Lab: Trees, Random Forests, Boosting

STAT 430, Spring 2017

For this lab you will need the following packages:

```r
library(ISLR)
library(tree)
library(rpart)
library(rpart.plot)
library(randomForest)
library(gbm)
library(caret)
library(plyr)
```

You will investigate the use of trees, random forests, and boosting on the `OJ` data from the `ISLR` package. The categorical response is `Purchase`, while all other variables are used as predictors. Below we create a test-train split of the data.

```r
set.seed(42)
oj_idx = createDataPartition(OJ$Purchase, p = 0.5, list = FALSE)
oj_trn = OJ[oj_idx,]
oj_tst = OJ[-oj_idx,]
```

Since we are performing classification, we will calculate several accuracies.

```r
accuracy = function(actual, predicted) {
  mean(actual == predicted)
}
```

> Fit a tree to the training data using the `tree()` function, 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? What variables are used?

```r
# your code here
```

```r
oj_tree = tree(Purchase ~ ., data = oj_trn)
summary(oj_tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = oj_trn)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "StoreID"       "ListPriceDiff"
## Number of terminal nodes:  9
## Residual mean deviance:  0.6781 = 357.3 / 527
## Misclassification error rate: 0.1437 = 77 / 536
```

> 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
# your code here
```

```r
oj_tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
```

```
##
##  1) root 536 716.90 CH ( 0.61007 0.38993 )
##    2) LoyalCH < 0.50395 233 270.00 MM ( 0.26609 0.73391 )
##      4) LoyalCH < 0.277977 119  77.81 MM ( 0.10084 0.89916 ) *
##      5) LoyalCH > 0.277977 114 156.30 MM ( 0.43860 0.56140 )
##       10) PriceDiff < 0.195 46  39.23 MM ( 0.15217 0.84783 ) *
##       11) PriceDiff > 0.195 68  89.45 CH ( 0.63235 0.36765 ) *
##    3) LoyalCH > 0.50395 303 228.80 CH ( 0.87459 0.12541 )
##      6) LoyalCH < 0.705326 100 118.60 CH ( 0.72000 0.28000 )
##       12) PriceDiff < 0.265 57  78.58 CH ( 0.54386 0.45614 )
##         24) StoreID < 5.5 43  59.03 MM ( 0.44186 0.55814 ) *
##         25) StoreID > 5.5 14  11.48 CH ( 0.85714 0.14286 ) *
##       13) PriceDiff > 0.265 43  16.18 CH ( 0.95349 0.04651 ) *
##      7) LoyalCH > 0.705326 203  79.71 CH ( 0.95074 0.04926 )
##       14) PriceDiff < 0.31 142  72.34 CH ( 0.92958 0.07042 )
##         28) ListPriceDiff < 0.135 46   0.00 CH ( 1.00000 0.00000 ) *
##         29) ListPriceDiff > 0.135 96  64.16 CH ( 0.89583 0.10417 ) *
##       15) PriceDiff > 0.31 61   0.00 CH ( 1.00000 0.00000 ) *
```

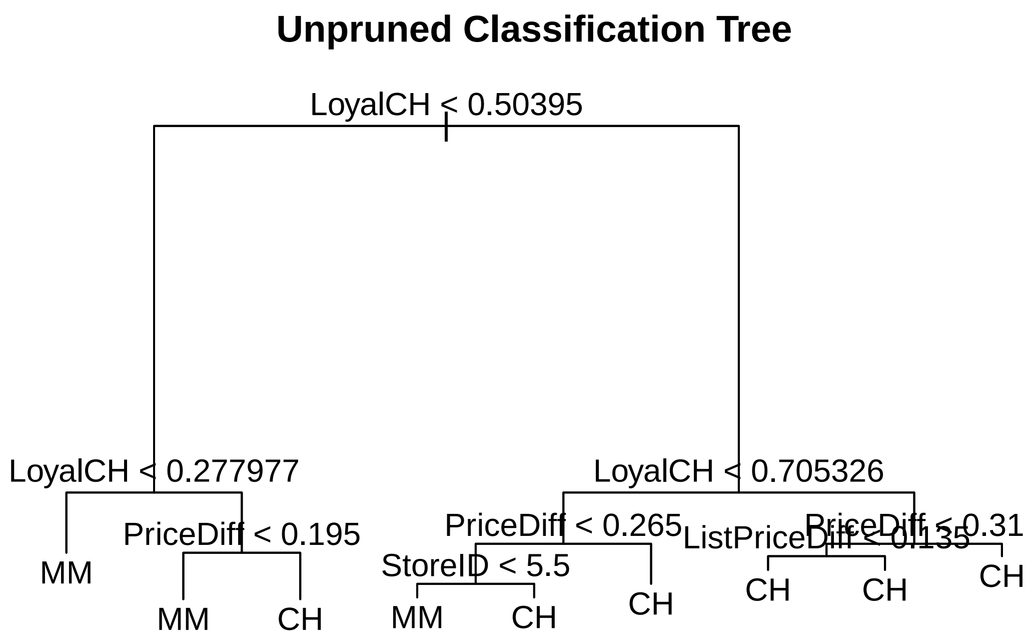
```
4) LoyalCH < 0.277977 119  77.81 MM ( 0.10084 0.89916 ) *
```

Observations in this node have `LoyalCH < 0.277977` as well as the properties of all the previous splits. There are `119` observations in this node. We predicted the class `MM` for observations following these splits since `0.89916` of the observations in this node have class `MM`.

Create a plot of the tree you fit. Interpret the results.

