The Final Project

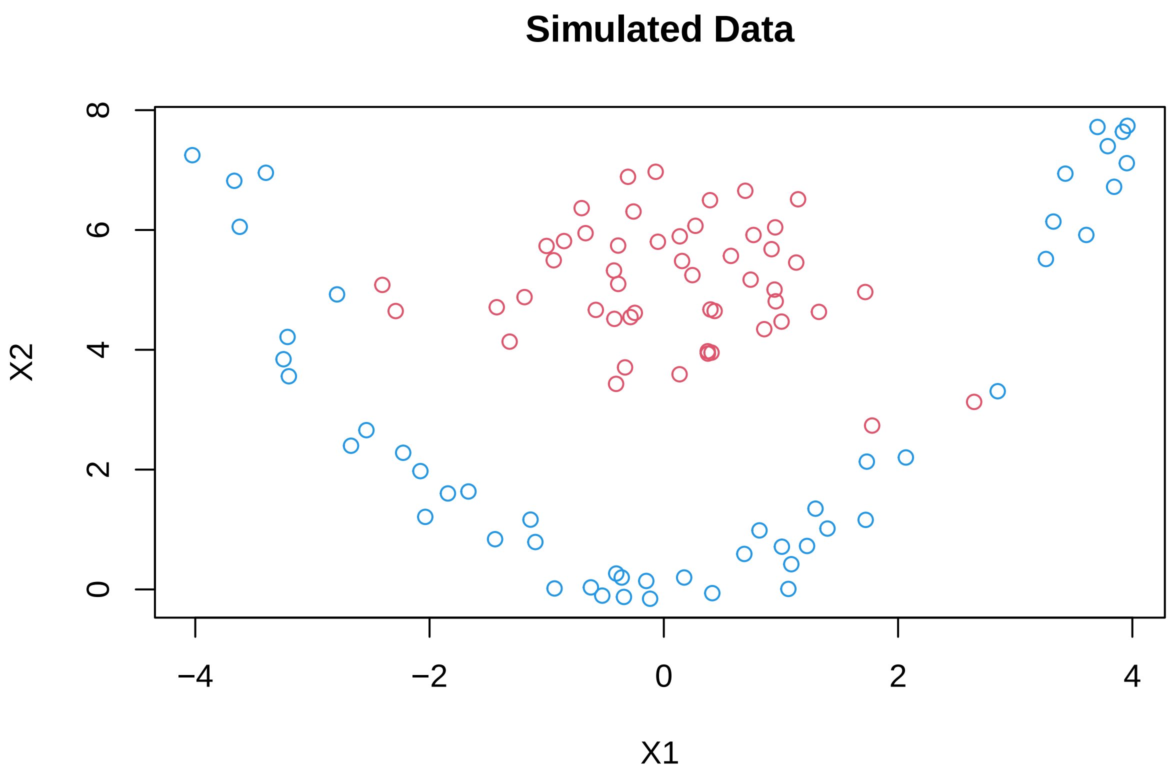
Winter Nguyen (htn937)

2026-01-27

Problem #1 (45 points)

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

```r
set.seed(1)
n=100 # number of observations

# Create simulated data
M1=matrix(NA, n/2, 2)

for (i in 1:n/2) {
  x1= runif(1,-4,4)
  x2=(1/2)*x1^2
  M1[i,1]=x1 + sample(c(-1,1),1)*runif(1,-0.5,0.5)
  M1[i,2]=x2+sample(c(-1,1),1)*runif(1,-0.2,0.2)
}
M2=matrix(rnorm(n),ncol=2)
M=rbind(M1,M2)

# Mark the first 50 is -1 and the last 50 is +1
y= c(rep(-1,n/2), rep(1,n/2))
M[y==1,2 ] = M[y==1,2] +5
plot(M, col=(3-y),
     xlab="X1",
     ylab="X2",
     main="Simulated Data")
```

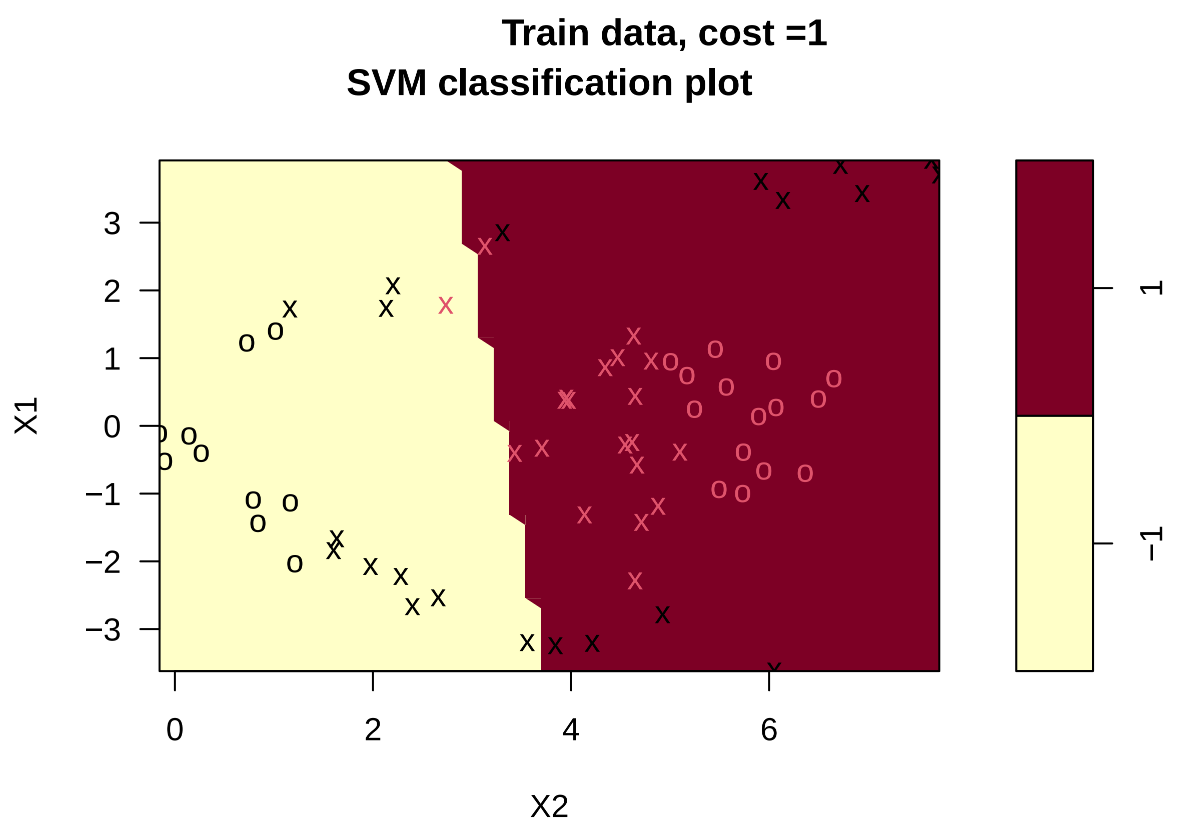


```r
# Split data into training set and test set
set.seed(100)
portion=2/3
train.index=sample(1:n,n*portion) # randomly select "portion" of obs to be training set
train=data.frame(M[train.index, ], y=factor(y[train.index]))
test=data.frame(M[-train.index,], y=factor(y[-train.index]))
```

I tuned each model and then trained it using the best cost value and, when applicable, the best degree or gamma. The models are linear SVM, polynomial SVM and radial SVM.

