# Statistical Learning HW 7 - Ch. 8 - Trees and Ensembles

*Christian Conroy*

*April 18, 2019*

## ISLR 4 (3 points)

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.

(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.

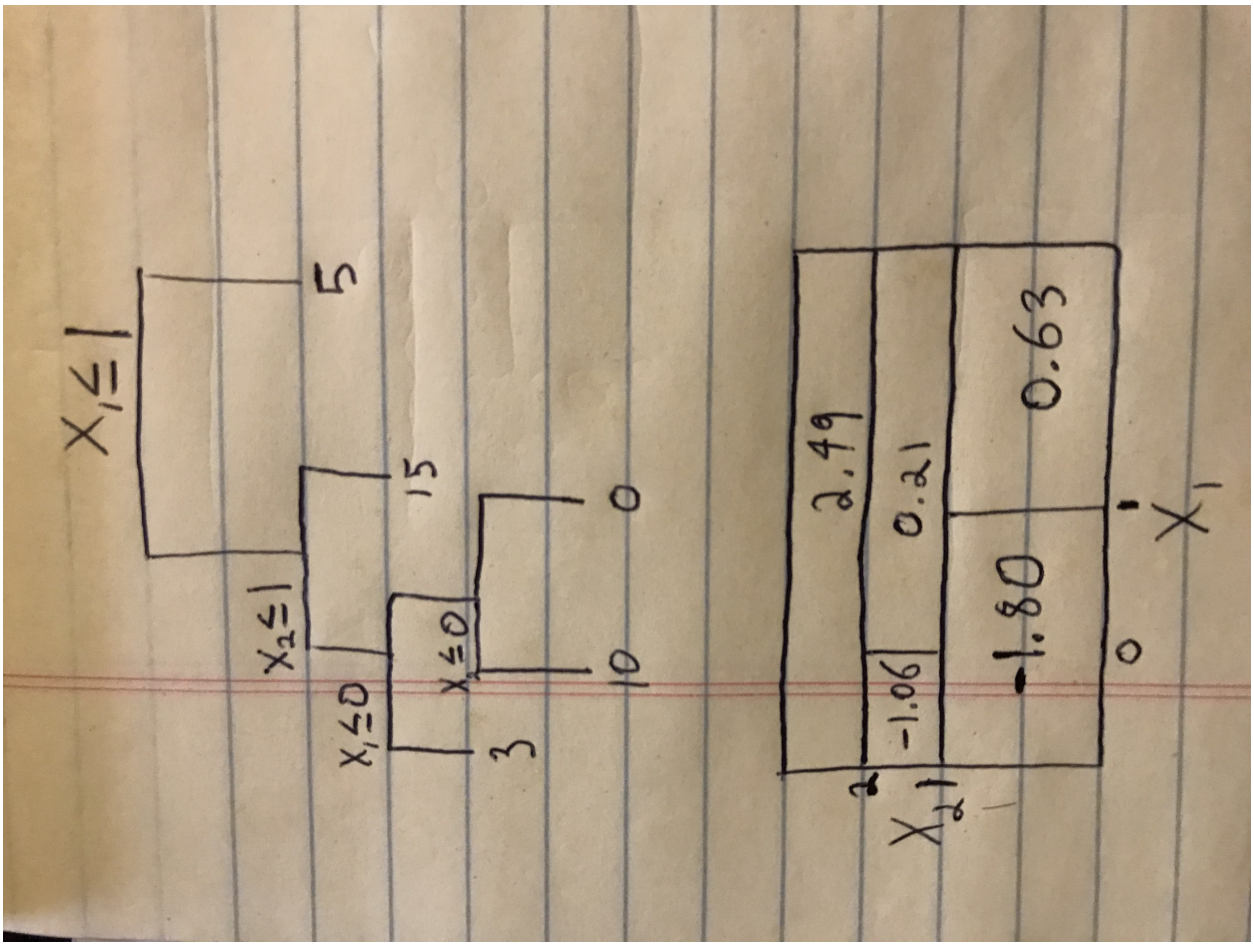



Figure 1: Alt text

## Extra 61 (3 points)

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

```
# Load in Data
concrete <- read_excel("Concrete_Data.xls")
# Clean up names
colnames(concrete) <- c("cementkg", "blustfur", "flyash", "water", "superplas", "courseagg", "fineagg",
```

