The Final Project

Antonio Gonzalez

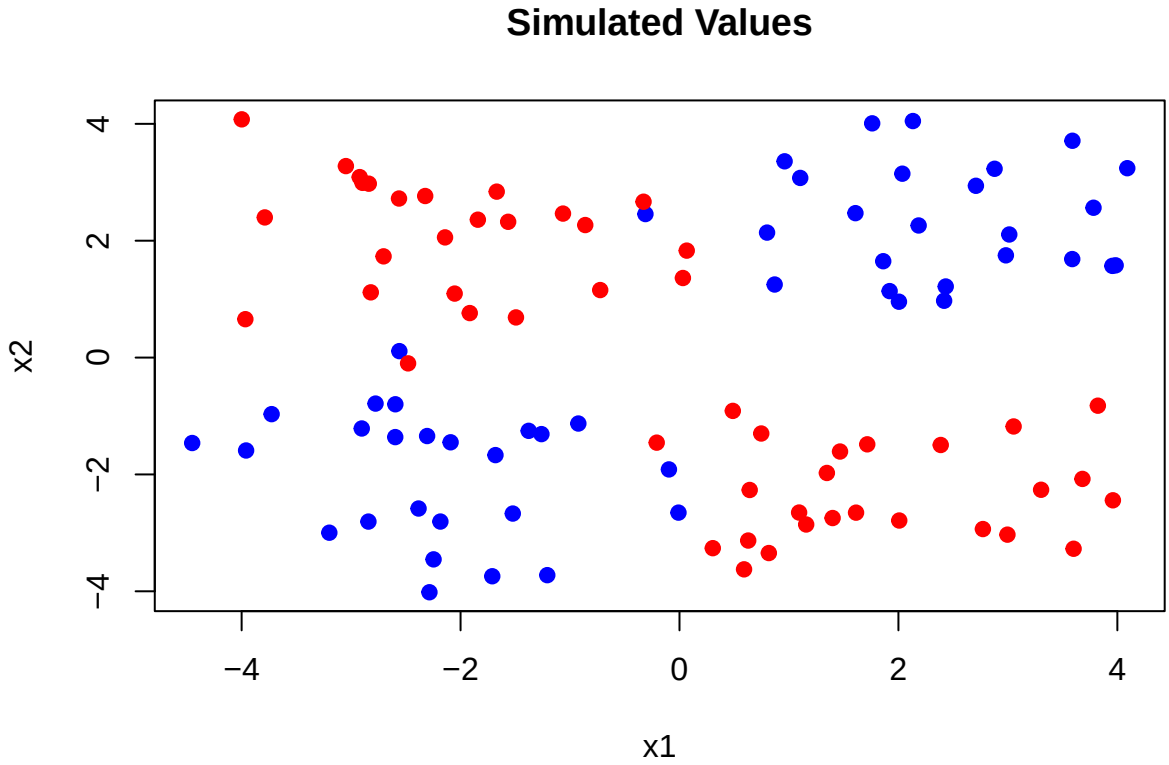
04/27/2025

Problem #1 (45 points)

Solve Problem **9.7.4** from the textbook (page 399).

I am first going to generate data points that appear to be non-linearly separable.

```r
set.seed(2)

x <- matrix(rnorm(100 * 2), ncol = 2)

x[1:25, ] <- x[1:25, ] + 2
x[26:50, ] <- x[26:50, ] - 2

x[51:75, 1] <- x[51:75, 1] - 2
x[51:75, 2] <- x[51:75, 2] + 2

x[76:100, 1] <- x[76:100, 1] + 2
x[76:100, 2] <- x[76:100, 2] - 2

y <- c(rep(1, 50), rep(-1, 50))

# Create dataframe
dat <- data.frame(x, y = as.factor(y))

# Plot
plot(x, col = ifelse(y == 1, "blue", "red"), pch = 19,
     main = "Simulated Values",
     xlab = "x1", ylab = "x2")
```



The plot above shows the red class in quadrants II and IV and the blue class in quadrant I and III, so obviously they are not linearly separable. Let's get our training data indices.

```
#svm, but we first we encode
set.seed(2)
mydata <- data.frame(x = x, y = as.factor(y))
svmtrain = sample(100, 80)
```

