+x7+x8,mtry=4, data=trainData)
        pred <- predict(rf.concrete, newdata=testData)
        rmse[i] <- sqrt(mean((pred-testData$y)^2))
    }
    cat(mean(rmse))
}
```

```
set.seed(100)
cv.randomForest(concrete)
```

```
## 4.805678
```

Extra 62

Consider the concrete strength data from problem 37. There are 8 numerical predictors and 1 numerical response. Load the data and split them into a training and test set (70%/30%).

Fit a gbm model to the training data to predict strength, for several choices of n.trees, shrinkage, and interaction.depth. Compute the rms prediction errors for the training and the test sets in each case and demonstrate that it is possible to overfit with this method.

```
boost.trees <- function(concrete, trees){
    trainIndex <- sample(nrow(concrete), .7*nrow(concrete), replace=FALSE)
    testIndex <- -trainIndex
    set.seed(100)
    boost.concrete <- gbm(y~., data=concrete[trainIndex,], distribution='gaussian', n.trees=trees, inte

    #test RMSE
    pred <- predict(boost.concrete, newdata=concrete[testIndex,], n.trees=trees)
    cat('test RMSE for', trees, 'trees is', sqrt(mean((pred-concrete[testIndex,]$y)^2)))

    #train RMSE
    pred <- predict(boost.concrete, newdata=concrete[trainIndex,], n.trees=trees)
    cat('\ntrain RMSE for', trees, 'trees is', sqrt(mean((pred-concrete[trainIndex,]$y)^2)))
}
```

```
boost.trees(concrete, 50)
```

```
## test RMSE for 50 trees is 13.53733
## train RMSE for 50 trees is 13.04474
```

```r
boost.trees(concrete, 2000)
```

```
## test RMSE for 2000 trees is 5.428962
## train RMSE for 2000 trees is 3.139743
```

```r
boost.trees(concrete, 10000)
```

```
## test RMSE for 10000 trees is 4.39427
## train RMSE for 10000 trees is 1.78428
```

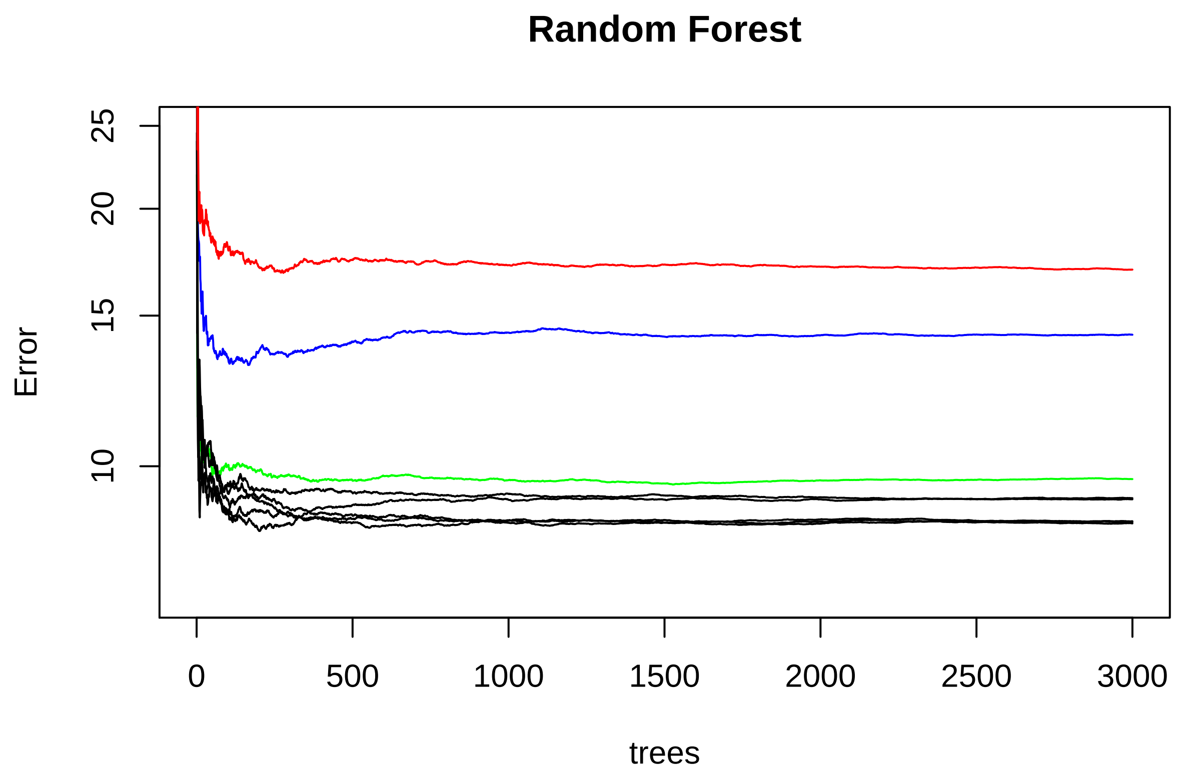
Book 7

In the lab, we applied random forests to the Boston data set using mtry=6 and using ntree=25 and ntree=500. Create a plot displaying the test error resulting from random forests on this data set for a more comprehensive range of values for mtry and ntree. You can model your plot after Figure 8.10. Describe the results obtained.

The optimal value of mtry is not at the extremes, 13 or 1. It turns out that it is somewhere in between because the test MSE is lower than the test MSE for when mtry is either 13 or 1.