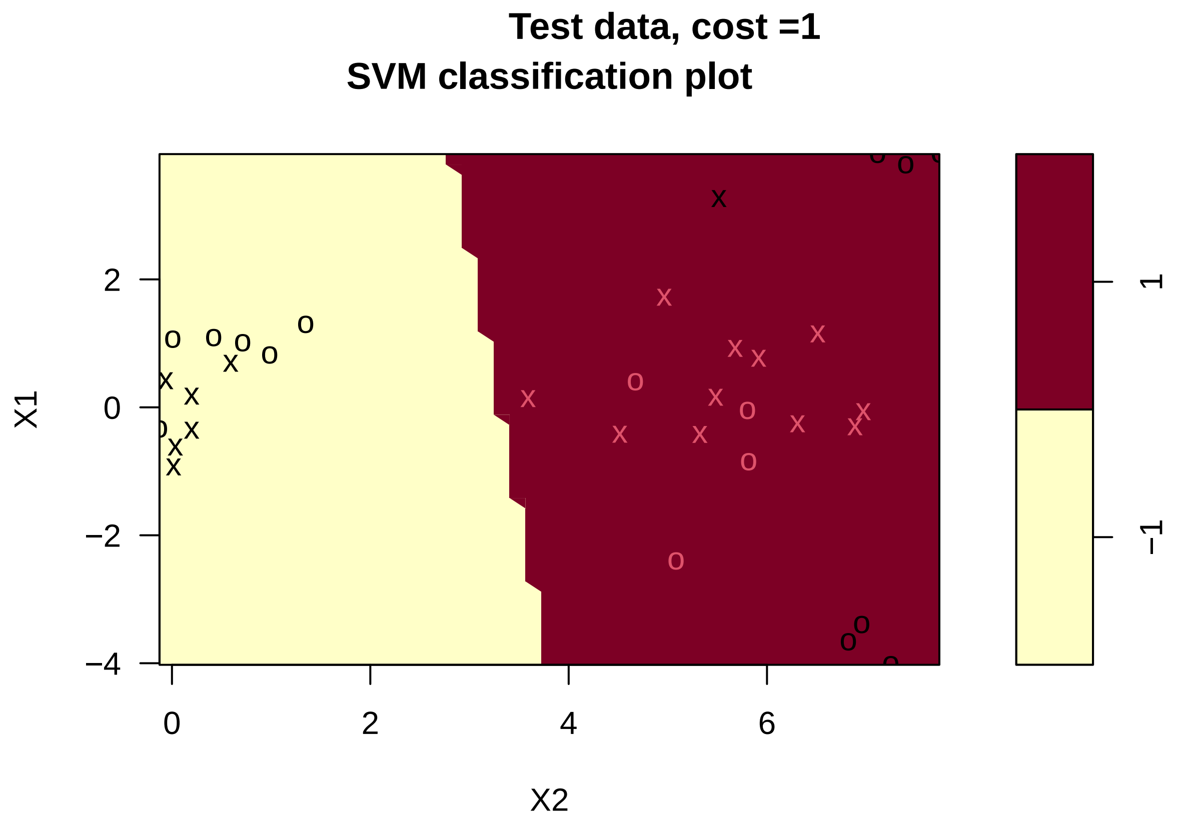
```r
# Support Vector Classifier (linear SVM)
data = data.frame(M, y = as.factor(y))
library(e1071)
svcfit.tune=tune(svm, y ~ ., data = data, kernel = "linear",
  ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100))) # find the best cost

svm_linear.tune = svm(y~.,
                 data=train, kernel ="linear",
                 cost = svcfit.tune$best.parameters$cost,
                 scale = FALSE)

err_rate= function(model, data) {
  mean(predict(model,data) != data$y)
}
```

```r
train_linear = err_rate(svm_linear.tune, train) # training error rate
test_linear=err_rate(svm_linear.tune, test) # test error rate

plot(svm_linear.tune, train) # plot linear SVM on the training set
title(main= paste0("Train data, ", "cost =",svcfit.tune$best.parameters$cost), line =3)
```



```r
summary(svm_linear.tune)
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "linear", cost = svcfit.tune$best.parameters$cost,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  41
##
##  ( 20 21 )
##
##
## Number of Classes:  2
##
```