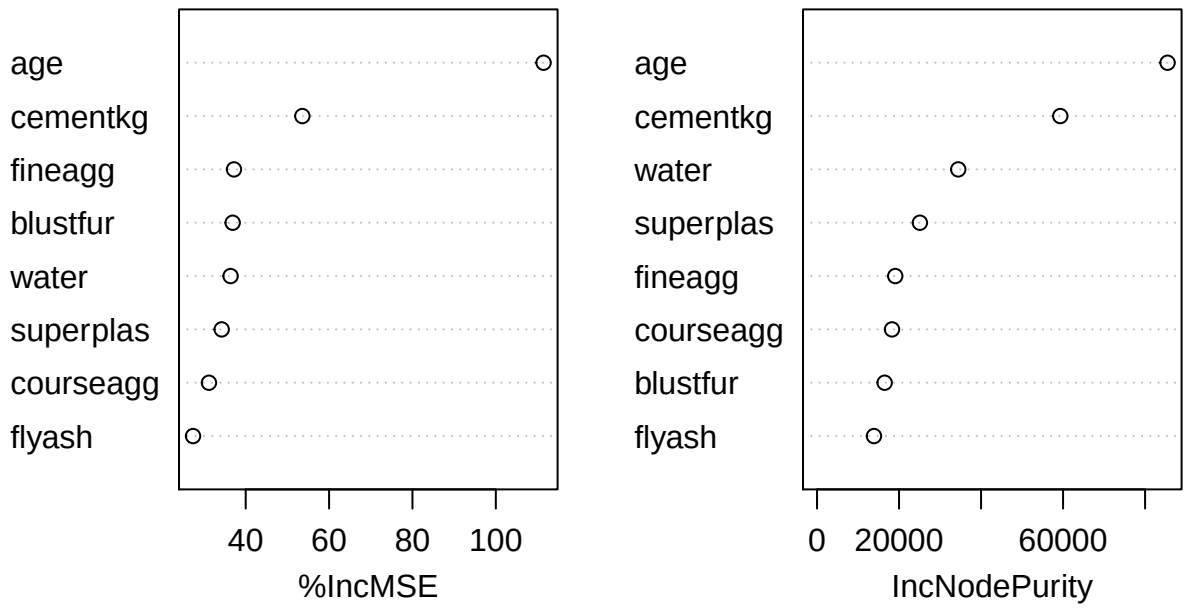
a) Fit a random forest to the data to predict strength. Estimate the rms prediction error using 10-fold cross validation. You will have to write your own cross validation code to do this.

```
# Creating the folds
vec <- runif(nrow(concrete))
folds <- cut(vec, 10, labels = FALSE)

# Affixing folds to observations
concrete$folds <- folds

# Function to train, predict, and get F1 across folds.
KCFRF <- function(k) {
  # Create a train subset consisting of all folds that do not equal k-1.
  trainspec <- concrete[concrete$folds != eval(k-1),]
 # Train the specification using the subset created.
  specreg <- randomForest(CCS ~ ., data= trainspec, importance =TRUE)
  # Create a subset for prediction equal to the fold k
  trainpred <- concrete[concrete$folds == k,]
  # Predict by applying the specification trained on all folds not equal to k-1 to the subset for predi
  trainpred$specpred <- predict(specreg, newdata = trainpred)
  # Evaluate the model by comparing the predicted versus actual values of the dependent variable.
  RMSE <- sqrt(mean((trainpred$specpred - trainpred$CCS)^2))
  return(RMSE)
}

# Run a linear regression and get accuracy for train on kth folds (1-10)
f <- 1:10
CFRF <- lapply(f, KCFRF)
CFRF2 <- do.call(rbind, CFRF)
mean(CFRF2)
```

```
## [1] 2.42
```

b) Make a variable importance plot for a random forest model for the full data.

```
concrete$folds <- NULL
rf.concrete <- randomForest(CCS ~ ., 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 observations.

