MISSISSIPPI STATE UNIVERSITY

Data Analysis One

Emily Hubbard

October 23, 2025

HW 5

Executive Summary:

A. Introduction:
Understanding what drives consumer brand choice helps predict market behavior. For example: Do discounts on a particular brand increase purchase likelihood? Does brand loyalty outweigh price sensitivity? How does changing the list price influence the final decision? Using the OJ dataset in R, we analyze which factors most influence a customer's choice between Minute Maid (MM) and Citrus Hill (CH). We will fit multiple machine-learning methods and compare their test data performance:

1. Unpruned Decision Tree
2. Pruned Decision Tree
3. Bagging (aggregation of trees)
4. Random Forests
5. Boosting Tree Method
6. Support Vetor Machines (SVMs) - Linear
7. Support Vetor Machines (SVMs) - Non-linear

For each model, we will train on a portion of the data and test on the remaining observations. A confusion matrix, accuracy and misclassification rates, and interpretations will be produced for each model. After fitting and testing all models, we will use comparative tools to evaluate which model does the best job of predicting the classification of the binary response variable Purchase (CH vs. MM).

The question is: Among these models, which will produce the greatest prediction power?

B. Data Collection

- **"Purchase"** A factor with levels CH and MM indicating whether the customer purchased Citrus Hill or Minute Maid Orange Juice.
- **"WeekofPurchase"** Week of Purchase
- **"StoreID"** Store ID
- **"PriceCH"** Price charged for CH
- **"PriceMM"** Price charged for MM
- **"DiscCH"** Discount offered for CH
- **"DiscMM"** Discount offered for MM
- **"SpecialCH"** Indicator of special on CH
- **"SpecialMM"** Indicator of special on MM
- **"LoyalCH"** Customer brand loyalty for CH
- **"SalePriceMM"** Sale price for MM
- **"SalePriceCH"** Sale price for CH
- **"PriceDiff"** Sale price of MM less sale price of CH
- **"Store7"** A factor with levels No and Yes indicating whether the sale is at Store 7.
- **"PctDiscMM"** Percentage discount for MM
- **"PctDiscCH"** Percentage discount for CH
- **"ListPriceDiff"** List price of MM less list price of CH
- **"STORE"** Which of 5 possible stores the sale occured at

C. Summary Information

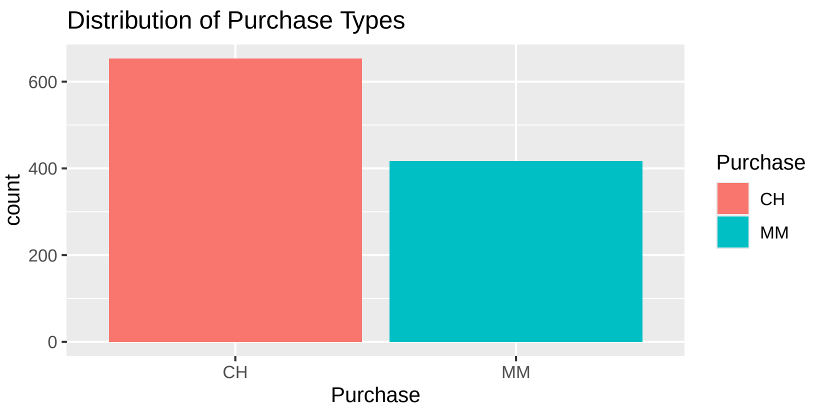


Figure 0.1: Visualization of OJ Data Binary Response Variable

This figure visualizes the proportion of the response variables that fall in the "CM" category versus the "MM" category.

Correlation Chart for OJ Numerical Data:

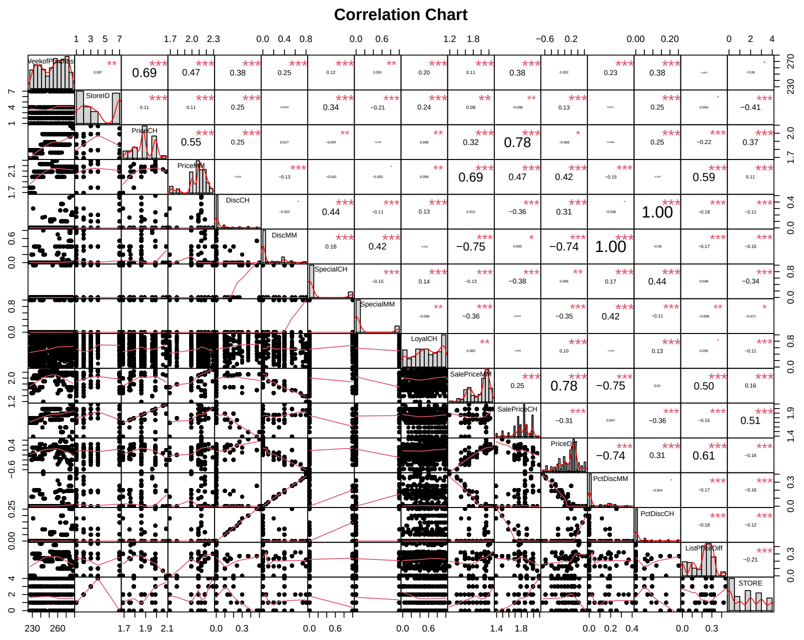


Figure 0.2: Correlation Chart

Observations:

This plot is busy, but the key takeaway is the many high correlations. There are 15 correlation coefficients that are greater than |0.5| which indicates substantial multicollinearity across the variables in the data set. Tree-based models handle multicollinearity really well. Instead of trying to use every overlapping variable at once, a tree just picks one at a time and mostly ignores the duplicates. When you use a collection of models (bagging, random forests, boosting), this gets even better: different trees may pick different collinear variables and averaging their decisions decreases variance. In short, tree methods won't get confused. They will pick what's useful, skip the redundancy, and still make solid predictions. This is why these methods will be good to use on the OJ data set. SVMs will also be a useful tool to draw the "best possible line" (or boundary) that separates the two classes with the biggest safety margin between them. Because of this margin maximization, SVMs also handle multicollinearity well and will be a useful tool in analyzing this data set.