We will now use cross-validation to choose the best cost for both models and the best degree value for the polynomial kernel.

```
library(e1071)
## Warning: package 'e1071' was built under R version 4.3.3
set.seed(2)

classifier.tune.out <- tune(svm, y ~ ., data = mydata, kernel = "linear",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))

poly.tune.out <- tune(svm, y ~ ., data = mydata, kernel = "polynomial",
    ranges = list(degree = c(2, 3, 4, 5),
                  cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
```

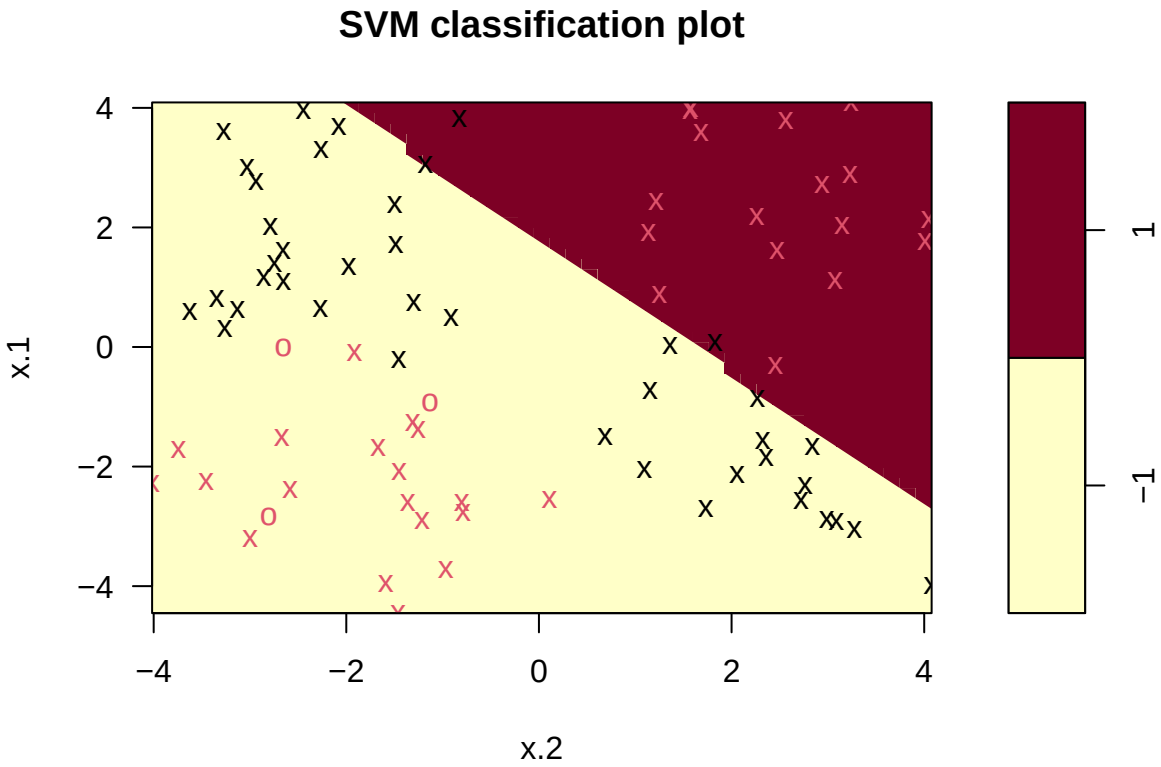
Let's store the best models.

```
bestclassmod <- classifier.tune.out$best.model
bestpolymod <- poly.tune.out$best.model

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

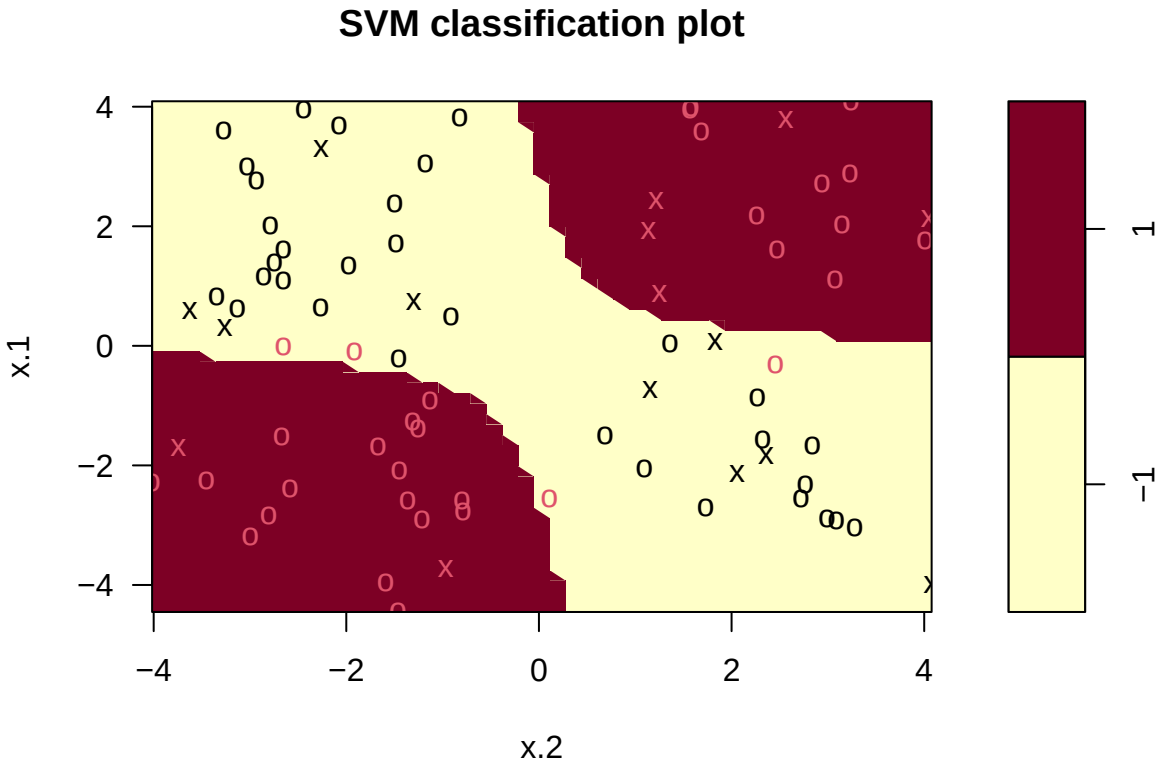
```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  5
##
## Number of Support Vectors:  96
##
##  ( 48 48 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
summary(bestpolymod)
##
## Call:
## best.tune(METHOD = svm, train.x = y ~ ., data = mydata, ranges = list(degree = c(2,
##     3, 4, 5), cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  5
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  19
##
##  ( 9 10 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Now we will plot the SVMs to see how well the decision boundary classifies each point.

```
plot(bestclassmod, mydata[svmtrain,], main = 'SVC plot')
```




```
plot(bestpolymod, mydata[svmtrain,], main = 'SVM plot')
```