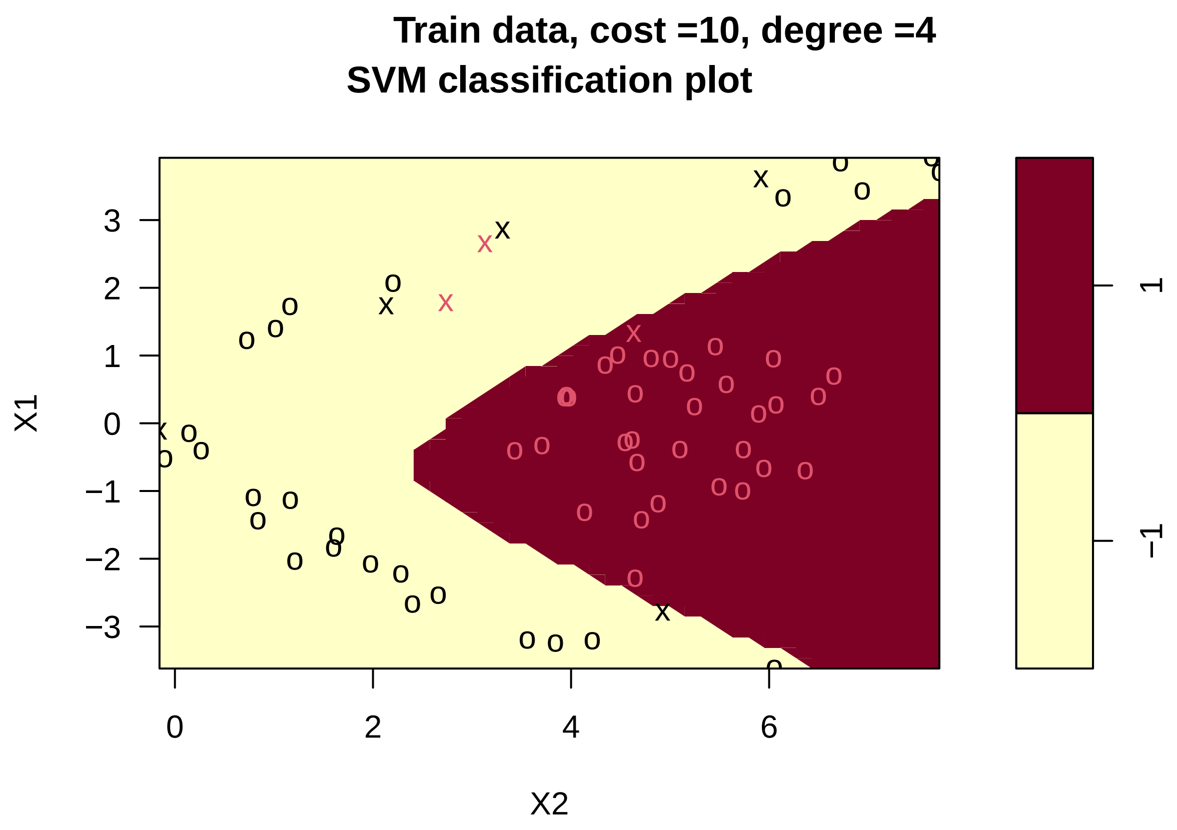
```
## Levels:
##  -1 1
plot(svm_linear.tune, test) # plot linear SVM on the test set
title(main= paste0("Test data, ","cost =",svcfit.tune$best.parameters$cost), line =3)
```



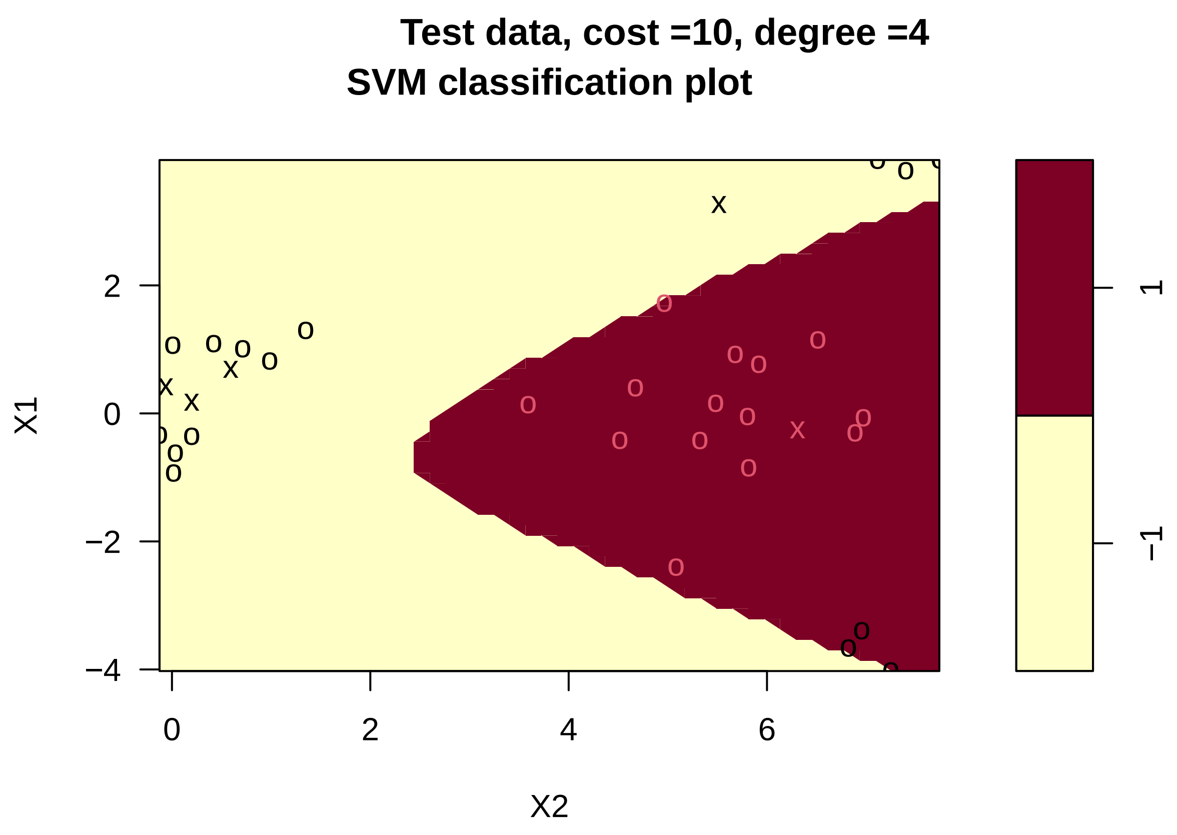
```
# Training and test error rates for the linear SVM
error_matrix=matrix(c(train_linear, test_linear), nrow=2, byrow =FALSE)
colnames(error_matrix) = "Linear"
rownames(error_matrix) = c("Train", "Test")
error_matrix
##          Linear
## Train 0.1818182
## Test  0.2058824
```

The linear SVM was trained with a cost parameter c = 1 selected by cross-validation. The model uses 41 support vectors (20 from class -1 and 21 from class 1), which means a large portion of the training observations lie close to the decision boundary. Together with the relatively high training error (18.18%) and test error (20.59%), this indicates that a straight-line boundary is not flexible enough to separate the two classes in this dataset, whose structure is clearly nonlinear.

```
# Polynomial SVM
poly.tune=tune(svm, y ~ ., data = data, kernel = "polynomial",
  ranges = list(cost = c(0.001, 0.01, 0.1, 1,5, 10),
              degree=c(1,2,3,4,5,6))) # find the best cost
svm_poly.tune=svm(y~., data=train, kernel ="polynomial",
                     degree=poly.tune$best.parameters$degree,
                     cost = poly.tune$best.parameters$cost, scale = FALSE)

train_poly = err_rate(svm_poly.tune, train) # training error rate
test_poly=err_rate(svm_poly.tune, test) # test error rate

plot(svm_poly.tune, train) # plot polynomial SVM on the training set
title(main=paste0("Train data, ", "cost =",poly.tune$best.parameters$cost,
                     ", degree =",poly.tune$best.parameters$degree), line =3)
```



```
plot(svm_poly.tune, test) # plot polynomial SVM on the test set
title(main=paste0("Test data, ","cost =",poly.tune$best.parameters$cost,
                     ", degree =",poly.tune$best.parameters$degree), line =3)
```