Before we get started, I want to define the binary response variable:

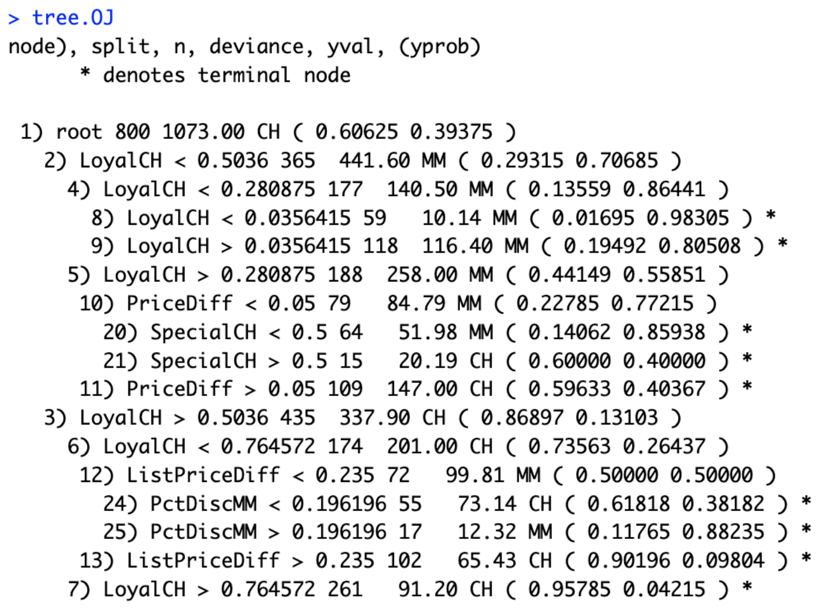
$$Y = \begin{cases} 0 & \text{if Citrus Hill is purchased} \\ 1 & \text{if Minute Maid is purchased} \end{cases}$$

D. Report Body

1. **Unpruned Decision Tree**

   To begin, we will create a decision tree without any pruning, leaving the tree with all of its terminal nodes.

   The following is the decision tree in text form.



```
> tree.OJ
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 800 1073.00 CH ( 0.60625 0.39375 )
   2) LoyalCH < 0.5036 365  441.60 MM ( 0.29315 0.70685 )
     4) LoyalCH < 0.280875 177  140.50 MM ( 0.13559 0.86441 )
       8) LoyalCH < 0.0356415 59   10.14 MM ( 0.01695 0.98305 ) *
       9) LoyalCH > 0.0356415 118  116.40 MM ( 0.19492 0.80508 ) *
     5) LoyalCH > 0.280875 188  258.00 MM ( 0.44149 0.55851 )
      10) PriceDiff < 0.05 79   84.79 MM ( 0.22785 0.77215 )
        20) SpecialCH < 0.5 64   51.98 MM ( 0.14062 0.85938 ) *
        21) SpecialCH > 0.5 15   20.19 CH ( 0.60000 0.40000 ) *
      11) PriceDiff > 0.05 109  147.00 CH ( 0.59633 0.40367 ) *
   3) LoyalCH > 0.5036 435  337.90 CH ( 0.86897 0.13103 )
     6) LoyalCH < 0.764572 174  201.00 CH ( 0.73563 0.26437 )
      12) ListPriceDiff < 0.235 72   99.81 MM ( 0.50000 0.50000 )
        24) PctDiscMM < 0.196196 55   73.14 CH ( 0.61818 0.38182 ) *
        25) PctDiscMM > 0.196196 17   12.32 MM ( 0.11765 0.88235 ) *
      13) ListPriceDiff > 0.235 102   65.43 CH ( 0.90196 0.09804 ) *
     7) LoyalCH > 0.764572 261   91.20 CH ( 0.95785 0.04215 ) *
```

Figure 0.3: Unpruned Tree Text Output

Explanation:
We can interpret each terminal node by looking at the previous nodes it falls under. For example, when looking at terminal node 11, we start at the node 2. This node states that these consumers have below average (< 0.50) loyalty to CH and are more likely to choose MM. This node is then broken down into customers with loyalty less than or greater than 0.28. Our terminal node falls underneath the customers with slightly more brand loyalty (> 0.28). This node states that even if CH has a slightly higher price (> 5 cents), 60 percent of the customers would still choose to purchase CH. This node does have a fairly high deviance of 147. This indicates a higher impurity in this node and a greater error rate when looking at this node alone.

Looking at a visual representation like the figure below is much simpler to interpret.

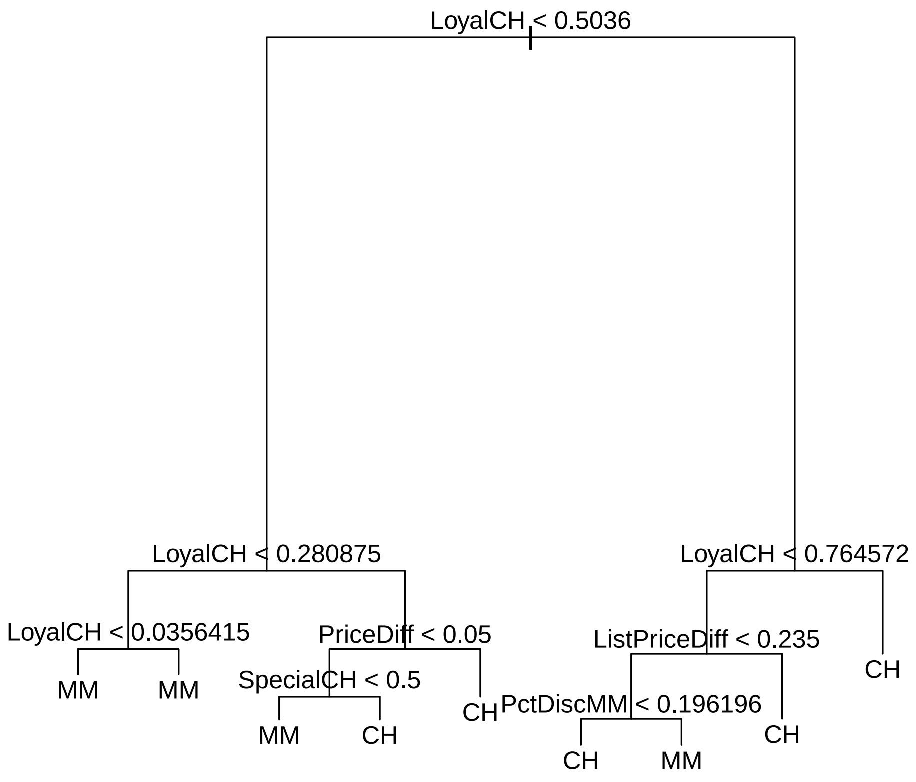
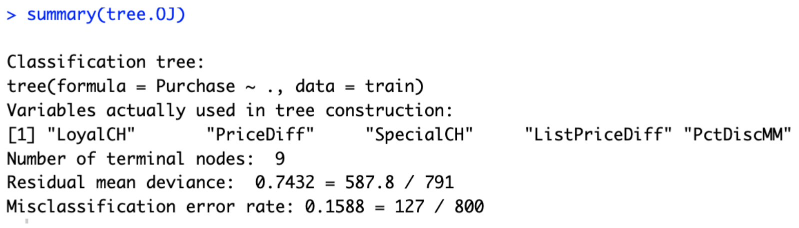


Figure 0.4: Unpruned Tree

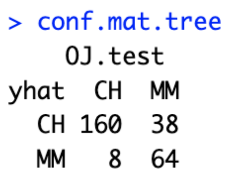
Explanation:
With this visual we can clearly count 9 terminal nodes. The more nodes we have, the more difficult to interpret. This is why it is important to consider pruning the decision tree, creating fewer terminal nodes with less complex leaves.