For the support vector classifier (SVC) with a linear kernel, we observe several misclassifications, particularly in the bottom-right quadrant. In contrast, the support vector machine (SVM) with a polynomial kernel performs noticeably better, misclassifying fewer points overall. It's also worth noting that the SVM with the polynomial kernel uses more support vectors, which suggests a more flexible decision boundary that better

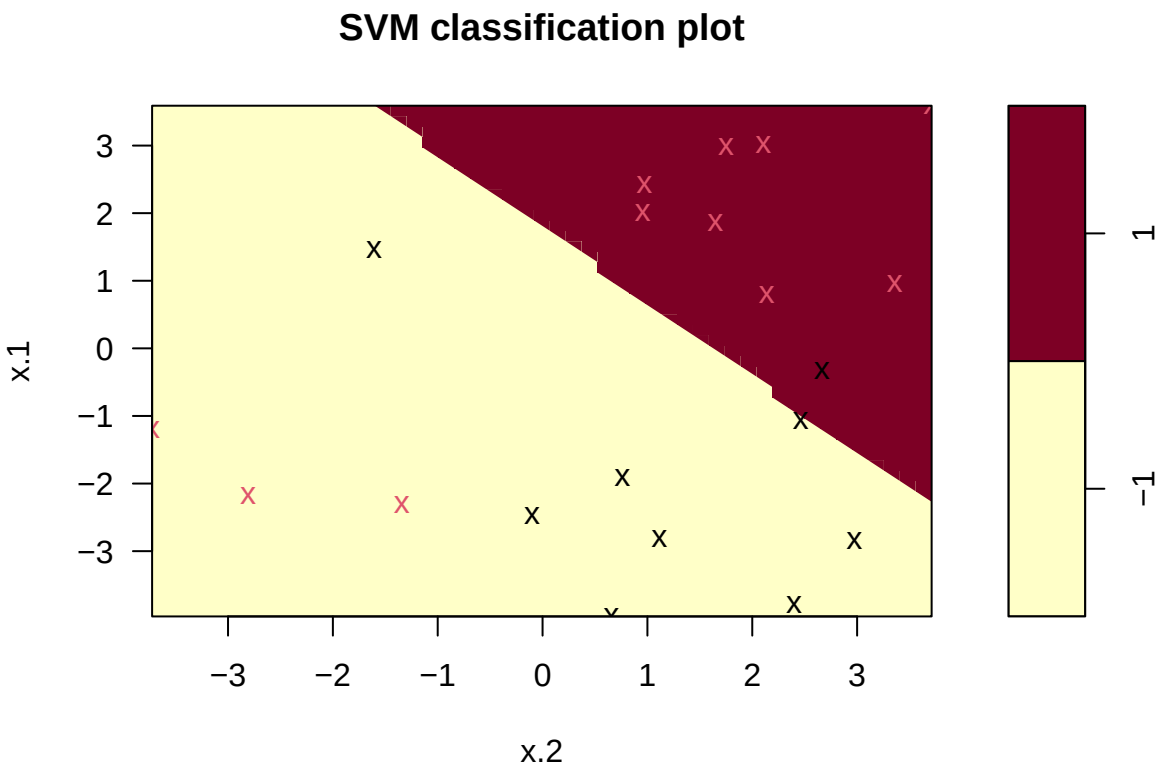
captures the non-linear structure in the data.

```
ypred_class = predict(bestclassmod, mydata[svmtrain, c(1, 2)])
tab1 <- table(Predicted = ypred_class, Actual = mydata[svmtrain, "y"])
1 - (sum(diag(tab1))/sum(tab1))
## [1] 0.3125
ypred_poly = predict(bestpolymod, mydata[svmtrain, c(1, 2)])
tab2 <- table(Predicted = ypred_poly, Actual = mydata[svmtrain, "y"])