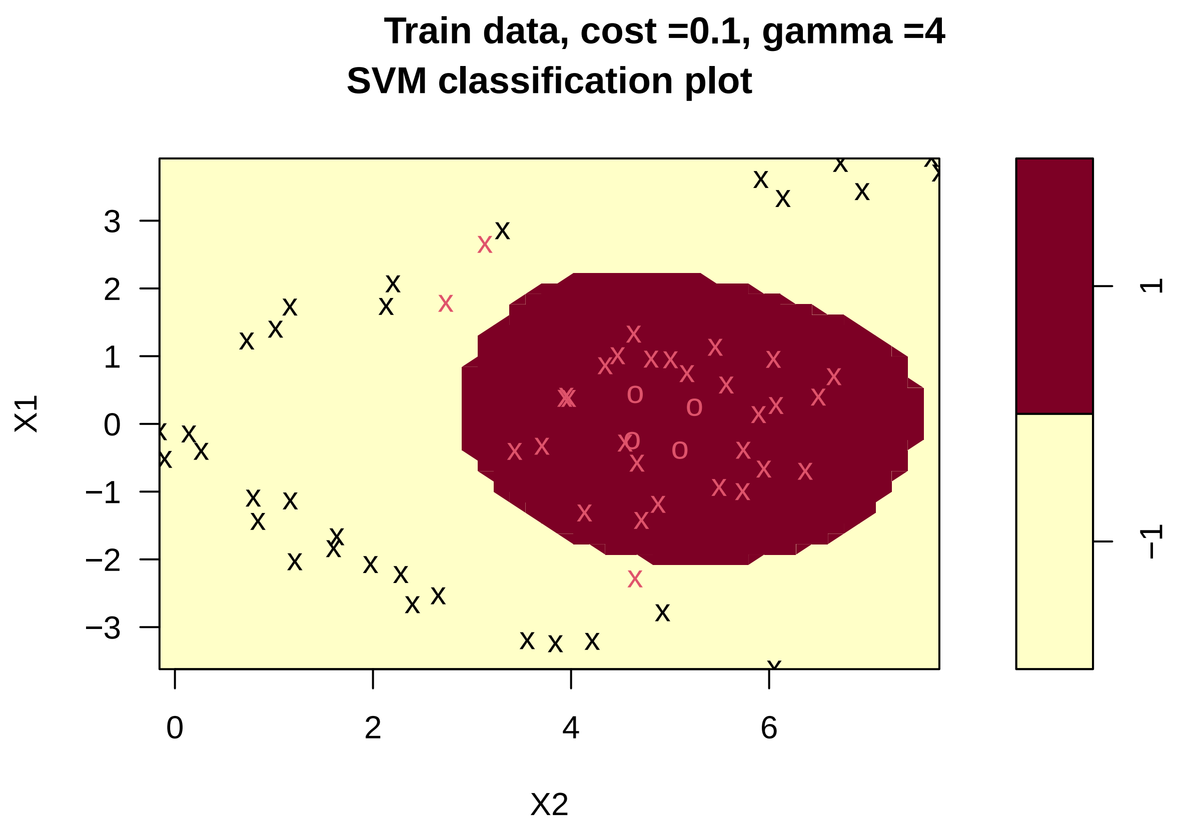


```
summary(svm_poly.tune)
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "polynomial", degree = poly.tune$best.parameters$degree,
##     cost = poly.tune$best.parameters$cost, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  4
##      coef.0:  0
##
## Number of Support Vectors:  8
##
##  ( 3 5 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
# Training and test error rates for the polynomial SVM
error_matrix = cbind(error_matrix,
  Polynomial = c(train_poly, test_poly))
```

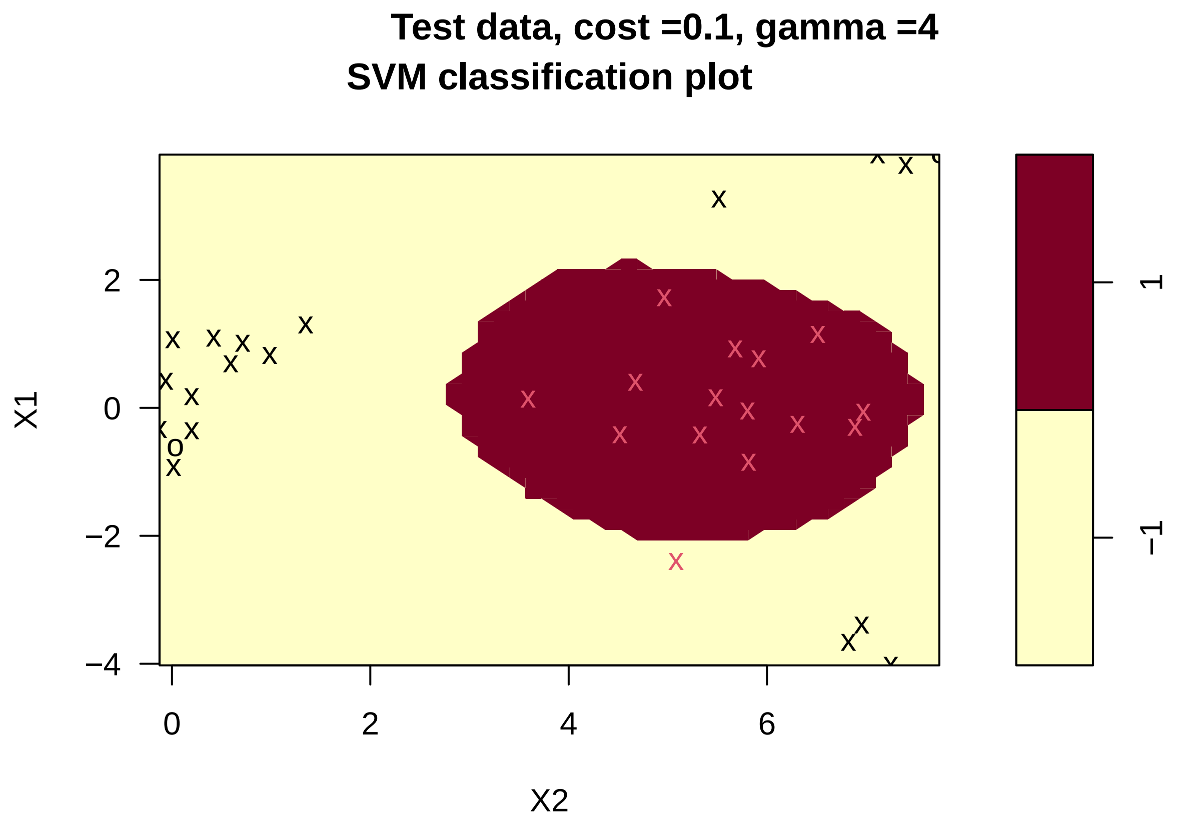
```
error_matrix
##          Linear Polynomial
## Train 0.1818182 0.03030303
## Test  0.2058824 0.08823529
```