Below, on the left hand side, is a summary output of the model, giving us relevant information and informing us on the training misclassification rate. To the right, we have the decision matrix which allows us to see where our model went wrong. It also allows us to caluculate the accuracy and miscalssification rates of the model on our test data.



```
> summary(tree.OJ)

Classification tree:
tree(formula = Purchase ~ ., data = train)
Variables actually used in tree construction:
[1] "LoyalCH"       "PriceDiff"     "SpecialCH"     "ListPriceDiff" "PctDiscMM"
Number of terminal nodes:  9
Residual mean deviance:  0.7432 = 587.8 / 791
Misclassification error rate: 0.1588 = 127 / 800
```

Figure 0.5: Unpruned Tree Summary Output



```
> conf.mat.tree
      OJ.test
yhat   CH  MM
  CH  160  38
  MM    8  64
```

Figure 0.6: Unpruned Tree CM

Explanation:
It is important to not that this model only used 5 of the 17 covariates to produce this tree. As mentioned before, this is helpful in reducing multicollinearity effects because the decision tree is calculated using only the most relevant covariates at a given decision point.

The summary output also reports a training misclassification rate of 15.88 percent.

On test data, according to the confusion matrix, the model predicted the response values with the following amount of accuracy:

$$\text{Accuracy Rate} = 0.8296$$
$$\text{Misclassification Rate} = 0.1704$$

Overall, this model does a fair job at predicting the appropriate classification for the response variable with an accuracy of approximately 83 percent.

Now, we will find out if we can increase the accuracy of the decision tree by finding the optimal number of nodes for this data and pruning our decision tree accordingly.

2. **Pruned Decision Tree**

To find the correct size of the tree, a cross-validation must be preformed to see which size corresponds to the lowest deviance. The following graph shows us number of nodes in relation to magnitude of deviance.

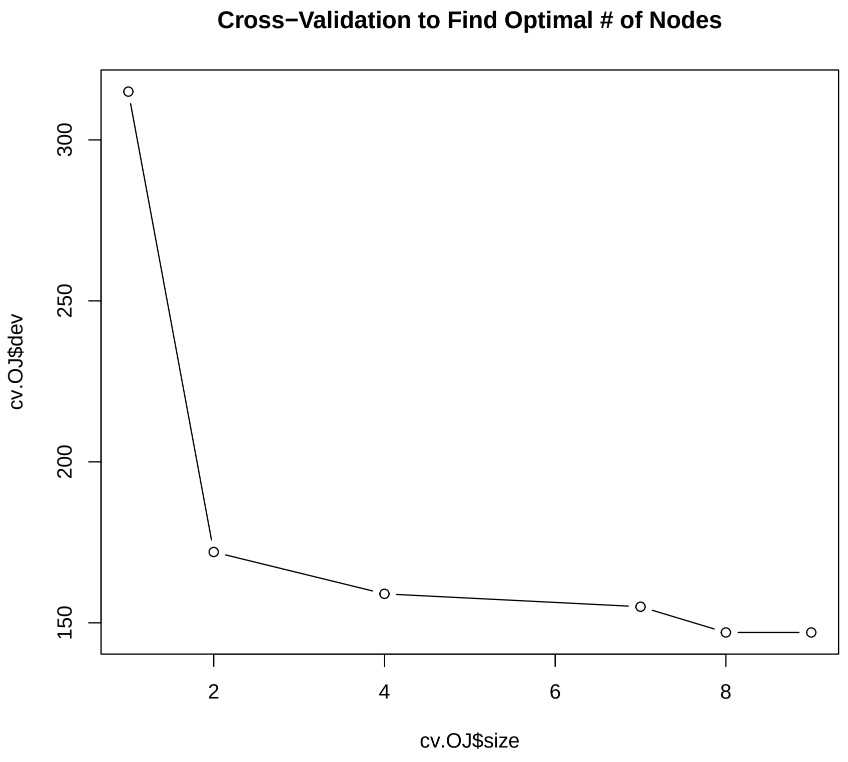


Figure 0.7: Cross-Validation to Find Optimal Number of Nodes

Explanation:
Based on this graph as well as an R function that extracts the exact number of nodes that corresponds to the lowest deviance, the best tree size has 9 terminal nodes. This means that, in this case, the lowest deviance actually occurs with an unpruned tree. Because of this, for the sake of comparison, we will create a model with the best tree size set to 5 nodes.

The following model is the visual representation of a tree with 5 terminal nodes:

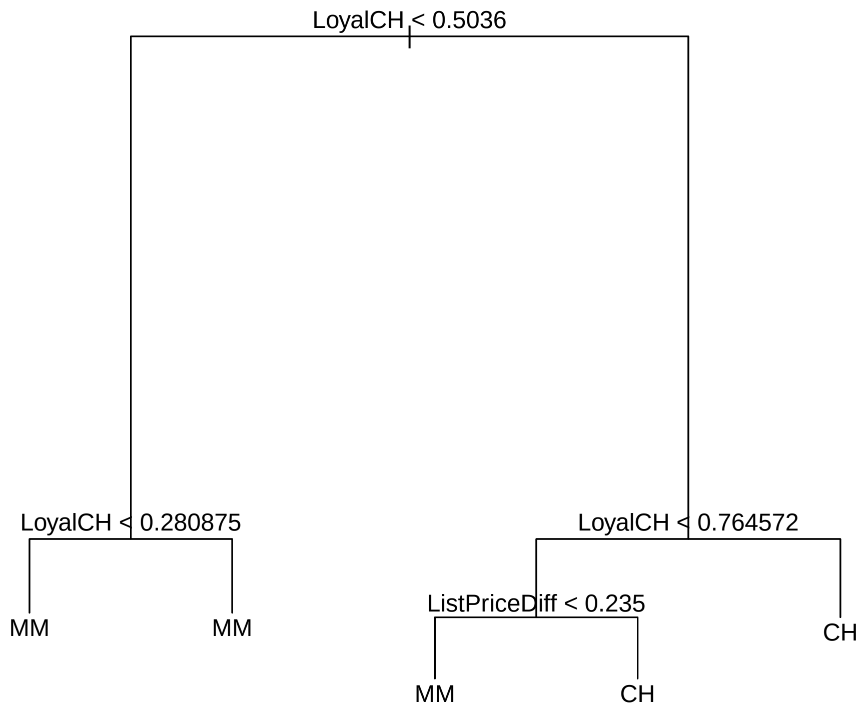
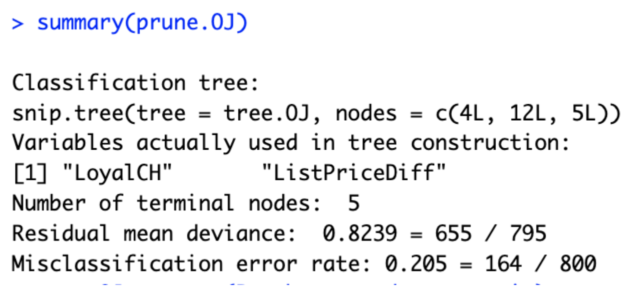