```
# Creating the folds
vec <- runif(nrow(concrete))
folds <- cut(vec, 10, labels = FALSE)

# Affixing folds to observations
concrete$folds <- folds

trainspec <- concrete[folds != eval(7-1),]
specreg <- randomForest(CCS ~ ., data= trainspec, importance =TRUE)
trainpred <- concrete[folds == 7,]
trainpred$specpred <- predict(specreg, newdata = trainpred)
Comps <- data.frame(trainpred$CCS, trainpred$specpred)

# Function to train, predict, and get F1 across folds.
KCFRF <- function(k) {
  # Create a train subset consisting of all folds that do not equal k-1.
  trainspec <- concrete[concrete$folds != eval(k-1),]
 # Train the specification using the subset created.
  specreg <- randomForest(CCS ~ cementkg + blustfur + water + superplas + courseagg + fineagg + age, da
  # Create a subset for prediction equal to the fold k
  trainpred <- concrete[concrete$folds == k,]
  # Predict by applying the specification trained on all folds not equal to k-1 to the subset for predi
  trainpred$specpred <- predict(specreg, newdata = trainpred)
  # Evaluate the model by comparing the predicted versus actual values of the dependent variable.
```

```
  RMSE <- sqrt(mean((trainpred$specpred - trainpred$CCS)^2))
  return(RMSE)
}

# Run a linear regression and get accuracy for train on kth folds (1-10)
f <- 1:10
CFRF <- lapply(f, KCFRF)
CFRF2 <- do.call(rbind, CFRF)
mean(CFRF2)
```

```
## [1] 2.7
```

The estimated RMS has risen slightly.

## Extra 62 (3 points)

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

```
# Load in Data
concrete <- read_excel("Concrete_Data.xls")
# Clean up names
colnames(concrete) <- c("cementkg", "blustfur", "flyash", "water", "superplas", "courseagg", "fineagg",
# Create Train and Test
train <- sample(nrow(concrete),(nrow(concrete) * .70), replace = FALSE)
concrete_train <- concrete[train,]
concrete_test <- concrete[-train,]
```

Fit a gbm model to the training data to predict strength, for several choices of n.trees, shrinkage, 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.

```
set.seed(1)

boostnt <- function(nt) {
  boost.concrete <- gbm(CCS ~ ., data= concrete_train, n.trees=nt, interaction.depth=1, shrinkage=.001,
  yhat.train <- predict(boost.concrete, n.trees=nt)
  rmse_train <- sqrt(mean((yhat.train - concrete_train$CCS)^2))
  yhat.test <- predict(boost.concrete, newdata = concrete_test, n.trees=nt)
  rmse_test <- sqrt(mean((yhat.test - concrete_test$CCS)^2))
  full <- cbind(nt, rmse_train, rmse_test)
  return(full)
}

nt <- c(1000, 3000, 5000, 7000, 9000)

boostntchoices <- lapply(nt, boostnt)
boostntchoices <- do.call(rbind.data.frame, boostntchoices)
boostntchoices
```

```
##     nt rmse_train rmse_test
## 1 1000      13.37     14.64
## 2 3000      10.27     11.43
## 3 5000       8.61      9.71
```

```
## 4 7000       7.56      8.63
## 5 9000       6.87      7.91
```

```
boostsh <- function(sh) {
  boost.concrete <- gbm(CCS ~ ., data= concrete_train, n.trees=5000, interaction.depth=1, shrinkage= sh
  yhat.train <- predict(boost.concrete, n.trees=5000)
  rmse_train <- sqrt(mean((yhat.train - concrete_train$CCS)^2))
  yhat.test <- predict(boost.concrete, newdata = concrete_test, n.trees=5000)
  rmse_test <- sqrt(mean((yhat.test - concrete_test$CCS)^2))
  full <- cbind(sh, rmse_train, rmse_test)
  return(full)
}

sh <-  c(0.001, 0.01, 0.1, 0.2, 0.5)

boostshchoices <- lapply(sh, boostsh)
boostshchoices <- do.call(rbind.data.frame, boostshchoices)
boostshchoices
```

```
##      sh rmse_train rmse_test
## 1 0.001       8.61      9.71
## 2 0.010       4.78      6.00
## 3 0.100       3.52      5.51
## 4 0.200       3.27      5.63
## 5 0.500       3.15      5.75
```