1 - (sum(diag(tab2))/sum(tab2))
## [1] 0.05
```

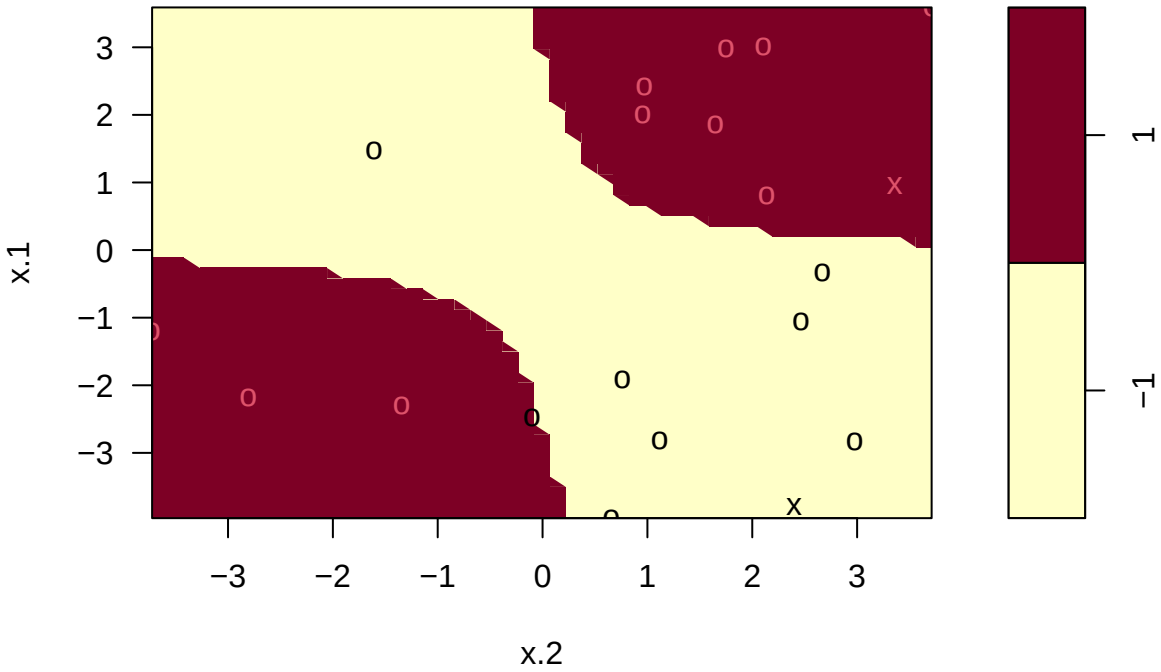
Based on the misclassification errors, the SVM with the polynomial kernel performs best on the training data, achieving an error rate of 5, compared to the support vector classifier (SVC) with a linear kernel, which has a higher error rate of 31.25.

```
ypred_classtest = predict(bestclassmod, mydata[-svmtrain, c(1, 2)])
tab1 <- table(Predicted = ypred_classtest, Actual = mydata[-svmtrain, "y"])
1 - (sum(diag(tab1))/sum(tab1))
## [1] 0.2
ypred_polytest = predict(bestpolymod, mydata[-svmtrain, c(1, 2)])
tab2 <- table(Predicted = ypred_polytest, Actual = mydata[-svmtrain, "y"])
1 - (sum(diag(tab2))/sum(tab2))
## [1] 0.05
```

```
plot(bestclassmod, mydata[-svmtrain,])
```




```
plot(bestpolymod, mydata[-svmtrain,])
```