```r
set.seed(100)
train <- sample(1:nrow(Boston), nrow(Boston)/2, replace=FALSE)
forest.Boston1 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=13, ntree=3000)
forest.Boston2 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=10, ntree=3000)
forest.Boston3 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=6, ntree=3000)
forest.Boston4 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=1, ntree=3000)
forest.Boston5 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=5, ntree=3000)
forest.Boston6 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=4, ntree=3000)
forest.Boston7 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=3, ntree=3000)
forest.Boston8 <- randomForest(medv~., data=Boston[train,],xtest=Boston[-train, c(1:13)],
                              ytest=Boston[-train,'medv'], mtry=8, ntree=3000)

plot(forest.Boston1, main='Random Forest', log='y', col='blue', ylim=c(7,25))
lines(x=c(1:forest.Boston2$ntree), y=forest.Boston2$test$mse, type='l', col='green')
lines(x=c(1:forest.Boston3$ntree), y=forest.Boston3$test$mse, type='l')
lines(x=c(1:forest.Boston5$ntree), y=forest.Boston5$test$mse, type='l')
lines(x=c(1:forest.Boston6$ntree), y=forest.Boston6$test$mse, type='l')
lines(x=c(1:forest.Boston7$ntree), y=forest.Boston7$test$mse, type='l')
lines(x=c(1:forest.Boston8$ntree), y=forest.Boston8$test$mse, type='l')
lines(x=c(1:forest.Boston4$ntree), y=forest.Boston4$test$mse, type='l', col='red')
```



Book 9

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

(a)

Creating a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
set.seed(100)
trainIdx <- sample(nrow(OJ), .7*nrow(OJ))
trainData <- OJ[trainIdx,]
testData <- OJ[-trainIdx,]
```

(b)

Fit a tree to the training data, with Purchase as the response and the other variables except for buy 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?

The number of terminal nodes is 9. The training error rate (misclassification rate) is 16.4%.

```
set.seed(100)
tree.oj <- tree(Purchase~., data=trainData)
summary(tree.oj)
```

```
##
```

```
## Classification tree:
## tree(formula = Purchase ~ ., data = trainData)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff" "SpecialCH" "StoreID"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7236 = 535.4 / 740
## Misclassification error rate: 0.1642 = 123 / 749
```

(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.

node 4 is a terminal node that had the split condition LoyalCH less than .0356. The number of observations in this terminal node is 55. The deviance of this node is 0. Every observation in this node is assigned to be MM. Lastly, the distribution of this node is 1 MM and 0 CH.

```r
tree.oj
```