```
boostid <- function(id) {
  boost.concrete <- gbm(CCS ~ ., data= concrete_train, n.trees=5000, interaction.depth=id, shrinkage=.0
  yhat.train <- predict(boost.concrete, n.trees=5000)
  rmse_train <- sqrt(mean((yhat.train - concrete_train$CCS)^2))
  yhat.test <- predict(boost.concrete, newdata = concrete_test, n.trees=5000)
  rmse_test <- sqrt(mean((yhat.test - concrete_test$CCS)^2))
  full <- cbind(id, rmse_train, rmse_test)
  return(full)
}

id <- c(1,2,3, 4, 5)

boostidchoices <- lapply(id, boostid)
boostidchoices <- do.call(rbind.data.frame, boostidchoices)
boostidchoices
```

```
##   id rmse_train rmse_test
## 1  1       8.60      9.71
## 2  2       6.55      7.66
## 3  3       5.63      6.83
## 4  4       5.10      6.41
## 5  5       4.74      6.14
```

It is possible to overfit, which is especially clear if we look at ntrees. As we increase the number of trees as specified by n.trees, the rmse is consistently higher for the train compared to the test. Boosting can overfit if the number of trees is too large.As we increase shrinkage, the opposite occurs, with a gap growing to the advantage of the test rmse as we increase shrinkage, As we increase the interaction depth, similarily, the test rmse looks stronger.

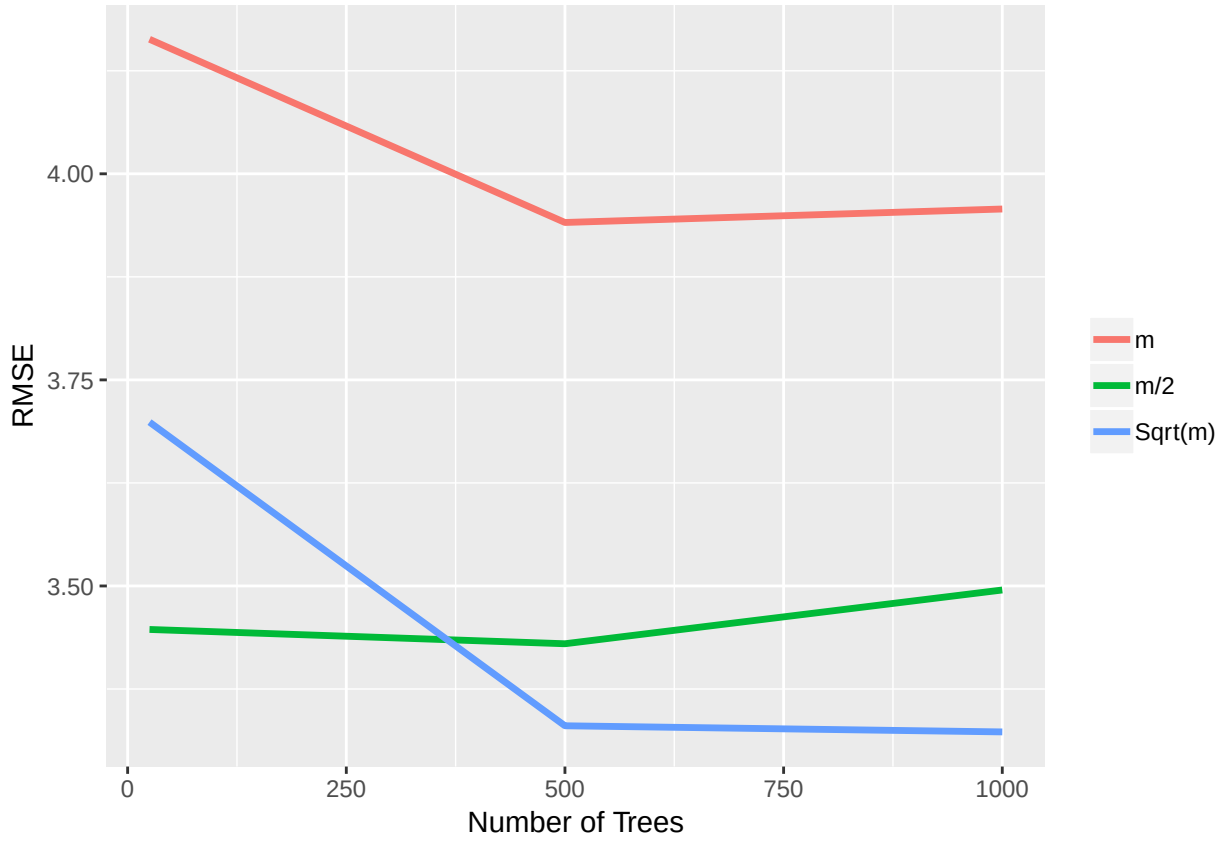
## ISLR 7 (5 points)

In the lab, we applied random forests to the Boston data 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.