The tuned polynomial SVM uses a degree-4 polynomial kernel with cost c = 10. The model relies on only 8 support vectors (3 from class -1 and 5 from class 1), in contrast to the 41 support vectors used by the linear SVM. This indicates that, in the polynomial feature space, the classes are almost perfectly separable with a relatively wide margin, which is consistent with the very low training error (3.03%).

```
# Radial SVM
radial.tune=tune(svm, y ~ ., data = data, kernel = "radial",
  ranges = list(
    cost = c(0.001,0.01, 0.1, 1, 5, 10),
    gamma = c(0.001,0.01,0.5, 1, 2, 3, 4, 5))) # find the best cost
svm_radial.tune= svm(y~., data=train, kernel="radial",
                     cost=radial.tune$best.parameters$cost,
                     gamma=radial.tune$best.parameters$gamma,
                     scale=FALSE )

train_radial = err_rate(svm_radial.tune, train) # training error rate
test_radial=err_rate(svm_radial.tune, test) # test error rate

plot(svm_radial.tune, train) # plot radial SVM on the training set
title(main=paste0("Train data, ","cost =",radial.tune$best.parameters$cost,
                  ", gamma =",poly.tune$best.parameters$degree),line =3)
```



```r
plot(svm_radial.tune, test) # plot radial SVM on the test set
title(main=paste0("Test data, ","cost =",radial.tune$best.parameters$cost,
                  ", gamma =",poly.tune$best.parameters$degree),line =3)
```



```r
summary(svm_radial.tune)
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "radial", cost = radial.tune$best.parameters$cost,
##     gamma = radial.tune$best.parameters$gamma, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##
## Number of Support Vectors:  62
##
##  ( 31 31 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

```r
# Add radial SVM error rates to error matrix
error_matrix = cbind(error_matrix,
  Radial = c(train_radial, test_radial))
error_matrix
##          Linear Polynomial     Radial
## Train 0.1818182 0.03030303 0.04545455
## Test  0.2058824 0.08823529 0.02941176
```

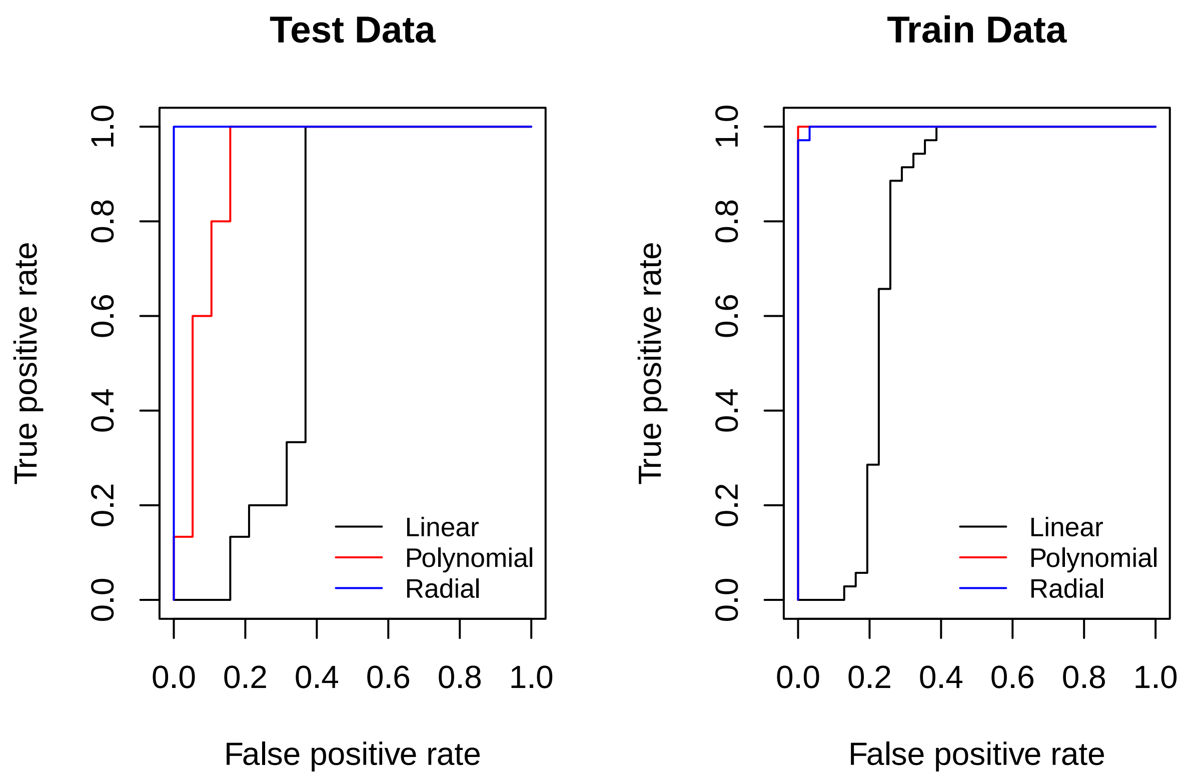
The tuned radial SVM has cost c = 0.1 and gamma = 4. The relatively small cost parameter allows a wider margin and some training misclassifications in exchange for a smoother decision boundary. The model uses 62 support vectors (31 from each class), indicating that many observations lie close to the margin and contribute to shaping the boundary. In contrast to the polynomial SVM, which achieves a slightly lower training error with only 8 support vectors, the radial SVM fits the training data more smoothly and relies on a larger number of points to define its boundary.

Overall, we can clearly see that the linear SVM performed worse than both the radial and polynomial SVMs, with training error rate 18.18%, while in radial SVM this number is 4.55% and 3.03% in polynomial SVM which is the smallest - the best performance model on training set is polynomial SVM.

Up to this point, I have focused on comparing the models using their training error rates. To evaluate how well these models generalize, I next examine their test error rates and ROC curves, which provide a more complete picture of their performance on unseen data.