```r
# your code here
```

```r
plot(oj_tree)
text(oj_tree, pretty = 0)
title(main = "Unpruned Classification Tree")
```



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 accuracy?

```
# your code here
```

```
oj_tree_tst_pred = predict(oj_tree, oj_tst, type = "class")
table(pred = oj_tree_tst_pred, act = oj_tst$Purchase)
```

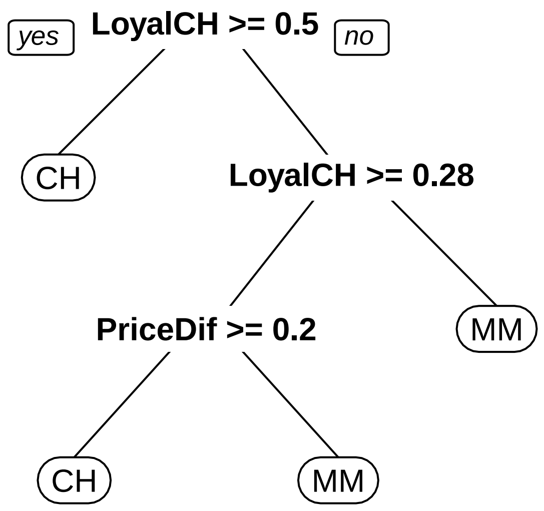
```
##     act
## pred  CH  MM
##   CH 267  59
##   MM  59 149
```

The code below trains a single tree using 5-fold cross-validation. (It will consider three different complexity parameters.) Also, instead of `tree()`, we use `rpart()` together with the `caret` package.

```
cv_5 = trainControl(method = "cv", number = 5)
oj_tree_cv  = train(Purchase ~ .,
                    data = oj_trn,
                    trControl = cv_5,
                    method = "rpart")
```

We also create a plot of the resulting tree.

```
prp(oj_tree_cv$finalModel)
```



Predict the response on the test data using this new tree. What is the test accuracy? Does this perform better than the previous (unpruned) tree? Is this tree smaller or larger?

```
# your code here
```

```
accuracy(predict(oj_tree_cv, oj_tst), oj_tst$Purchase)
```

```
## [1] 0.7940075
```

Now we'll consider ensemble methods. The following defines OOB resampling for use when tuning a random forest.

```
oob = trainControl(method = "oob")
```

Tune a random forest using OOB resampling and **all** possible values of `mtry`. What value of `mtry` is found to be the best?

```
# your code here
```

```
rf_grid = expand.grid(mtry = 1:(ncol(oj_trn) - 1))
oj_rf_oob  = train(Purchase ~ .,
                   data = oj_trn,
                   trControl = oob,
                   method = "rf",
                   tuneGrid = rf_grid)
oj_rf_oob$bestTune
```

```
##   mtry
## 9    9
```

Report the OOB classification accuracy for a bagged tree model.