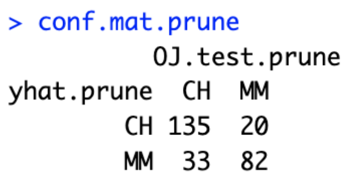
Figure 0.8: Pruned Tree

Explanation:
By visualizing alone, it is easy to see why it would be easier to interpret. It is less crowded and has fewer decision splits. However, ease of interpretability does not always equal greatest predictive power. We will see this as we analyze the tree in the following outputs.



```
> summary(prune.OJ)

Classification tree:
snip.tree(tree = tree.OJ, nodes = c(4L, 12L, 5L))
Variables actually used in tree construction:
[1] "LoyalCH"       "ListPriceDiff"
Number of terminal nodes:  5
Residual mean deviance:  0.8239 = 655 / 795
Misclassification error rate: 0.205 = 164 / 800
```

Figure 0.9: Pruned Tree Summary Output



```
> conf.mat.prune
          OJ.test.prune
yhat.prune  CH  MM
        CH 135  20
        MM  33  82
```

Figure 0.10: Pruned Tree CM

Explanation:
Compared to the unpruned tree, this model used 3 less covariates for construction, meaning only 2 covariates were considered of the 17 covariates to start.

The summary output reports a training misclassification rate of 20.50 percent. This makes sense given that 9 terminal nodes is optimal. This tree has less nodes with fewer decision points and in this case, it led to a higher error rate in the training set.

On test data specifically, according to the confusion matrix, the model predicted the response values with the following amount of accuracy:

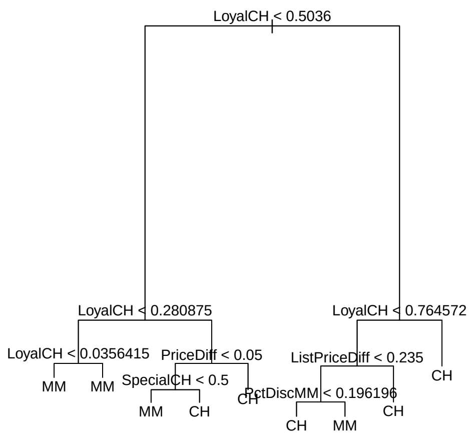
$$\text{Accuracy Rate} = 0.8037$$
$$\text{Misclassification Rate} = 0.1962$$

This model has an approximate 3 percent higher misclassification rate than that of the unpruned decision tree. Looking at these two models alone, it is clear that the 9 node model is superior.

We will now look at a few visuals and reemphasize important statistics that allow us to make a decision on which model to superior

Comparison of Unpruned vs Pruned Tree Models:

Models:



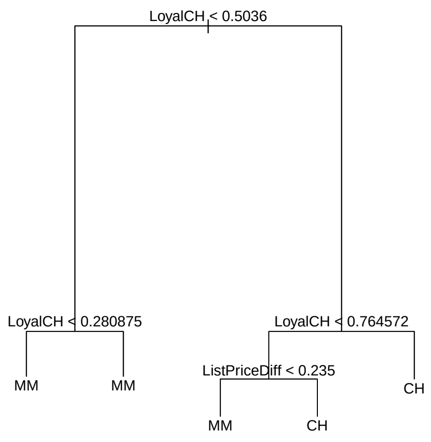


Figure 0.11: Comparison of Unpruned vs Pruned Tree

Explanation:
Visually, the tree with fewer nodes is more appealing but statistically less powerful. The training error rate within these trees as well as the difference in the two is as follows:

Training Error Rate (Unpruned) = 0.1588

Training Error Rate (Pruned) = 0.2050

Difference = 0.0462

The pruned tree has a training error rate 4.62 percent higher than the unpruned tree.

Matrices:

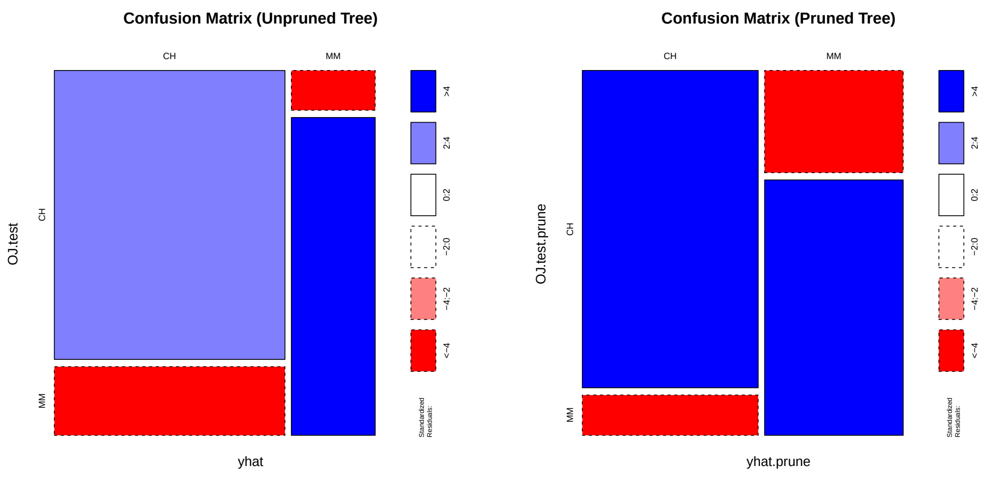


Figure 0.12: Comparison of Confusion Matrices

Explanation:
Ther are a few small things to note on these visualized confusion matrices. One, the lighter the shade that closer the precidcted values were to the actualy values (lower deviance). This means that the lighter blue color should give us higher confidence in the unpruned model. Two, the pruned tree misclassifies MM purchases more often seen by the larger red tile on the top right hand side.