```r
# Graph ROC curves
library(ROCR)

# Create ROC curves plot function
rocplot = function(pred, truth, ...) {
  predob = prediction(pred, truth)
  perf = performance(predob, "tpr", "fpr")
  plot(perf, ...)}

# Get SVM decision values on the test set for linear, polynomial, and radial models
test_linear_fitted = attributes(predict(svm_linear.tune,
                                   test,
                                   decision.values=T))$decision.values
test_poly_fitted = attributes(predict(svm_poly.tune,
                                   test,
                                   decision.values=T))$decision.values
test_radial_fitted=attributes(predict(svm_radial.tune,
                                   test,
                                   decision.values=T))$decision.values

# Get SVM decision values on the training set for linear, polynomial, and radial models
train_linear_fitted = attributes(predict(svm_linear.tune,
                                   train,
                                   decision.values=T))$decision.values
train_poly_fitted = attributes(predict(svm_poly.tune,
                                   train,
                                   decision.values=T))$decision.values
train_radial_fitted=attributes(predict(svm_radial.tune,
                                   train,
                                   decision.values=T))$decision.values

# Graph ROC curves (both training set and test set, side by side)
```

```r
par(mfrow = c(1, 2))

rocplot(test_linear_fitted,test$y, main="Test Data")
rocplot(test_poly_fitted,test$y, add=T, col="red")
rocplot(test_radial_fitted,test$y, add=T, col="blue")

legend("bottomright",
       legend = c("Linear", "Polynomial", "Radial"),
       col    = c("black",  "red",        "blue"),
       lty    = 1,
       bty    = "n",
       cex    = 0.8)

rocplot(train_linear_fitted,train$y, main="Train Data")
rocplot(train_poly_fitted,train$y, add=T, col="red")
rocplot(train_radial_fitted,train$y, add=T, col="blue")

legend("bottomright",
       legend = c("Linear", "Polynomial", "Radial"),
       col    = c("black",  "red",        "blue"),
       lty    = 1,
       bty    = "n",
       cex    = 0.8)
```



```
# Error matrix table
print(error_matrix)
##           Linear Polynomial     Radial
## Train 0.1818182 0.03030303 0.04545455
## Test  0.2058824 0.08823529 0.02941176
```

On the test set, the ranking of the models is different. The linear SVM performed worst(20.59% test error rate), polynomial SVM performed better (8.82% test error rate) and radial SVM performed the best (2.94% test error rate). We can also see how the order has changed by looking at the "Test Data" ROC curves, where the radial SVM curve lies closest to the top-left corner (highest true positive rate for a given false positive rate), the polynomial SVM sits in the middle, and the linear SVM curve stays farthest from the top-left. The change in order can be due to the test set is relatively small, even one or two changes in predictions can cause noticeable fluctuation in test error rates.

Besides, in polynomial SVM, only 8 support vectors, while in the radial SVM, there are 62 vectors, indicating that radial SVM fits the training data more flexibly but gets the lowest test error among all models. This suggests that the radial kernel provides a better bias–variance tradeoff and performance on unseen data, and the polynomial model fits the training data very closely but doesn't transfer as well to unseen data.

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

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

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 occurred at

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```r
library (ISLR2)
library(tree)
set.seed(200)
OJ.train.index = sample (dim(OJ)[1], 800)
OJ.train=OJ[OJ.train.index,]
OJ.test=OJ[-OJ.train.index,]
```

(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?

```r
OJ.tree= tree(Purchase ~ ., data=OJ.train)
summary(OJ.tree)
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  6
## Residual mean deviance:  0.7964 = 632.4 / 794
## Misclassification error rate: 0.1713 = 137 / 800
# Deviance is based on entropy, smaller deviance indicates
# a better fit to the training data.
```

Out of all variables in OJ data set, only LoyalCH, ListPriceDiff, PctDiscMM were used to construct the classification tree. The training misclassification error rate is 17.13%, with residual mean deviance to be 0.7964, a moderate impurity, suggesting the tree can be improved.

(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.

```r
OJ.tree
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1075.000 CH ( 0.60250 0.39750 )
##    2) LoyalCH < 0.445519 279  285.200 MM ( 0.20789 0.79211 )
##      4) LoyalCH < 0.0356415 55    9.996 MM ( 0.01818 0.98182 ) *
##      5) LoyalCH > 0.0356415 224  254.100 MM ( 0.25446 0.74554 ) *
##    3) LoyalCH > 0.445519 521  500.800 CH ( 0.81382 0.18618 )
##      6) LoyalCH < 0.764572 269  338.600 CH ( 0.67658 0.32342 )
##       12) ListPriceDiff < 0.235 110  151.900 MM ( 0.46364 0.53636 )
##         24) PctDiscMM < 0.196196 86  118.100 CH ( 0.55814 0.44186 ) *
##         25) PctDiscMM > 0.196196 24   18.080 MM ( 0.12500 0.87500 ) *
##       13) ListPriceDiff > 0.235 159  148.000 CH ( 0.82390 0.17610 ) *
##      7) LoyalCH > 0.764572 252   84.130 CH ( 0.96032 0.03968 ) *
```

Those with "*" at the end are terminal nodes.

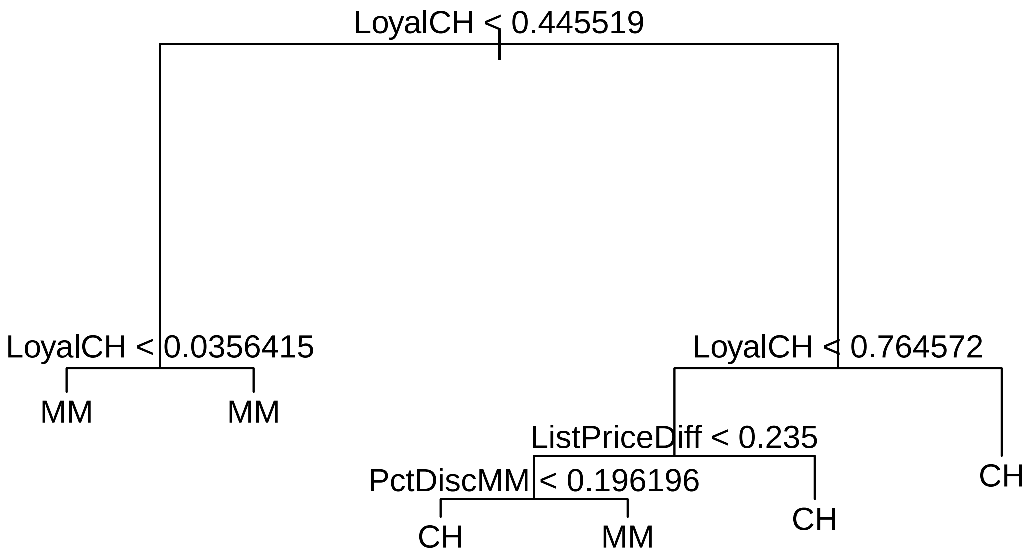
Let us pick "LoyalCH < 0.0356415 55 9.996 MM (0.01818 0.98182) *"