```
data('Boston')

# Create Train and Test
set.seed(1)
train <- sample(nrow(Boston),(nrow(Boston) * .70), replace = FALSE)
Boston_train <- Boston[train,]
Boston_test <- Boston[-train,]
```

```
rfmt <- function(mt, nt) {
  rf.Boston <- randomForest(medv ~ ., data= Boston_train, mtry = mt, ntree = nt)
  yhat.rf = predict(rf.Boston, newdata=Boston_test)
  rmse_test <- sqrt(mean((yhat.rf-Boston_test$medv)^2))
  full <- cbind(mt, nt, rmse_test)
  return(full)
}

mtry <- c(13, 13/2, sqrt(13))
ntr <-  c(25, 500, 1000)

args <- expand.grid(mt = mtry, nt = ntr)

rfchoices <- mapply(FUN = rfmt, mt = args$mt, nt = args$nt)
```

```
rfchoices_plot <- as.data.frame(t(rfchoices))

rfchoices_plot$m <- ifelse(rfchoices_plot$V1 <= 3.62, "Sqrt(m)", ifelse(rfchoices_plot$V1 == 13, "m", i

ggplot(data = rfchoices_plot, mapping = aes(y = V3, x = V2, group = factor(m), colour=factor(m))) +
   geom_line(size=1.2) + xlab("Number of Trees") + ylab("RMSE") +  theme(legend.title = element_blank()
```




## ISLR 9 (5 points)

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.

```
data('OJ')

# Create Train and Test
set.seed(1)
train <- sample(nrow(OJ), 800, replace = FALSE)
OJ_train <- OJ[train,]
OJ_test <- 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?

```
mytree <- tree(factor(Purchase) ~ ., data = OJ_train)
summary(mytree)
```

```
##
## Classification tree:
## tree(formula = factor(Purchase) ~ ., data = OJ_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
## Number of terminal nodes:  8
```

```
## Residual mean deviance:  0.731 = 579 / 792
## Misclassification error rate: 0.165 = 132 / 800
```

The training error rate is .165 and the tree has 8 terminal nodes. [MISCLASSIFCATION ERROR, RIGHT?]

(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.