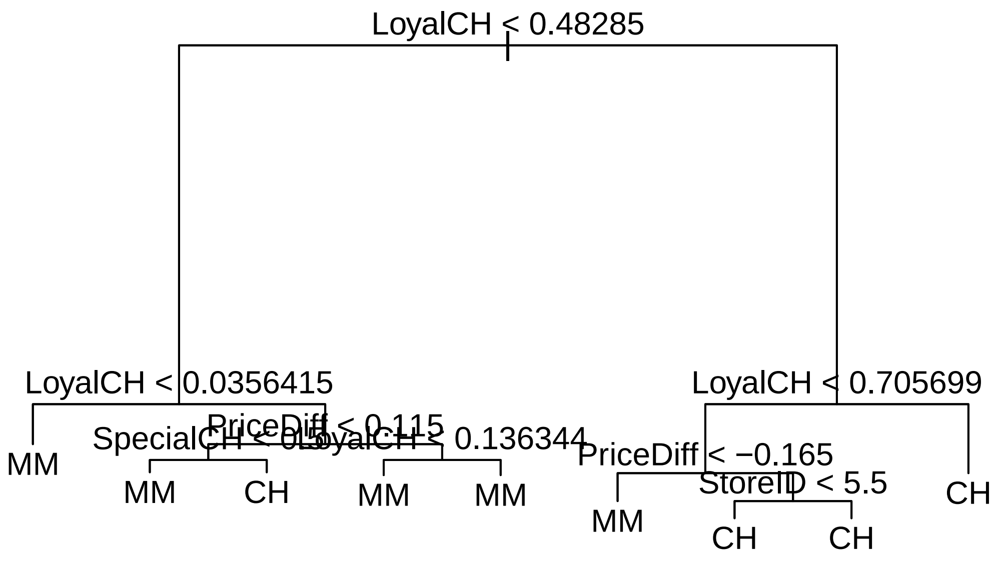
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 749 999.90 CH ( 0.61282 0.38718 )
##    2) LoyalCH < 0.48285 292 319.30 MM ( 0.23630 0.76370 )
##      4) LoyalCH < 0.0356415 55   0.00 MM ( 0.00000 1.00000 ) *
##      5) LoyalCH > 0.0356415 237 285.90 MM ( 0.29114 0.70886 )
##       10) PriceDiff < 0.115 108 100.50 MM ( 0.17593 0.82407 )
##         20) SpecialCH < 0.5 98  76.71 MM ( 0.13265 0.86735 ) *
##         21) SpecialCH > 0.5 10  13.46 CH ( 0.60000 0.40000 ) *
##       11) PriceDiff > 0.115 129 172.30 MM ( 0.38760 0.61240 )
##         22) LoyalCH < 0.136344 23  13.59 MM ( 0.08696 0.91304 ) *
##         23) LoyalCH > 0.136344 106 146.00 MM ( 0.45283 0.54717 ) *
##    3) LoyalCH > 0.48285 457 380.90 CH ( 0.85339 0.14661 )
##      6) LoyalCH < 0.705699 192 231.80 CH ( 0.70833 0.29167 )
##       12) PriceDiff < -0.165 27  32.82 MM ( 0.29630 0.70370 ) *
##       13) PriceDiff > -0.165 165 175.60 CH ( 0.77576 0.22424 )
##         26) StoreID < 5.5 107 132.20 CH ( 0.69159 0.30841 ) *
##         27) StoreID > 5.5 58  29.11 CH ( 0.93103 0.06897 ) *
##      7) LoyalCH > 0.705699 265  91.54 CH ( 0.95849 0.04151 ) *
```

(d)

Create a plot of the tree, and interpret the results.

There are 8 total splits in the decision tree. Only 4 different predictor variables were used as split conditions.

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



(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?

The test misclassification error is 19%.

```
pred <- predict(tree.oj, testData, type='class')
cm <- table(pred, testData$Purchase)
cm
```