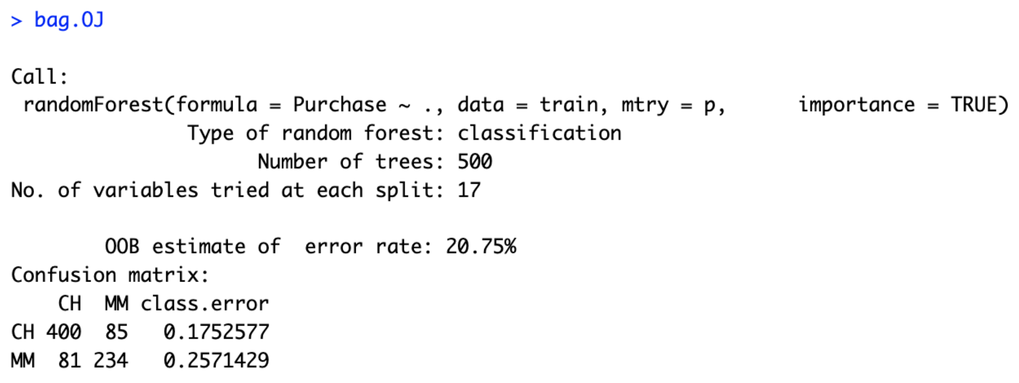
$$\text{Testing Error Rate (Unpruned)} = 0.1704$$
$$\text{Testing Error Rate (Pruned)} = 0.1962$$
$$\text{Difference} = 0.0258$$

The pruned tree also has a 2.58 percent higher error rate on the test data.

Both these visuals and error rates help to confirm the superiority of the 9 terminal node model.

Now, we can see if other machine-learning methods create a more reliable model. We will start with bagging which entails compiling many trees and taking an average of their classification votes.

3. **Bagging (aggregation of trees)**



```
> bag.OJ

Call:
 randomForest(formula = Purchase ~ ., data = train, mtry = p,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 17

        OOB estimate of  error rate: 20.75%
Confusion matrix:
    CH  MM class.error
CH 400  85   0.1752577
MM  81 234   0.2571429
```

Figure 0.13: Output for Bag Model

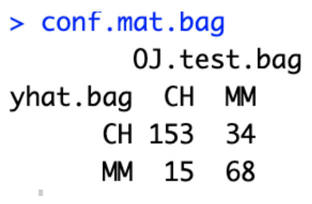
Explanation:
One important thing to note is that bagging considers all 17 variables at each split. This can sometimes cause problems with multicollinearity. Averaging 500 total trees can help with this multicollinearity affect, but I suspect this one will not beat the unpruned tree model. The training error rates are as follows:

CH Class Error Rate (Unpruned) = 0.1752

MM Class Error Rate (Pruned) = 0.2571

Average or OOB Estimate Error Rate = 0.2075

This training rate is higher than both previous training rates. However, the testing misclassification rates hold more importance when it comes to model reliability, so we will take a look at that next in the below confusion matrix.



```
> conf.mat.bag
          OJ.test.bag
yhat.bag   CH   MM
      CH  153   34
      MM   15   68
```

Figure 0.14: Confusion Matrix for Bagging Model

Explanation:
On test data, according to this confusion matrix, the model predicted the response values with the following amount of accuracy:

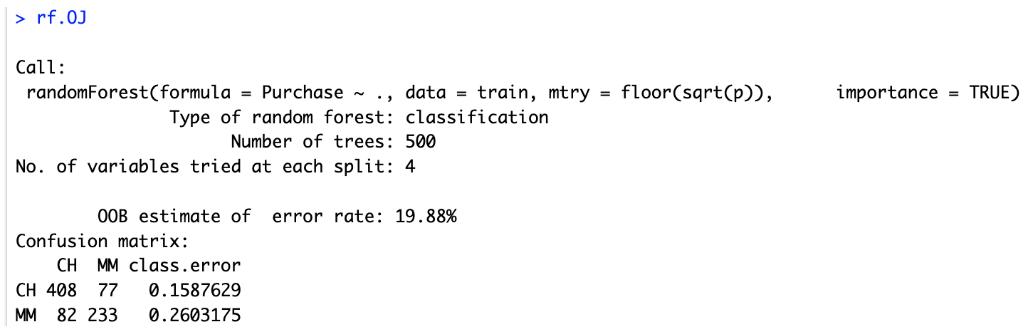
Accuracy Rate = 0.8185

Misclassification Rate = 0.1815

Thought it holds a lower accuracy compared to the unpruned model, the bagging method did perform better than the pruned model.

Now, we will take a look at Random Forests method. This method uses less variables in each decision split. This allows it to better deal with multicollinearity. Therefore, I believe it will perform better than the bagging method. Let's take a look.

4. **Random Forests**
The following summary gives an overview of the random forest model created in R.



```
> rf.OJ

Call:
 randomForest(formula = Purchase ~ ., data = train, mtry = floor(sqrt(p)),      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 19.88%
Confusion matrix:
    CH  MM class.error
CH 408  77   0.1587629
MM  82 233   0.2603175
```

Figure 0.15: Output for RF Model

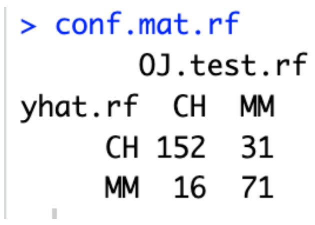
Explanation:
So, the number of variables that was tried at each decision split was 4. This is approximately the $\sqrt{17}$. This is typical for the random forest method when dealing with classification. This is what will allow a greater handle on multicollinearity.

CH Class Error Rate (Unpruned) = 0.1588

MM Class Error Rate (Pruned) = 0.2603

Average or OOB Estimate Error Rate = 0.19.88

As suspected, the random forest method creates a slightly lower training misclassification rate compared to bagging.



```
> conf.mat.rf
        OJ.test.rf
yhat.rf  CH  MM
     CH 152  31
     MM  16  71
```

Figure 0.16: Confusion Matrix for RF Model

Explanation:
On test data, according to the confusion matrix above, the model predicted the response values with the following amount of accuracy:

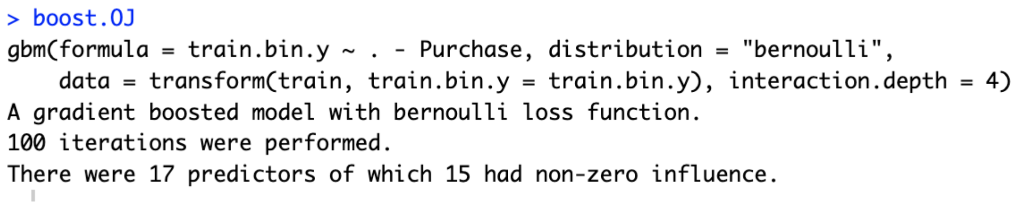
Accuracy Rate = 0.8259

Misclassification Rate = 0.1741

This is the second best model yet, just slightly underneath the unpruned accuracy rate. As I suspected, this method was superior to the bagging method.