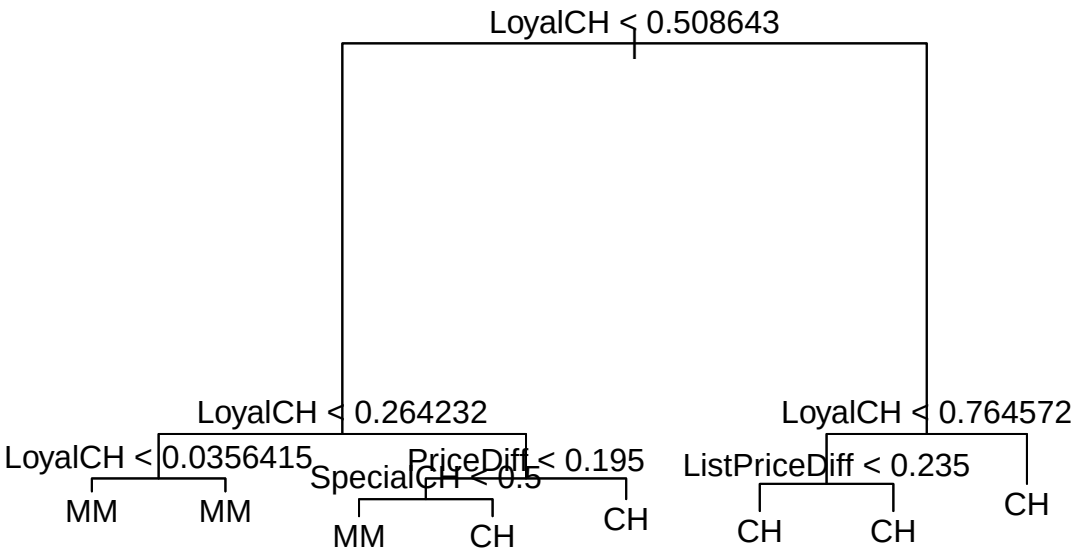
```
mytree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1000 CH ( 1 0 )
##    2) LoyalCH < 0.508643 350  400 MM ( 0 1 )
##      4) LoyalCH < 0.264232 166  100 MM ( 0 1 )
##        8) LoyalCH < 0.0356415 57   10 MM ( 0 1 ) *
##        9) LoyalCH > 0.0356415 109  100 MM ( 0 1 ) *
##      5) LoyalCH > 0.264232 184  200 MM ( 0 1 )
##       10) PriceDiff < 0.195 83   90 MM ( 0 1 )
##         20) SpecialCH < 0.5 70   60 MM ( 0 1 ) *
##         21) SpecialCH > 0.5 13   20 CH ( 1 0 ) *
##       11) PriceDiff > 0.195 101  100 CH ( 1 0 ) *
##    3) LoyalCH > 0.508643 450  300 CH ( 1 0 )
##      6) LoyalCH < 0.764572 172  200 CH ( 1 0 )
##       12) ListPriceDiff < 0.235 70  100 CH ( 1 0 ) *
##       13) ListPriceDiff > 0.235 102   70 CH ( 1 0 ) *
##      7) LoyalCH > 0.764572 278   90 CH ( 1 0 ) *
```

For the fifth node we see that we check if the value of LoyalCH is greater than 0.264. There are 184 such observations. For these values the tree predicts MM as the juice of choice with a deviance of 200.

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

```
plot(mytree)
text(mytree, pretty =0)
```




The plot matches our interpretation of the output in part c. For the fifth node we are evaluating if LoyalCH is greatet than 0.264. If it is, we evaluate again this time on whether the PriceDiff is less than or greater than 0.195. If it is less than, we evaluate again based on whether SpecialCH is less than or grater 0.5.

(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?

```
y.pred <- predict(mytree, newdata = OJ_test, type= "class")

table(OJ_test$Purchase, y.pred)
```

```
##     y.pred
##       CH  MM
##   CH 147  12
##   MM  49  62
```

```
confusionMatrix(y.pred, OJ_test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 147  49
##         MM  12  62
##
##                Accuracy : 0.774
##                  95% CI : (0.719, 0.823)
##     No Information Rate : 0.589
```

```
##     P-Value [Acc > NIR] : 1.08e-10
##
##                   Kappa : 0.509
##  Mcnemar's Test P-Value : 4.04e-06
##
##             Sensitivity : 0.925
##             Specificity : 0.559
##          Pos Pred Value : 0.750
##          Neg Pred Value : 0.838
##              Prevalence : 0.589
##          Detection Rate : 0.544
##    Detection Prevalence : 0.726
##       Balanced Accuracy : 0.742
##
##        'Positive' Class : CH
##
```

We end up with an accuracy of 0.774.

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

```
cv.mytree <- cv.tree(mytree ,FUN=prune.misclass)
names(cv.mytree)
```

```
## [1] "size"   "dev"    "k"      "method"
```