```
##
## pred  CH  MM
##   CH 169  36
##   MM  25  91
```

```
(cm[3]+cm[2])/(sum(cm))
```

```
## [1] 0.1900312
```

(f)

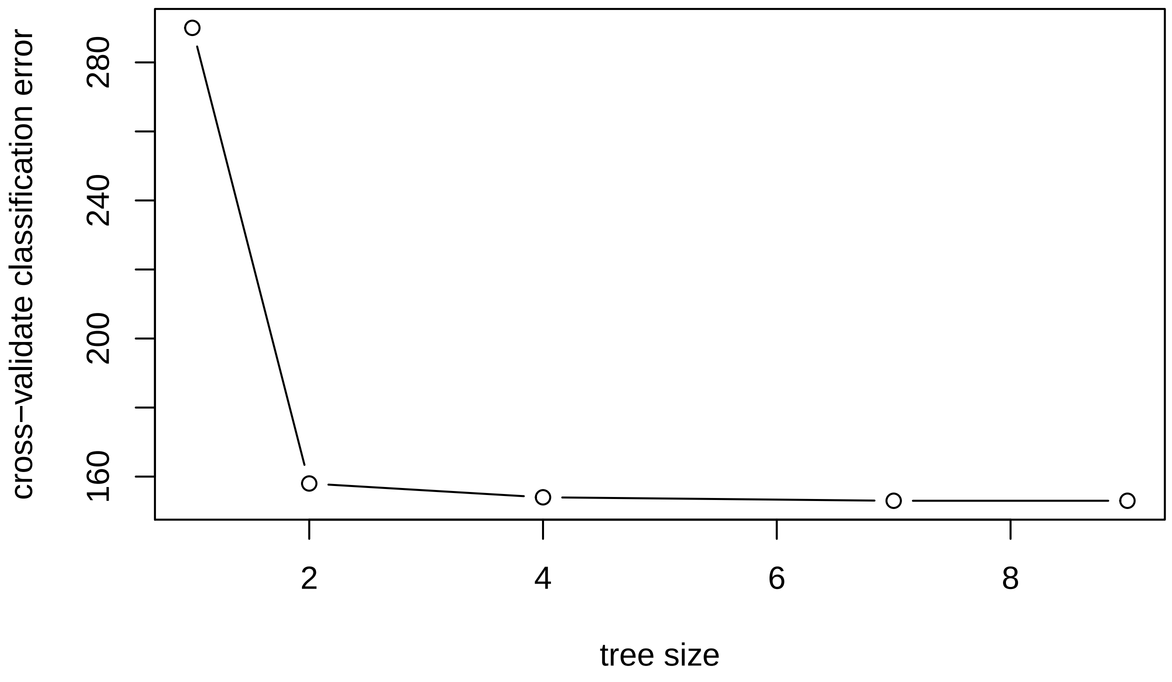
Apply the cv.tree() function to the training set in order to determine the optimal tree size.

```
cv.oj <- cv.tree(tree.oj, FUN=prune.misclass) # prune.misclass means to use classification error to det
```

(g)

Produce a plot with tree size on the x-axis and cross-validate classification error rate on the y-axis.

```r
plot(x=cv.oj$size, y=cv.oj$dev, type='b', xlab='tree size', ylab='cross-validate classification error')
```



(h)

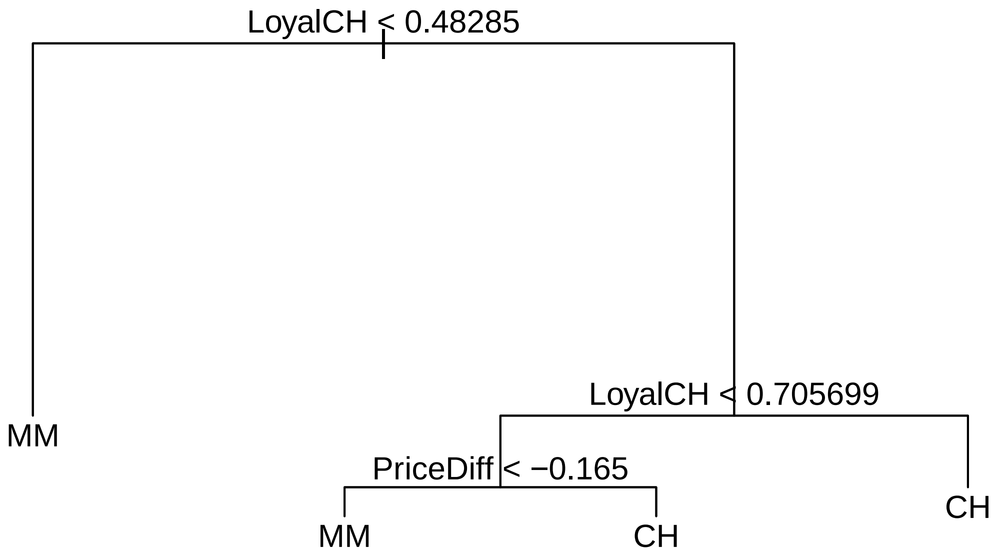
Which tree size corresponds to the lowest cross-validate classification error rate?

The tree size with the lowest cross-validate classification error is 7 and 9. However, there is not much difference between cross-validate classification error of tree size 4 and tree size 7. Therefore, it might be best to use tree size 4.

(i)

Produce a pruned tree corresponding to the optimmal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with 5 terminal nodes.