This node is created after a split on the LoyalCH variable. LoyalCH < 0.0356 indicates a very non-loyal CH customers who always buy MM, more specifically, they bought CH less than 3.5% of the time. There are

55 customers in this node. 9.996 is the residual deviance which measures impurity, 9.996 is considered very small number, indicating this is a pure node meaning that almost all the observations in this node belong to the same class which is MM. (0.01818 0.98182) is the probability of which brand the customers are, there is 1.8% CH customers and 98.2% MM customers, further confirms this is an extremely pure node.

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

```r
# Graph the tree
plot (OJ.tree, main="Unpruned Tree")
text(OJ.tree, pretty=0)
```



Based on the tree plot, LoyalCH is at the top, indicating that LoyalCH is the most important variable. The tree further confirms that my chosen node has extremely low impurity, as the split at LoyalCH < 0.0356 results in two MM groups. Even though the model would still be correct if there were just one MM group instead of two, splitting into two increases the confidence that the observations are assigned to the correct node as it decreases the impurity. Overall, we can see that MM is actually more attractive to customers, since LoyalCH has to be greater than 0.76 for customers to be placed confidently in the CH group.

Besides loyalty, ListPriceDiff - how much more expensive or cheaper MM is compared to CH based on the posted list price - is the second most important variable in determining the split. The next important variable is PctDiscMM, which is the percentage discount for MM. If the MM posted price minus the CH posted price is less than 0.235, and the MM discount is greater than 19.6%, customers are more likely to be in the MM group. In all other cases, customers are placed in the CH group.

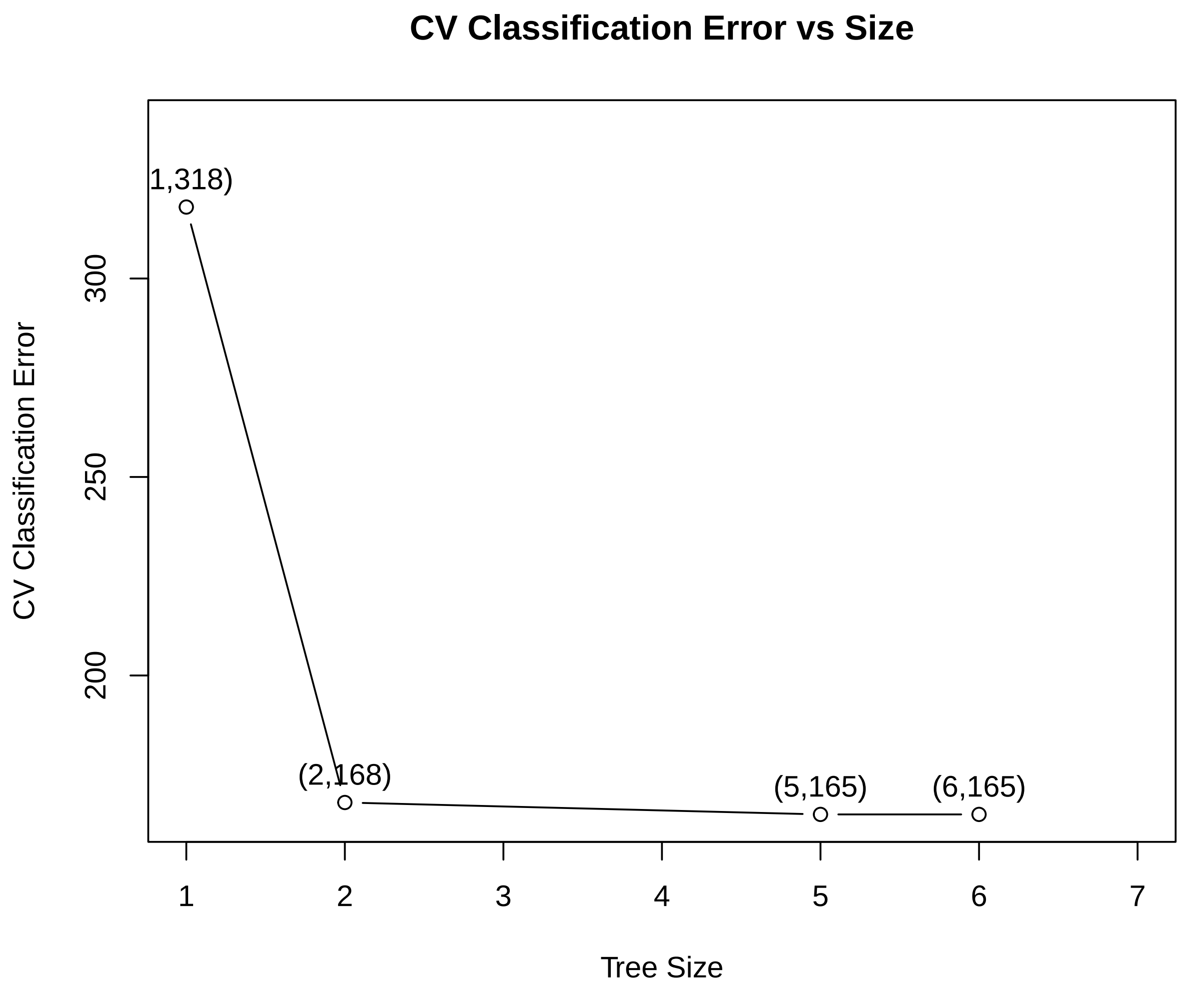
(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```r
# Use training tree to predict test set

