The Final Project

Julian Frank

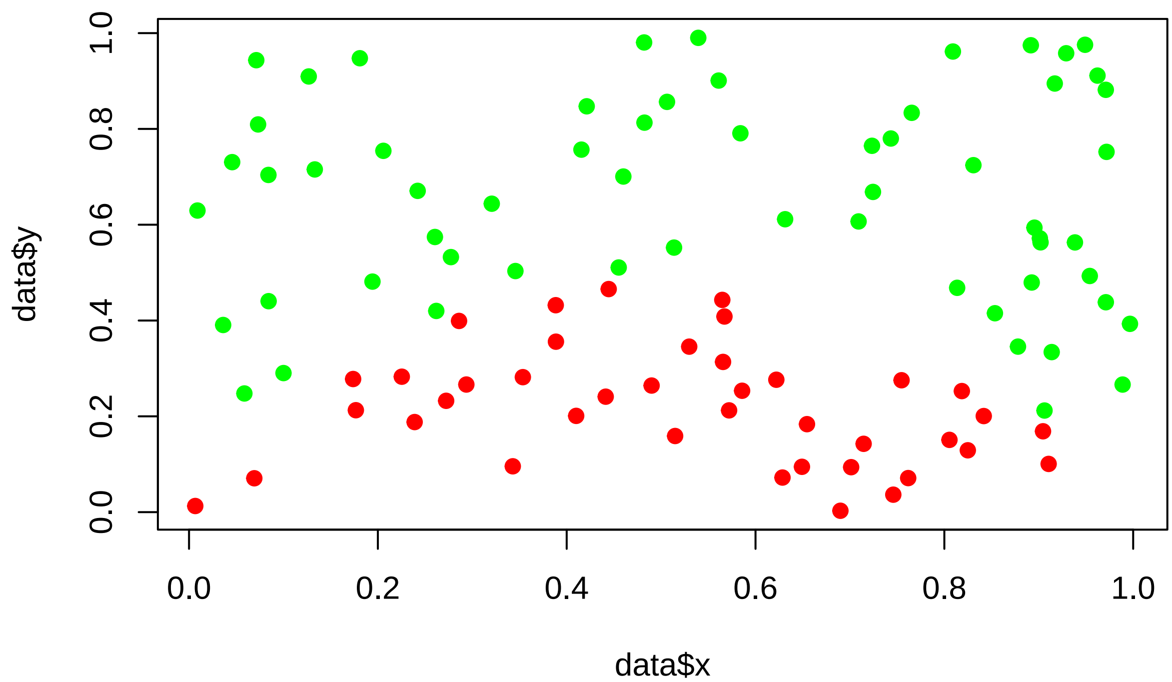
2024-08-30

Part 1

Let's generate a simulated two-class data set with 100 observations and two features in which there is a visible but non-linear separation be tween the two classes. I will show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the train ing data. I will also examine which technique performs best on the test data.

```r
# Generating data separated by a polynomial class decision boundary
set.seed(4)
data <- data.frame(
  x = runif(100, 0),
  y = runif(100, 0)
)
decision_boundary <- 2*(data$x-0.5)^2 + (data$y) - 0.5
data$class <- factor(ifelse(decision_boundary < 0, "A", "B"))
# data

plot(data$x, data$y, pch=19, col = c("red", "green")[data$class])
```



Let's fit the SVMs to the training data

```
library(e1071)

# Split data into training and test sets
set.seed(7)
train <- 1:70
test <- 71:100

train_data <- data[train, ]
test_data <- data[-train, ]

# Fit the SVM models to the training data
poly_fit <- svm(class ~ ., data = train_data, kernel = "polynomial", degree = 3)
rad_fit <- svm(class ~ ., data = train_data, kernel = "radial")
lin_fit <- svm(class ~ ., data = train_data, kernel = "linear")
```