```r
prune.oj <- prune.misclass(tree.oj, best=4)
plot(prune.oj)
text(prune.oj, pretty=0)
```



(j)

Compare the training error rates between the pruned and unpruned tree. Which is higher?

The training misclassification error rate is 16.4% for the unpruned tree, and 16.69% for pruned tree. The pruned tree has a higher error rate.

```
summary(tree.oj)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = trainData)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff" "SpecialCH" "StoreID"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7236 = 535.4 / 740
## Misclassification error rate: 0.1642 = 123 / 749
```

```
summary(prune.oj)
```

```
##
## Classification tree:
## snip.tree(tree = tree.oj, nodes = c(13L, 2L))
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  4
## Residual mean deviance:  0.8313 = 619.3 / 745
## Misclassification error rate: 0.1669 = 125 / 749
```

(k)

Compare the test error rates between the pruned and unpruned trees. WHich is higher?

The test error rate for unpruned tree was found to be 19% earlier. The test error rate for pruned tree is 18.69%. The test error rate for unpruned tree is slightly higher.

```r
pred <- predict(prune.oj, testData, type='class')
cm <- table(pred, testData$Purchase)
cm
```

```
##
## pred  CH  MM
##   CH 167  33
##   MM  27  94
```

```r
(cm[3]+cm[2])/(sum(cm))
```

```
## [1] 0.1869159
```

Extra 59

This problem uses the MNIST image classificatio ndata, available as mnist_all.RData that were used earlier. We want to distinguish between 4 and 5.

```r
train.x <- train$x[(train$y == 4) | (train$y == 5),]
train.y <- train$y[train$y == 4 | train$y == 5]
train.y <- as.factor(as.numeric(train.y == 4)) # Class 1 = 4; class 0 = 5

test.x <- test$x[(test$y == 4) | (test$y == 5),]
test.y <- test$y[test$y == 4 | test$y == 5]
test.y <- as.factor(as.numeric(test.y == 4))
#vars <- apply(train.x, MARGIN=2, var)
#var.0 <- vars != 0
#train.x <- train.x[,var.0]
#vars <- apply(test.x, MARGIN=2, var)
#var.0 <- vars != 0
#test.x <- test.x[,var.0]

train.df <- data.frame(train.x, y=train.y)
test.df <- data.frame(test.x, y=test.y)
```

(a)

Fit a tree model to the training data and assess its accuracy (prediction error) using the test data. Be sure to make a classification tree and to predict a class, not a numerical value.

The test prediction error rate is 3.3%.

```r
set.seed(100)
tree.mnist <- tree(y~., data=train.df)
pred <- predict(tree.mnist, newdata=test.df, type='class')

cm <- table(pred, test.df$y)

(cm[2] + cm[3])/sum(cm)
```

```
## [1] 0.03361793
```

(b)

Fit a random forest model to the training data. Choose the number of trees such that the algorithm runs no longer than 5 minutes. Then assess the accuracy (prediction error) using the test data.

The test misclassification error rate is .16%.

```r
set.seed(100)
forest.mnist <- randomForest(y~., data=train.df, mtry=5, ntree=200)
```

```r
pred <- predict(forest.mnist, newdata=test.df, type='class')
cm <- table(pred, test.df$y)
cm
```

```
##
## pred   0   1
##    0 890   1
##    1   2 981
```

```r
(cm[2] + cm[3])/sum(cm)
```

```
## [1] 0.001600854
```

(c)

Fit a bagging model to the training data, using the same number of trees as in part b. Assess the accuracy (prediction error) using the test data.

The prediction error is .53%.