```r
# your code here
```

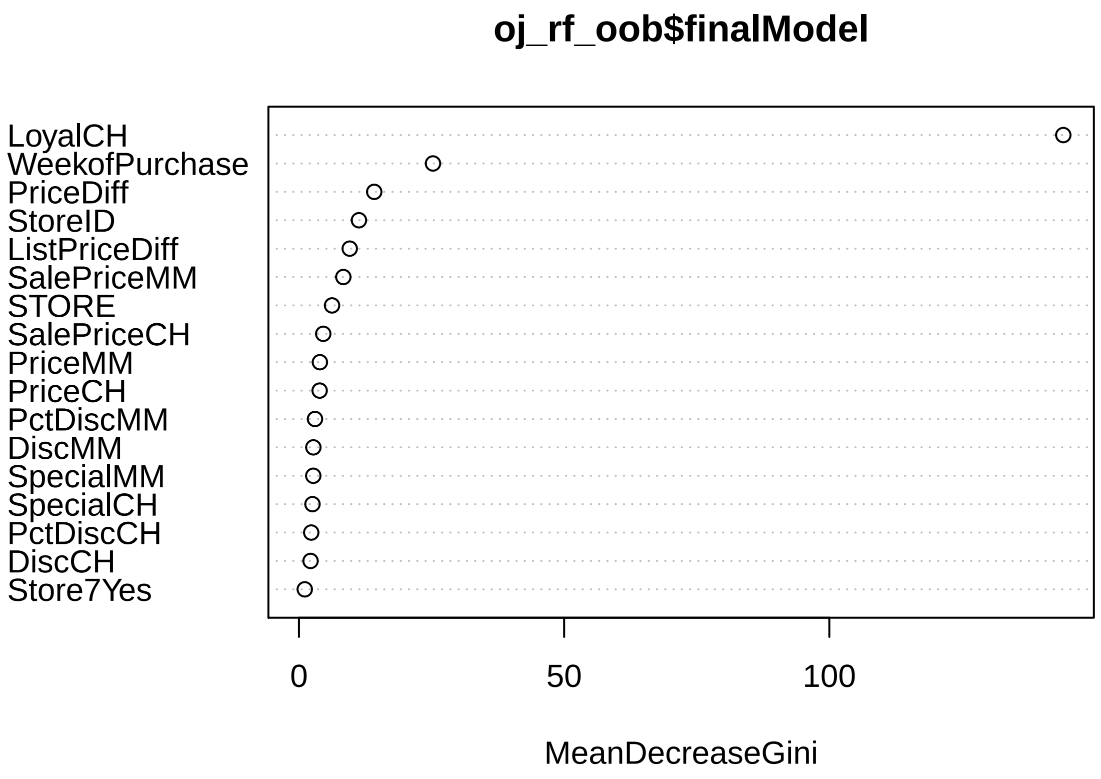
```r
oj_rf_oob$results[ncol(oj_trn) - 1, ]$Accuracy
```

```
## [1] 0.8152985
```

Create a variable importance plot for the random forest you tuned.

```r
# your code here
```

```r
varImpPlot(oj_rf_oob$finalModel)
```



Report the test accuracy for the tuned random forest.

```r
# your code here
```

```r
accuracy(predict(oj_rf_oob, oj_tst), oj_tst$Purchase)
```

```
## [1] 0.7771536
```

Use the following tuning grid to tune a boosted tree model using 5-fold cross-validation.

```r
gbm_grid_small = expand.grid(interaction.depth = c(1, 2),
                             n.trees = c(500, 1000, 1500),
                             shrinkage = c(0.01, 0.1),
                             n.minobsinnode = 10)
```

```r
# your code here
```

```
oj_gbm_small_cv  = train(Purchase ~ .,
                   data = oj_trn,
                   trControl = cv_5,
                   method = "gbm",
                   verbose = FALSE,
                   tuneGrid = gbm_grid_small)
```

Report the **two** most important variables according to the tuned boosted model. (Attempt to suppress the plot that is usually created.)

```
# your code here
```

```
summary(oj_gbm_small_cv, plotit = FALSE)[1:2, ]
```

```
##                 var   rel.inf
## LoyalCH     LoyalCH 77.299084
## PriceDiff PriceDiff  9.427254
```

```
# summary(oj_gbm_small_cv, plotit = FALSE)
```

What were the tuning parameters of the chosen boosted model?

```
# your code here
```

```
oj_gbm_small_cv$bestTune
```

```
##   n.trees interaction.depth shrinkage n.minobsinnode
## 4     500                 2      0.01             10
```

Calculate the test accuracy for the chosen boosted model.

```
# your code here
```

```
accuracy(predict(oj_gbm_small_cv, oj_tst), oj_tst$Purchase)
```

```
## [1] 0.8314607
```

Create a larger tuning grid and fit a new boosted model. Report the tuning parameters and new test accuracy. Is this new model better? Is this model different?

```
# your code here
```

```
gbm_grid_large = expand.grid(interaction.depth = c(1, 2, 3),
                             n.trees = c(500, 1000, 1500),
                             shrinkage = c(0.001, 0.01, 0.1),
                             n.minobsinnode = 10)

oj_gbm_large_cv  = train(Purchase ~ .,
                         data = oj_trn,
                         trControl = cv_5,
                         method = "gbm",
                         verbose = FALSE,
                         tuneGrid = gbm_grid_large)

oj_gbm_large_cv$bestTune
```

```
##   n.trees interaction.depth shrinkage n.minobsinnode
## 7     500                 3     0.001             10
```

```
accuracy(predict(oj_gbm_large_cv, oj_tst), oj_tst$Purchase)
```

```
## [1] 0.7940075
```

Of all the models you have fit, which is best?