OJ.tree.test = predict(OJ.tree, newdata = OJ.test, type = "class")
table(OJ.tree.test, OJ.test$Purchase)
##
## OJ.tree.test  CH  MM
##           CH 147  24
##           MM  24  75
OJ.err=mean(OJ.tree.test != OJ.test$Purchase)
OJ.err
## [1] 0.1777778
# I create a data frame that contains classification error rate.
classification_err_rate= data.frame(test= OJ.err) # add the test error rate
```

The test error rate is 17.8%, meaning that 17.8% of observations were misclassified.

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

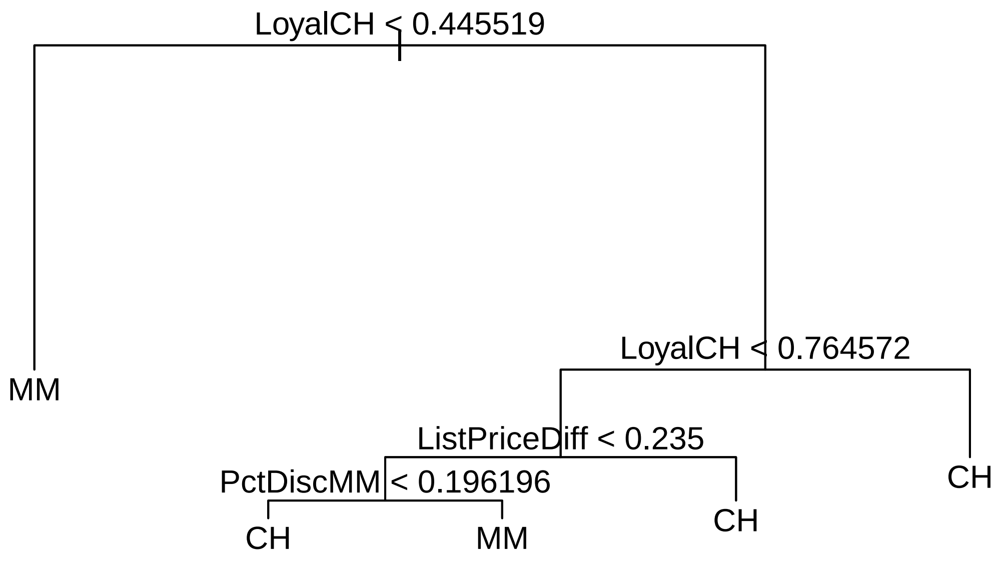
```r
# Implement cross-validation on tree
OJ.cv.tree= cv.tree(OJ.tree, FUN = prune.misclass)
names(OJ.cv.tree)
## [1] "size"   "dev"    "k"      "method"
OJ.cv.tree
## $size
## [1] 6 5 2 1
##
## $dev
## [1] 165 165 168 318
##
## $k
## [1] -Inf    0    6  163
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
# Graph the "Cross-Validatoin Error Rate vs Size" graph

plot(OJ.cv.tree$size, OJ.cv.tree$dev, type="b",
     xlab="Tree Size",
     xlim = c(min(OJ.cv.tree$size), max(OJ.cv.tree$size) + 1),
     ylab="CV Classification Error",
     ylim = c(min(OJ.cv.tree$dev), max(OJ.cv.tree$dev) + 20),
     main="CV Classification Error vs Size")

# Label points in "Cross-Validation Error Rate vs Size" graph
for (i in (1:length(OJ.cv.tree$size))) {
  text(OJ.cv.tree$size[i], OJ.cv.tree$dev[i],
       paste0("(",OJ.cv.tree$size[i],",",OJ.cv.tree$dev[i],")"),
       pos=3)}
```



According to the graph, trees with sizes 5 and 6 have the lowest CV classification error.

(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.

```r
# Prune the tree and plot
OJ.tree.prune= prune.misclass(OJ.tree, best=5)
plot(OJ.tree.prune, main="Pruned Tree")
text(OJ.tree.prune, pretty=0)
```



```
summary(OJ.tree.prune)
##
## Classification tree:
## snip.tree(tree = OJ.tree, nodes = 2L)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  5
## Residual mean deviance:  0.822 = 653.5 / 795
## Misclassification error rate: 0.1713 = 137 / 800
```

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

```
# This extra steps would allow me to retrieve training error rate from the code,
# I also want to update classification error rate data frame.
# (can be ignored)
train_pred=predict(OJ.tree, OJ.train,type="class")
table(train_pred, OJ.train$Purchase)
##
## train_pred  CH  MM
##         CH 421  76
##         MM  61 242
train_err_rate= mean(train_pred != OJ.train$Purchase)

train_pred.prune=predict(OJ.tree.prune, OJ.train,type="class")
table(train_pred.prune, OJ.train$Purchase)
```

```
##
## train_pred.prune  CH  MM
##               CH 421  76
##               MM  61 242
train_err_rate.prune= mean(train_pred.prune != OJ.train$Purchase)

classification_err_rate$train = train_err_rate # add the training error rate

classification_err_rate$train.prune = train_err_rate.prune # error rate after pruning

classification_err_rate
##        test   train train.prune
## 1 0.1777778 0.17125     0.17125
```

After pruning, the training classification error rate stays the same. This happens because the pruned split was not actually improving classification performance: the split nodes predicted the same class as the parent node (obtained after pruning), so removing that split does not change any predicted labels and therefore does not change the error rate.

In the cross-validation plot of classification error versus tree size, we can see that the tree with 6 terminal nodes has essentially the same cross-validation error as the tree with 5 terminal nodes. In this case, because the larger tree does not give a clear improvement, it is reasonable to choose the smaller, 5-node tree for simplicity.

The order of important variables remains the same, with LoyalCH being the most important, followed by ListPriceDiff, and then PctDiscMM.

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

```
OJ.tree.pred= predict(OJ.tree.prune, OJ.test, type="class")
table(OJ.tree.pred, OJ.test$Purchase)
##
## OJ.tree.pred  CH  MM
##           CH 147  24
##           MM  24  75
OJ.err.prune=mean(OJ.tree.pred != OJ.test$Purchase)

# Update classification error rate table
classification_err_rate$test.prune = OJ.err.prune
rownames(classification_err_rate) = c("Classification error rate")
classification_err_rate=classification_err_rate[,c(2,3,1,4)]
classification_err_rate
##                             train train.prune      test test.prune
## Classification error rate 0.17125     0.17125 0.1777778  0.1777778
```

Similar to the previous argument, the pruned tree has the same test classification error rate as the unpruned tree. Pruning removed splits that did not change the predicted classes, so the overall misclassification rate stayed the same.

As expected from the bias–variance trade-off, the test error rate is higher than the training error rate for both models. Using cross-validation to evaluate trees of different sizes, we find that the tree with 5 terminal nodes achieves (approximately) the lowest cross-validated classification error while being the simplest model. Therefore, we select the 5-node tree as our final model.