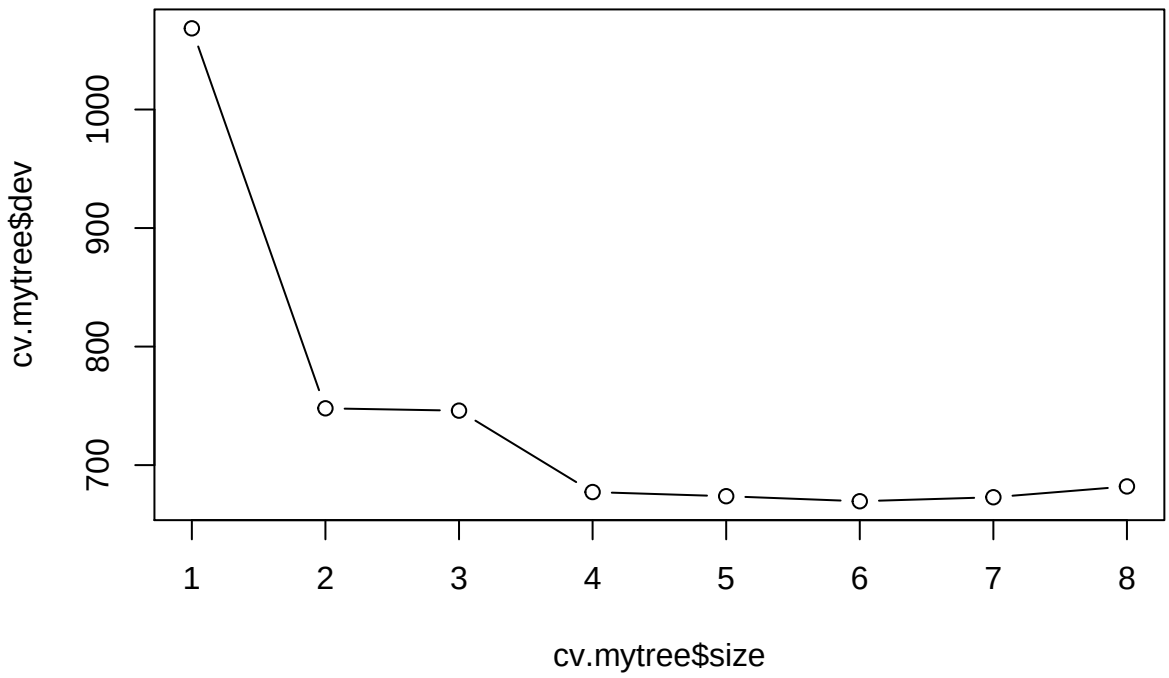
```
cv.mytree
```

```
## $size
## [1] 8 5 2 1
##
## $dev
## [1] 156 156 156 306
##
## $k
## [1]   -Inf   0.00   4.67 160.00
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

The lowest error is for a tree of size 8.

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

```
cv.mytree <- cv.tree(mytree)
plot(cv.mytree$size, cv.mytree$dev, type='b')
```




(h) Which tree size corresponds to the lowest cross-validated classification error rate?

Tree size of 8.

(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.

```
prune.mytree <- prune.tree(mytree, best=8)
summary(prune.mytree)
```

```
##
## Classification tree:
## tree(formula = factor(Purchase) ~ ., data = OJ_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff"
## Number of terminal nodes:  8
## Residual mean deviance:  0.731 = 579 / 792
## Misclassification error rate: 0.165 = 132 / 800
```

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

The training error (misclassification error rate) exactly the same for the pruned and unpruned trees.

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

```
y.pred <- predict(prune.mytree, newdata = OJ_test, type= "class")

table(OJ_test$Purchase, y.pred)
```

```
##     y.pred
##       CH  MM
##   CH 147  12
##   MM  49  62
```

```
# IS THE ACCURACY RATE HERE SERVING AS TEST ERROR OR SHOULD WE DO SOMETHING DIFFERENTLY?
confusionMatrix(y.pred, OJ_test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 147  49
##         MM  12  62
##
##                Accuracy : 0.774
##                  95% CI : (0.719, 0.823)
##     No Information Rate : 0.589
##     P-Value [Acc > NIR] : 1.08e-10
##
##                   Kappa : 0.509
##  Mcnemar's Test P-Value : 4.04e-06
##
##             Sensitivity : 0.925
##             Specificity : 0.559
##          Pos Pred Value : 0.750
##          Neg Pred Value : 0.838
##              Prevalence : 0.589
##          Detection Rate : 0.544
##    Detection Prevalence : 0.726
##       Balanced Accuracy : 0.742
##
##        'Positive' Class : CH
##
```

It is the exact same for test as well. In both training and test cases, it is because the best selection was the model with all variables.

## Extra 59 (5 Points)

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

```
mnist <-load('mnist_all.RData')

mnist_train <- data.frame(train$x, train$y)
mnist_train <- mnist_train[mnist_train$train.y == 4 | mnist_train$train.y == 5,]

mnist_test <- data.frame(test$x, test$y)
mnist_test <- mnist_test[mnist_test$test.y == 4 | mnist_test$test.y == 5,]
```

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.