The performance of the SVMs on the testing data closely resembles the results from the training data. The plots reveal that the polynomial kernel SVM correctly classifies nearly all test points, with only one point lying exactly on the decision boundary. In contrast, the linear SVC visibly misclassifies two points. Additionally, the linear SVC appears to classify nearly all points as support vectors, resulting in a very wide margin.

Numerically, the SVC has a misclassification error of 20, while the polynomial kernel SVM achieves a lower error of just 5. These results indicate that the polynomial SVM generalizes well and outperforms the linear SVC on both training and testing data, suggesting that it is neither underfitting nor overfitting the data.

Problem #2 ($5 \times 11 = 55$ points)

Solve Problem **8.4.9** from the textbook (pages 363-364). ## part a

```
library(ISLR2)
## Warning: package 'ISLR2' was built under R version 4.3.3
library(tree)
## Warning: package 'tree' was built under R version 4.3.3
data(OJ) #load data

set.seed(1)
train.ind <- sample(1:nrow(OJ), 800)
train <- OJ[train.ind,]
test <- OJ[-train.ind,]
```

part b

Now we fit the tree.

```
tree.oj <- tree(Purchase ~ . , data = train)
summary(tree.oj)
##
```

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

The tree has nine terminal nodes and a misclassification error of 0.1588

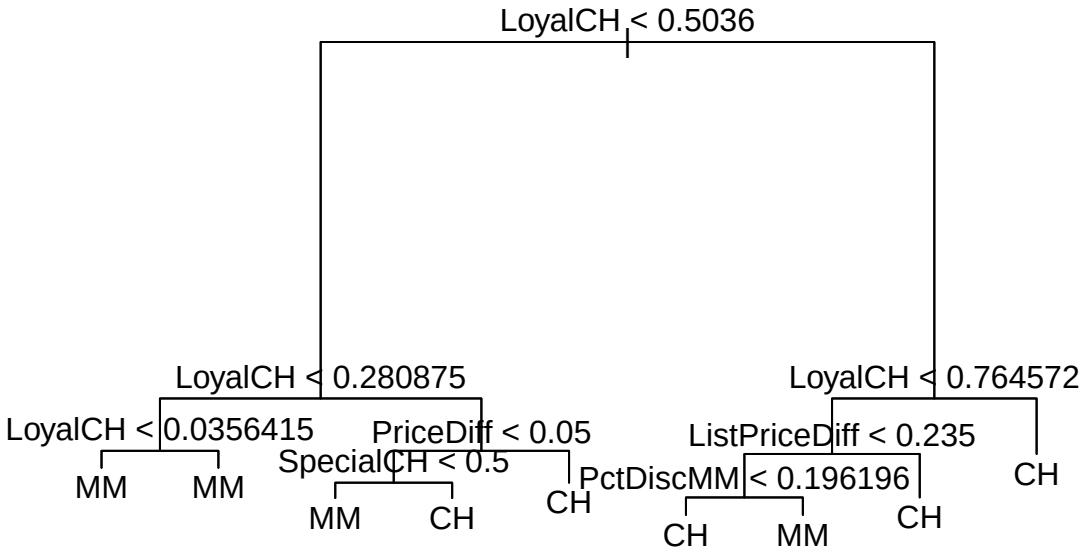
part c

```
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 ) *
```