Now, we will look at a more unique method called Boosting.

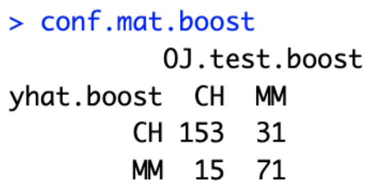
5. **Boosting Tree Method**
This method creates a multitude of smaller treees, each trying to correct the mistakes of the previous tree. The following is the summary output for the boosting model.



```
> boost.OJ
gbm(formula = train.bin.y ~ . - Purchase, distribution = "bernoulli",
    data = transform(train, train.bin.y = train.bin.y), interaction.depth = 4)
A gradient boosted model with bernoulli loss function.
100 iterations were performed.
There were 17 predictors of which 15 had non-zero influence.
```

Figure 0.17: Output for Boost Model

Explanation:
This model filtered out 2 of the 17 models, leaving 15 that were used in its construction.



```
> conf.mat.boost
          OJ.test.boost
yhat.boost  CH  MM
        CH 153  31
        MM  15  71
```

Figure 0.18: Confusion Matrix for RF Model

Explanation:
On test data, according to the above confusion matrix, the model predicted the response values with the following amount of accuracy:

Accuracy Rate = 0.8296

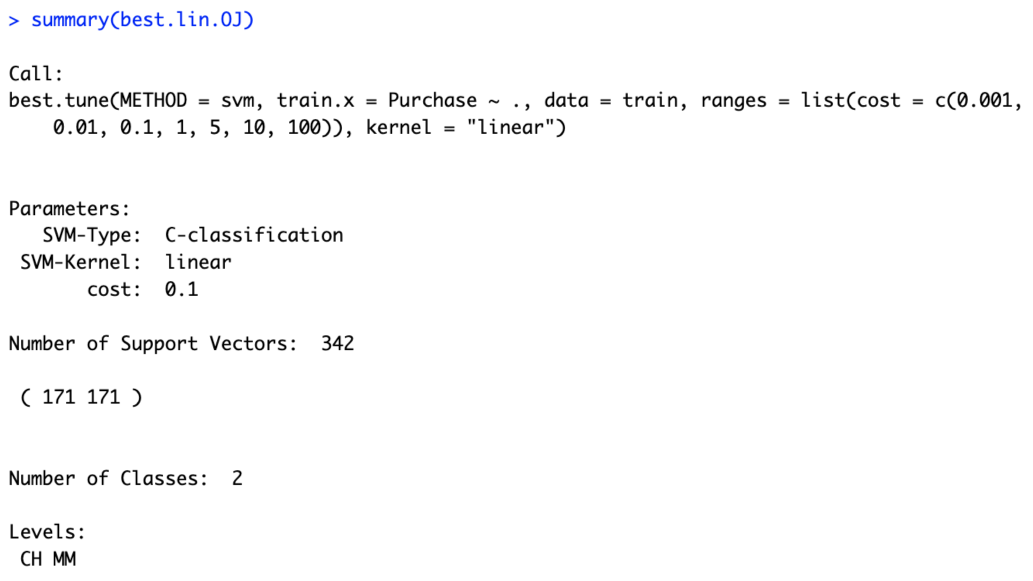
Misclassification Rate = 0.1704

This model is now tied with the unpruned decision tree for level of accuracy. It outperformed both bagging and random forests methods.

Moving away from decision tree models, we are also going to build some models based on Support Vectors Machines.

6. **Support Vetor Machines (SVMs) - Linear**
   SVMs have the goal to provide a hyperplane that can split the respnse space into nicely seperated classification areas. This allows the model to predict further data based on past support vectors. SVMs can be used linearly or non-linearly. It depends entirely on what the data will work better with.

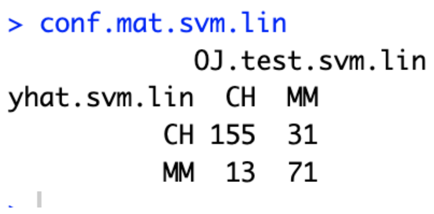
   The following is the summary output for the Linear SVM Model:



```
> summary(best.lin.OJ)

Call:
best.tune(METHOD = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.001,
    0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  342

 ( 171 171 )


Number of Classes:  2

Levels:
 CH MM
```

Figure 0.19: Output for SVM Linear Model

Explanation:
The cost associated with the SVM is 0.1. This is the penalty associated with mistakes on the training data. It helps shape how wide or narrow the margin will be. There are shown to be 342 support vectors, or points that lie on or within the margin. These also include the misclassified data points.



```
> conf.mat.svm.lin
              OJ.test.svm.lin
yhat.svm.lin   CH   MM
          CH  155   31
          MM   13   71
```

Figure 0.20: Confusion Matrix for SVM Linear Model

Explanation:
On test data, according to the matrix above, the model predicted the response values with the following amount of accuracy:

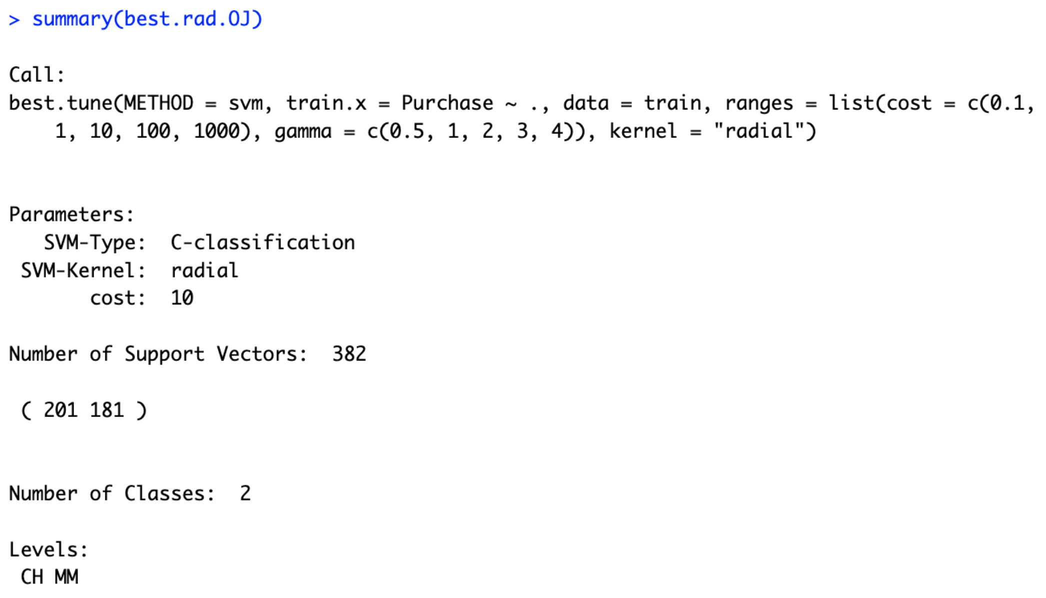
$$\text{Accuracy Rate} = 0.8370$$

$$\text{Misclassification Rate} = 0.1630$$

This outperforms our very first method by 0.74 percent.