```r
mytree <- tree(factor(train.y) ~ ., data = mnist_train)
summary(mytree)
```

```
##
## Classification tree:
## tree(formula = factor(train.y) ~ ., data = mnist_train)
## Variables actually used in tree construction:
##  [1] "X597" "X327" "X325" "X357" "X324" "X436" "X438" "X409" "X323" "X517"
## [11] "X355"
## Number of terminal nodes:  12
## Residual mean deviance:  0.222 = 2490 / 11300
## Misclassification error rate: 0.0321 = 361 / 11263
```

```r
# Why is prediction not working here?
y.pred <- predict(mytree, newdata = mnist_test, type= "class")

table(mnist_test$test.y, y.pred)
```

```
##    y.pred
##       4   5
##   4 959  23
##   5  40 852
```

```r
confusionMatrix(y.pred, mnist_test$test.y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   4   5
##          4 959  40
##          5  23 852
##
##                Accuracy : 0.966
##                  95% CI : (0.957, 0.974)
##     No Information Rate : 0.524
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.933
##  Mcnemar's Test P-Value : 0.0438
##
##             Sensitivity : 0.977
##             Specificity : 0.955
##          Pos Pred Value : 0.960
##          Neg Pred Value : 0.974
##              Prevalence : 0.524
##          Detection Rate : 0.512
##    Detection Prevalence : 0.533
##       Balanced Accuracy : 0.966
##
##        'Positive' Class : 4
##
```

Accuracy of 0.966.

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.

```
rf.mnist <- randomForest(factor(train.y) ~ .,data= mnist_train, mtry=100, ntree=25, importance =TRUE)

yhat.rf = predict(rf.mnist ,newdata= mnist_test)

table(mnist_test$test.y, yhat.rf)
```

```
##    yhat.rf
##       4   5
##   4 981   1
##   5   1 891
```

```
confusionMatrix(yhat.rf, mnist_test$test.y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   4   5
##          4 981   1
##          5   1 891
##
##                Accuracy : 0.999
##                  95% CI : (0.996, 1)
##     No Information Rate : 0.524
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.998
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.999
##             Specificity : 0.999
##          Pos Pred Value : 0.999
##          Neg Pred Value : 0.999
##              Prevalence : 0.524
##          Detection Rate : 0.523
##    Detection Prevalence : 0.524
##       Balanced Accuracy : 0.999
##
##        'Positive' Class : 4
##
```

Accuracy of 0.997.

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

```
bag.mnist <- randomForest(factor(train.y) ~.,data=mnist_train, mtry=784, ntree=25)
summary(bag.mnist)
```

```
##                 Length Class  Mode
## call               5  -none- call
## type               1  -none- character
## predicted      11263  factor numeric
## err.rate          75  -none- numeric
## confusion          6  -none- numeric
## votes          22526  matrix numeric
## oob.times      11263  -none- numeric
```

```
## classes             2  -none- character
## importance        784  -none- numeric
## importanceSD        0  -none- NULL
## localImportance     0  -none- NULL
## proximity           0  -none- NULL
## ntree               1  -none- numeric
## mtry                1  -none- numeric
## forest             14  -none- list
## y               11263  factor numeric
## test                0  -none- NULL
## inbag               0  -none- NULL
## terms               3  terms  call
```

```
yhat.bag <- predict (bag.mnist, newdata = mnist_test)

table(mnist_test$test.y, yhat.bag)
```

```
##    yhat.bag
##       4   5
##   4 975   7
##   5   8 884
```

```
confusionMatrix(yhat.bag, mnist_test$test.y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   4   5
##          4 975   8
##          5   7 884
##
##                Accuracy : 0.992
##                  95% CI : (0.987, 0.996)
##     No Information Rate : 0.524
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.984
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.993
##             Specificity : 0.991
##          Pos Pred Value : 0.992
##          Neg Pred Value : 0.992
##              Prevalence : 0.524
##          Detection Rate : 0.520
##    Detection Prevalence : 0.525
##       Balanced Accuracy : 0.992
##
##        'Positive' Class : 4
##
```

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

All three models were very accurate - 0.966, 0.997, and .986, respectively. The accuracy improved through each model, with the bagging model in c) having the best accuracy.