Now let's see how well they perform on the training data

```
# Predictions on training data
train_pred_poly <- predict(poly_fit, train_data)
train_pred_rad <- predict(rad_fit, train_data)
train_pred_lin <- predict(lin_fit, train_data)

# Training data confusion matrices
```

```r
print("Confusion matrices:")
## [1] "Confusion matrices:"
table(train_pred_poly, train_data$class)
##
## train_pred_poly  A  B
##               A 18  3
##               B 11 38
table(train_pred_rad, train_data$class)
##
## train_pred_rad  A  B
##              A 29  2
##              B  0 39
table(train_pred_lin, train_data$class)
##
## train_pred_lin  A  B
##              A 26  3
##              B  3 38
# Classification error rates
# Error rate for polynomial SVM
mean(train_pred_poly != train_data$class)
## [1] 0.2
# Error rate for radial SVM
mean(train_pred_rad != train_data$class)
## [1] 0.02857143
# Error rate for Linear SVM
mean(train_pred_lin != train_data$class)
## [1] 0.08571429
```

The svm with a radial kernel performs the best on the training dataset with the lowest classification error rate of 0.02857143.

Now let's see how well they perform on the test data

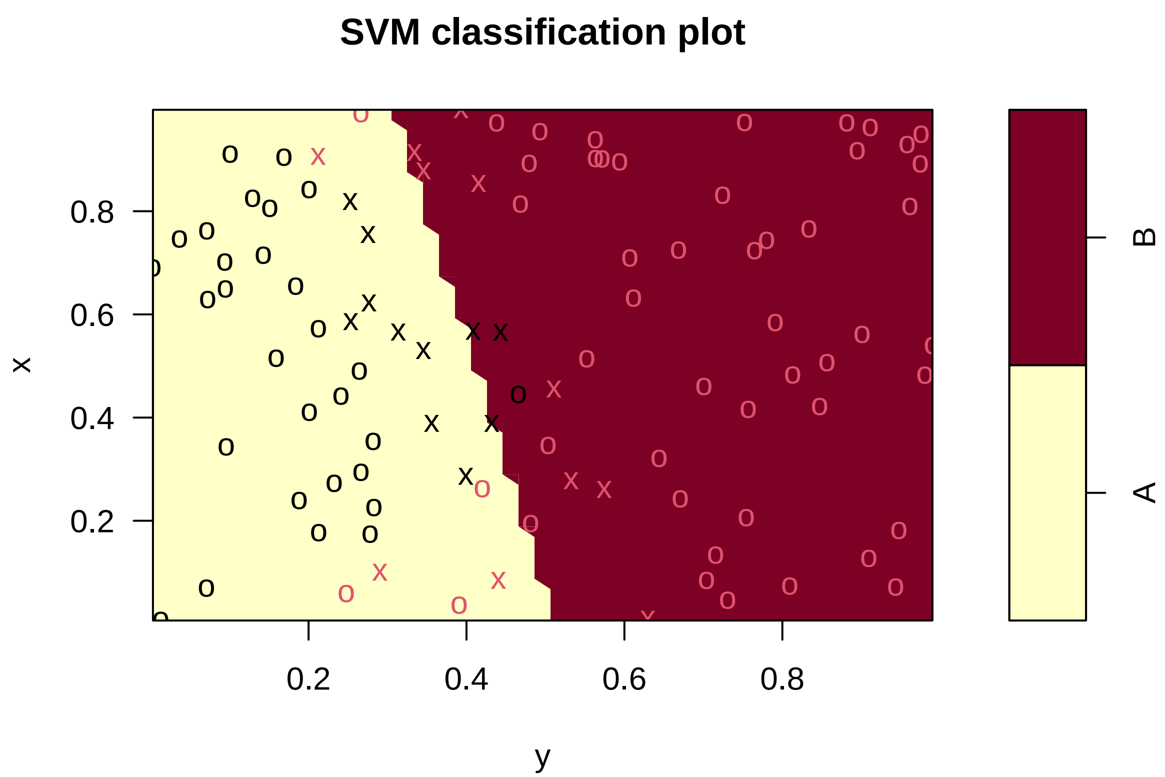
```r
# Predictions on test data
test_pred_poly <- predict(poly_fit, test_data)
test_pred_rad <- predict(rad_fit, test_data)
test_pred_lin <- predict(lin_fit, test_data)

# Test data confusion matrices
table(test_pred_poly, test_data$class)
##
## test_pred_poly  A  B
##              A 10  3
##              B  1 16
table(test_pred_rad, test_data$class)
##
## test_pred_rad  A  B
##             A 11  2
##             B  0 17
table(test_pred_lin, test_data$class)
##
## test_pred_lin  A  B
##             A 10  4
##             B  1 15
```