Now we will see if a non-linear SVM can do as well. My assumption is no because if it works well with linear it is most likely going to perform well in a non-linear model.

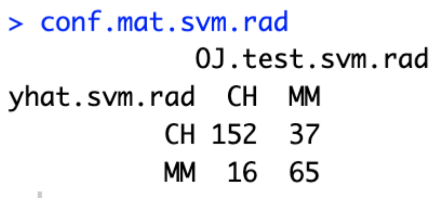
7. **Support Vetor Machines (SVMs) - Non-linear**
Non-linear SVMs have two important factors, the cost and the gamma. We have discussed cost but not gamma. With non-linearity there is a curves and bumps and wiggles. Gamma determines how complex (snug to the data) these bumps are allowed to be.



```
> summary(best.rad.OJ)

Call:
best.tune(METHOD = svm, train.x = Purchase ~ ., data = train, ranges = list(cost = c(0.1, 
    1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification 
 SVM-Kernel:  radial 
       cost:  10 

Number of Support Vectors:  382

 ( 201 181 )


Number of Classes:  2 

Levels: 
 CH MM
```

Figure 0.21: Output for SVM Non-linear Model

Explanation:
The cost associated with this non-linear SVM is 10. The number of support vectors is 382. Without looking at the accuracy rate, it is already implied that this model underperforms compared to its linear counterpart. The more support vectors, the more likely there is less clear demarcation between classifications.



```
> conf.mat.svm.rad
             OJ.test.svm.rad
yhat.svm.rad  CH  MM
          CH 152  37
          MM  16  65
```

Figure 0.22: Confusion Matrix for SVM Non-linear Model

Explanation:
On test data, based on the matrix above, the model predicted the response values with the following amount of accuracy:

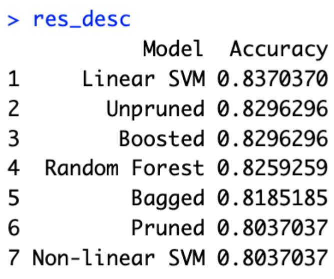
Accuracy Rate = 0.8037

Misclassification Rate = 0.1963

As suspected this underperforms compared to most other models.

Now that we have seen all of the models, as well as their accuracy and misclassification rates, let's conclude which model has the greatest prediction power.

8. **Overall Model Comparisons**



```
> res_desc
           Model  Accuracy
1     Linear SVM 0.8370370
2       Unpruned 0.8296296
3        Boosted 0.8296296
4  Random Forest 0.8259259
5         Bagged 0.8185185
6         Pruned 0.8037037
7 Non-linear SVM 0.8037037
```

Figure 0.23: Model Accuracy Comparisons

Final Conclusion:
Based on these accuracy rates, we have a clear winner on prediction power: Linear SVM. Among all models, the Linear SVM achieved the highest test accuracy (0.837), edging out the tree-based methods (unpruned: 0.830; RF/boosting around 0.826–0.830). Based on this, we can infer that the pattern dividing CH and MM is almost a straight-line rule.