Node 8, a terminal node, comes from split LoyalCH $<$ 0.280875, 59 observations ended up at this node, there's a deviance of 10.14, leads to the prediction of Minute Maid (MM) being purchased and the probability of MM being purchased is 0.98305 and CH is 0.01695.

part d

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



Based on the tree, the most important indicator in determining the purchase of Minute Maid or Citrus Hill seems to be customer brand loyalty for Citrus Hill. Since `LoyalCH` appears a lot it may seems we might have to prune this tree. It seems the other important indicators are `PriceDiff`, `ListPriceDiff` , `PctDiscMM`, and `SpecialCH`.

##part e

```
tree.pred <- predict(tree.oj, newdata = test, type = 'class')
confusion <- table(tree.pred, test$Purchase)
confusion
##
## tree.pred  CH  MM
##        CH 160  38
##        MM   8  64
test.error <- 1 - sum(diag(confusion))/sum(confusion)
test.error
## [1] 0.1703704
```

After producing our confusion matrix, it seems our testing error rate is 17.03%.

##part f

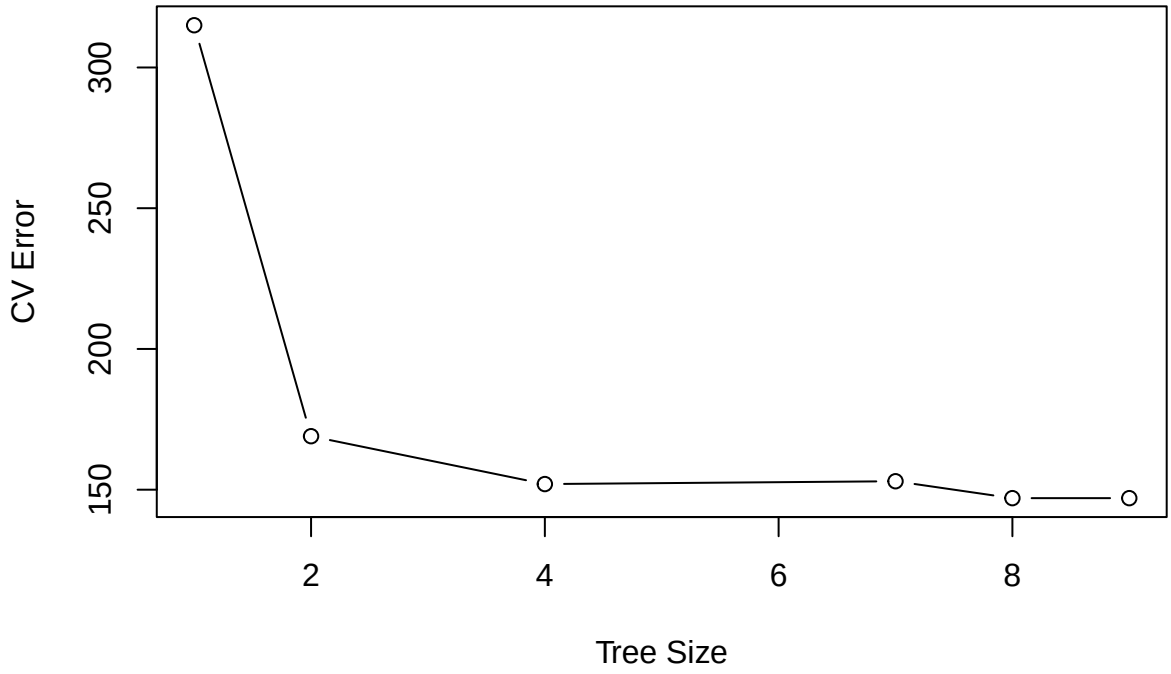
```
set.seed(2)
cv.oj <- cv.tree(tree.oj, FUN = prune.misclass)
cv.oj
## $size
## [1] 9 8 7 4 2 1
##
## $dev
## [1] 147 147 153 152 169 315
##
## $k
## [1]       -Inf   0.000000   3.000000   4.333333  10.500000 151.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
```

```
## [1] "prune"         "tree.sequence"
```