```r
set.seed(100)
bagging.mnist <- randomForest(y~., data=train.df, mtry=784, ntree=200)
```

```r
pred <- predict(bagging.mnist, newdata=test.df, type='class')
cm <- table(pred, test.df$y)
(cm[2] + cm[3])/sum(cm)
```

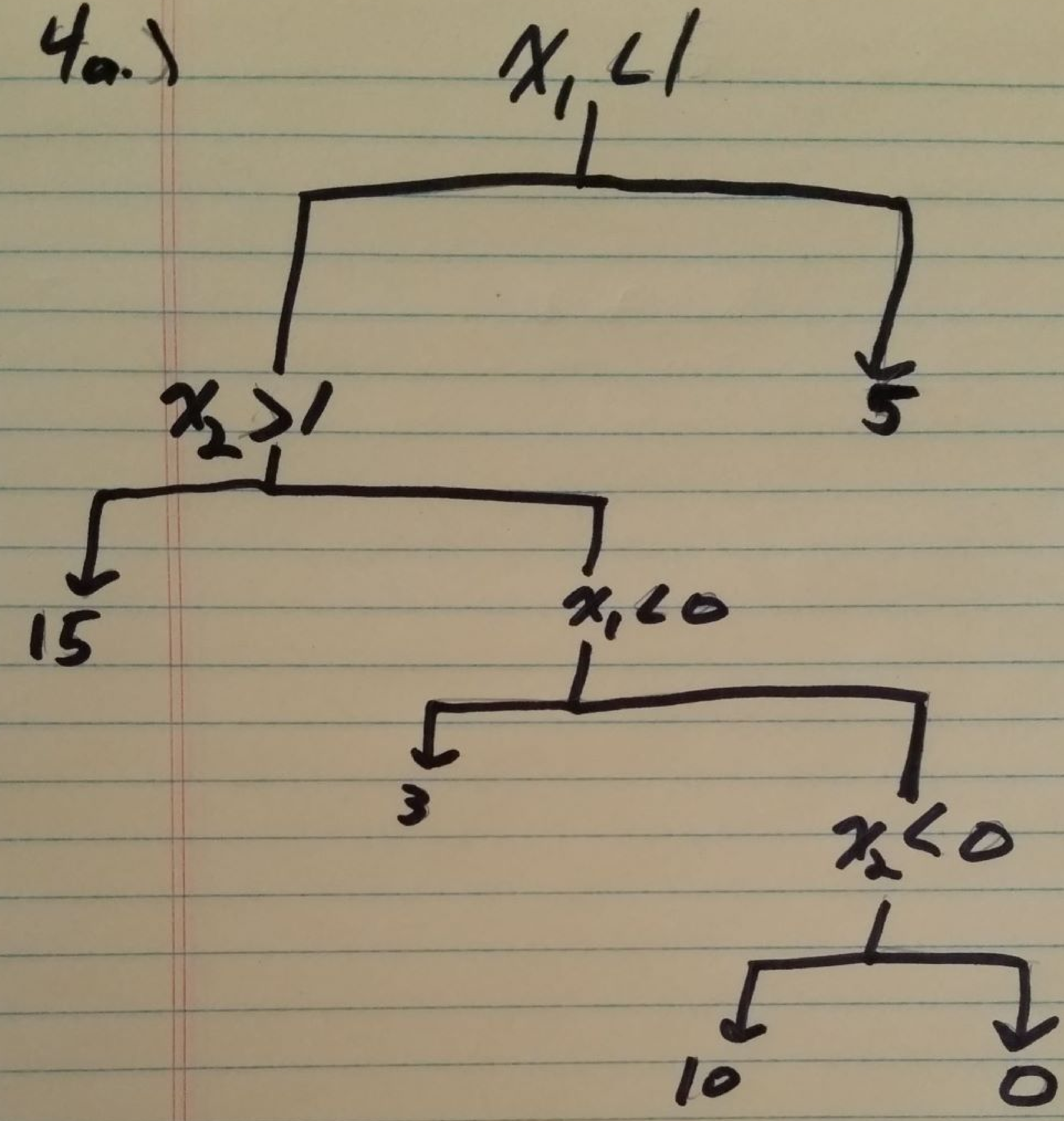
```
## [1] 0.005336179
```

(d)

Comment on your observations. How many trees were you able to simulate? How accurate are the three methods?

200 trees were simulated for bagging and random forest models. The least accurate model is the single decision tree model. The most accurate model is the random forest.

4a.)


$x_1 < 1$
$x_2 > 1$
5
15
$x_1 < 0$
3
$x_2 < 0$
10
0