APPENDIX
R code:

```
#### HW 5 ####
#### 1. OJ Data and Package Installation ####
library(tree)
library(ISLR)
data(OJ)
View(OJ)
?OJ
summary(OJ)
str(OJ)

#### 2. Producing Training and Test Sets ####
set.seed (1)
sample.index <- sample (1:nrow(OJ), 800)
train <- OJ[sample.index,]
test <- OJ[-sample.index, ]

#### 3 and 4. Produce a tree output/summary ####
tree.OJ <- tree(Purchase~., data = train)
tree.OJ
summary(tree.OJ)
##Training Data Rates:
##Misclass: 0.1588

#### 5. Plot the Tree ####
plot(tree.OJ)
text(tree.OJ ,pretty =0)

#### 6. Predict Response and Create Confusion Matrix ####
yhat <- predict (tree.OJ ,newdata = test, type = "class")
OJ.test <- test$Purchase

conf.mat.tree <- table(yhat, OJ.test)
conf.mat.tree
mosaicplot(conf.mat.tree, shade = TRUE, main = "Confusion␣Matrix␣(Unpruned␣Tree)")
accuracy.tree <- mean(yhat == OJ.test)
accuracy.tree
misclassification.rate.tree <- 1 - accuracy.tree
misclassification.rate.tree
## Test Data Rates:
##Accuracy: 0.8296
```

```
##Misclass: 0.1704

#### 7, 8 and 9. Producing a Plot and Finding Optimal Tree Size ####
set.seed(2)
cv.OJ <- cv.tree(tree.OJ, FUN = prune.misclass, K = 10)
cv.OJ
plot(cv.OJ$size ,cv.OJ$dev ,type="b",
main = "Cross-Validation␣to␣Find␣Optimal␣#␣of␣Nodes")

best.tree.index <- which.min(cv.OJ$dev)
best.tree.size <- cv.OJ$size[best.tree.index]
print(paste("The␣optimal␣tree␣size␣has", best.tree.size, "terminal␣nodes."))
##The optimal size is the same as the unpruned tree, so we will
# be using 5 terminal nodes instead.

#### 10. Pruned Tree Model ####
prune.OJ <- prune.tree(tree.OJ ,best = 5)
plot(prune.OJ)
text(prune.OJ ,pretty =0)
summary(prune.OJ)
##Training Data Rates:
##Misclass: 0.205


#### 11 ####
#The training misclassification error rate for the pruned tree
# is 4.62% higher than the one for the unpruned tree.

#### 12 ####
#The test misclassification error rate for the pruned tree
#is 2.58% higher than the unpruned tree.

#### Predict Response and Create Confusion Matrix for the Pruned Tree
yhat.prune <- predict (prune.OJ ,newdata = test, type = "class")
OJ.test.prune <- test$Purchase

conf.mat.prune <- table(yhat.prune, OJ.test.prune)
conf.mat.prune
mosaicplot(conf.mat.prune, shade = TRUE, main = "Confusion␣Matrix␣(Pruned␣Tree)")
accuracy.prune <- mean(yhat.prune == OJ.test.prune)
accuracy.prune
misclassification.rate.prune <- 1 - accuracy.prune
misclassification.rate.prune
## Test Data Rates:
```

```
##Accuracy: 0.8037
##Misclass: 0.1962

#### 13 ####
library (randomForest)
?randomForest()

p <- ncol(OJ) - 1
##Bagging:
set.seed(4)
bag.OJ <- randomForest(Purchase~.,data = train, mtry = p, importance =TRUE)
bag.OJ
##Training Data Rates:
##Misclass: 20.75%

yhat.bag <- predict (bag.OJ ,newdata = test, type = "class")
OJ.test.bag <- test$Purchase

conf.mat.bag <- table(yhat.bag, OJ.test.bag)
conf.mat.bag
accuracy.bag <- mean(yhat.bag == OJ.test.bag)
accuracy.bag
misclassification.rate.bag <- 1 - accuracy.bag
misclassification.rate.bag
## Test Data Rates:
##Accuracy: 0.8185
##Misclass: 0.1815


##Random Forest:
set.seed(5)
rf.OJ <- randomForest(Purchase~.,data = train, mtry = floor(sqrt(p)),
importance =TRUE)
rf.OJ
##Training Data Rates:
##Misclass: 19.88%

yhat.rf <- predict (rf.OJ ,newdata = test, type = "class")
OJ.test.rf <- test$Purchase

conf.mat.rf <- table(yhat.rf, OJ.test.rf)
conf.mat.rf
accuracy.rf <- mean(yhat.rf == OJ.test.rf)
accuracy.rf
```

```
misclassification.rate.rf <- 1 - accuracy.rf
misclassification.rate.rf
## Test Data Rates:
##Accuracy: 0.8259
##Misclass: 0.1741


##Boosting Method:
library (gbm)
train.bin.y <- ifelse(train$Purchase == "CH", 1, 0)
test.bin.y  <- ifelse(test$Purchase  == "CH", 1, 0)

class(test.bin.y)

set.seed (6)
boost.OJ <- gbm(train.bin.y~.-Purchase,
            data = transform(train, train.bin.y = train.bin.y),
            distribution = "bernoulli", interaction.depth = 4)
boost.OJ
summary(boost.OJ)

best.n.trees <- gbm.perf(boost.OJ, method = "OOB", plot.it = FALSE)
best.n.trees

phat.boost <- predict(boost.OJ, newdata = transform(test, test.bin.y = test.bin.y),
n.trees = best.n.trees, type = "response")
yhat.boost <- factor(ifelse(phat.boost > 0.5, "CH", "MM"),
levels = levels(test$Purchase))
OJ.test.boost <- test$Purchase

conf.mat.boost <- table(yhat.boost, OJ.test.boost)
conf.mat.boost
accuracy.boost <- mean(yhat.boost == OJ.test.boost)
accuracy.boost
misclassification.rate.boost <- 1 - accuracy.boost
misclassification.rate.boost
## Test Data Rates:
##Accuracy: 0.8296
##Misclass: 0.1704

#### SVM ####
install.packages("e1071")
library(e1071)
is.factor(train$Purchase)
```

```
##Linear SVM:
set.seed(1)
tune.lin.OJ <- tune(
  svm, Purchase ~ ., data = train,
  kernel = "linear",
  ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100))
)
best.lin.OJ <- tune.lin.OJ$best.model
best.lin.OJ
summary(best.lin.OJ)

yhat.svm.lin <- predict(best.lin.OJ, newdata = test)
OJ.test.svm.lin <- test$Purchase

conf.mat.svm.lin <- table(yhat.svm.lin, OJ.test.svm.lin)
conf.mat.svm.lin
accuracy.svm.lin <- mean(yhat.svm.lin == OJ.test.svm.lin)
accuracy.svm.lin
misclassification.rate.svm.lin <- 1 - accuracy.svm.lin
misclassification.rate.svm.lin

##Non-linear SVM
set.seed(1)
tune.rad.OJ <- tune(
  svm, Purchase ~ ., data = train,
  kernel = "radial",
  ranges = list(
    cost  = c(0.1, 1, 10, 100, 1000),
    gamma = c(0.5, 1, 2, 3, 4)
  )
)
best.rad.OJ <- tune.rad.OJ$best.model
best.rad.OJ
summary(best.rad.OJ)

yhat.svm.rad <- predict(best.rad.OJ, newdata = test)
OJ.test.svm.rad <- test$Purchase

conf.mat.svm.rad <- table(yhat.svm.rad, OJ.test.svm.rad)
conf.mat.svm.rad
accuracy.svm.rad <- mean(yhat.svm.rad == OJ.test.svm.rad)
accuracy.svm.rad
misclassification.rate.svm.rad <- 1 - accuracy.svm.rad
misclassification.rate.svm.rad


#### Compare Models Code ####
#Tree Plots:
par(mfrow=c(1,2))
plot(tree.OJ)
text(tree.OJ ,pretty =0)
plot(prune.OJ)
text(prune.OJ ,pretty =0)
#Confusion Matrices:
par(mfrow=c(1,2))
mosaicplot(conf.mat.tree, shade = TRUE, main = "Confusion␣Matrix␣(Unpruned␣Tree)")
mosaicplot(conf.mat.prune, shade = TRUE, main = "Confusion␣Matrix␣(Pruned␣Tree)")

#Results Table:
res <- data.frame(
  Model = c("Unpruned", "Pruned", "Bagged", "Random␣Forest",
  "Boosted", "Linear␣SVM", "Non-linear␣SVM"),
  Accuracy = c(accuracy.tree,
               accuracy.prune,
               accuracy.bag,
               accuracy.rf,
               accuracy.boost,
               accuracy.svm.lin,
               accuracy.svm.rad)
)

res_desc <- res[order(res$Accuracy, decreasing = TRUE), ]
row.names(res_desc) <- NULL  # tidy row numbers
res_desc

#Code Check for Unpruned vs Boosted and Pruned vs Non-linear SVM:
identical(yhat, yhat.boost)#FALSE
sum(yhat != yhat.boost)#16 predictions are different

identical(yhat.prune, yhat.svm.rad)    #FALSE
sum(yhat.prune != yhat.svm.rad)    #50 predictions are different

#Because these pairs of models had the same accuracy, I wanted to make
#sure there wasn't an error in my coding.
```