It seems that size and 8 and 9 have the lowest deviance so maybe we don't have to prune this tree. Let's check this elbow plot.

part g

```
plot(cv.oj$size, cv.oj$dev, type = "b", xlab = "Tree Size", ylab = "CV Error")
```

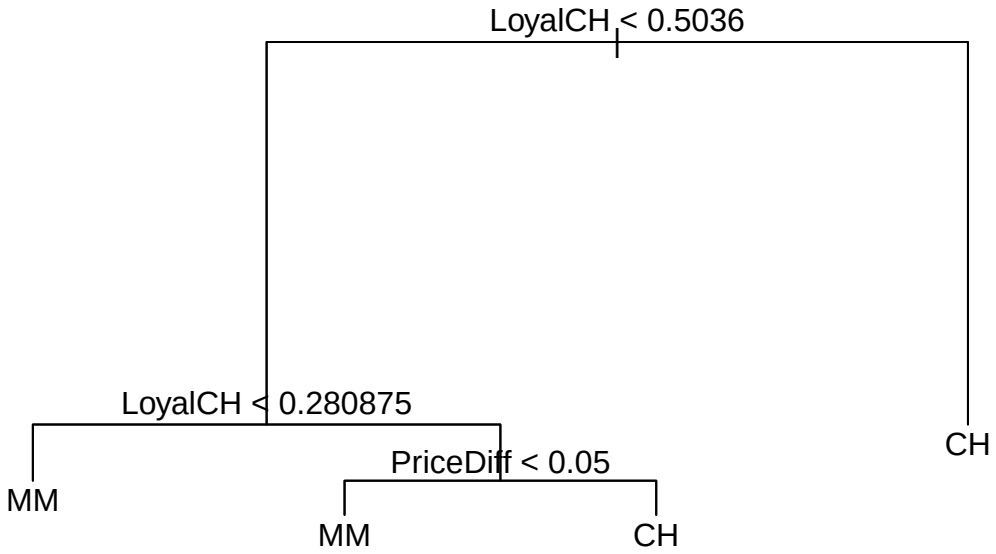


part h

According to the plot, it seems the optimal tree size is 4.

#part i

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




```
summary(prune.oj)
##
## Classification tree:
## snip.tree(tree = tree.oj, nodes = c(4L, 10L, 3L))
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  4
## Residual mean deviance:  0.8922 = 710.2 / 796
## Misclassification error rate: 0.1788 = 143 / 800
```

part j

```
train.pred.full <- predict(tree.oj, train, type = "class")
train.error.full <- mean(train.pred.full != train$Purchase)

train.pred.pruned <- predict(prune.oj, train, type = "class")
train.error.pruned <- mean(train.pred.pruned != train$Purchase)

train.error.full
## [1] 0.15875
train.error.pruned
## [1] 0.17875
```

The training error for the pruned tree ended up being higher, which suggests that maybe pruning wasn't necessary or we might've overpruned. I based the pruning decision on the elbow plot, but now I see that a tree size of eight or nine actually gave the lowest cross-validated error. So maybe, pruning to size four may have been too aggressive, error-wise.

##part k

```
test.pred.full <- predict(tree.oj, test, type = "class")
test.error.full <- mean(test.pred.full != test$Purchase)

test.pred.pruned <- predict(prune.oj, test, type = "class")
test.error.pruned <- mean(test.pred.pruned != test$Purchase)

test.error.full
```

```
## [1] 0.1703704
test.error.pruned
## [1] 0.1777778
```

The test error actually went up for the unpruned tree, from 0.15875 to 0.17037, while the pruned tree had a slightly higher test error of 0.17778. So even though the pruned tree didn't beat the unpruned one on test accuracy, the difference isn't huge. It looks like pruning helped reduce overfitting a bit, and both trees ended up with pretty similar performance overall.

Plus, pruning makes the tree simpler and easier to interpret, which is helpful when you want to understand or explain the model's decisions.