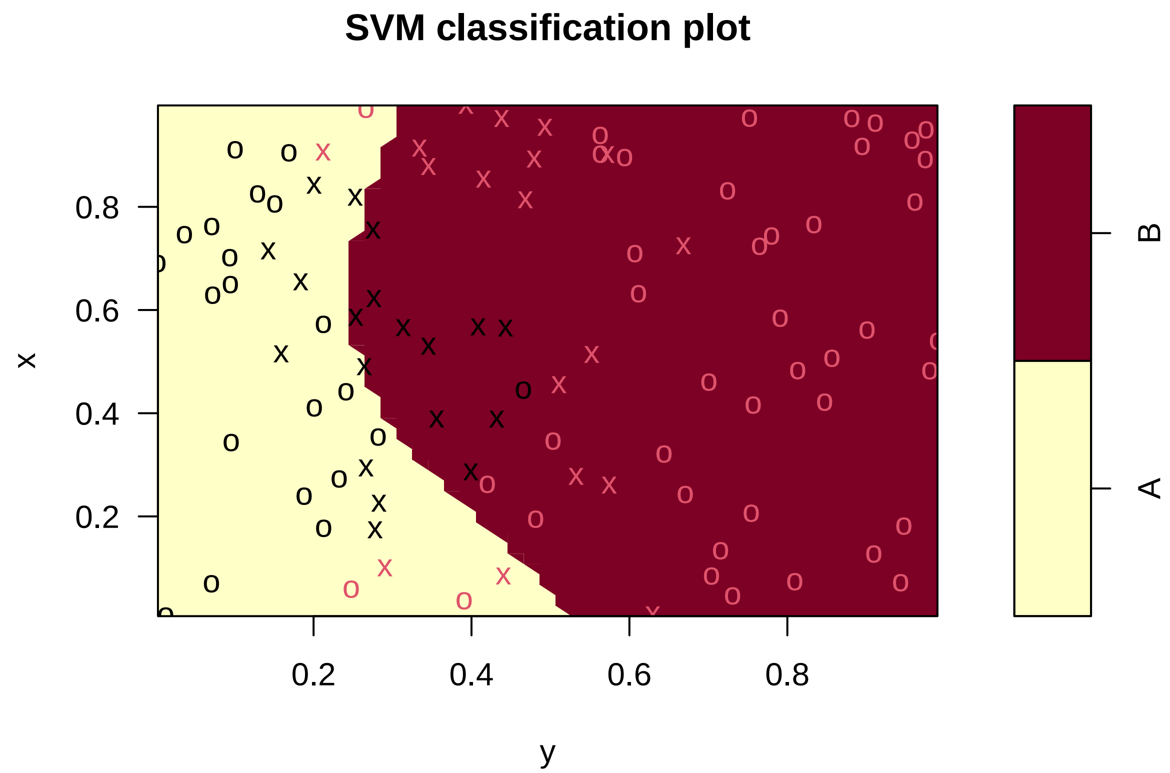
```r
# Error rate for polynomial SVM:
mean(test_pred_poly != test_data$class)
## [1] 0.1333333
# Error rate for radial SVM
mean(test_pred_rad != test_data$class)
## [1] 0.06666667
# Error rate for linear SVM
mean(test_pred_lin != test_data$class)
## [1] 0.1666667
```

Again, the SVM with a radial kernel performs the best with the test data, yielding the lowest classification error rate of 0.06666667

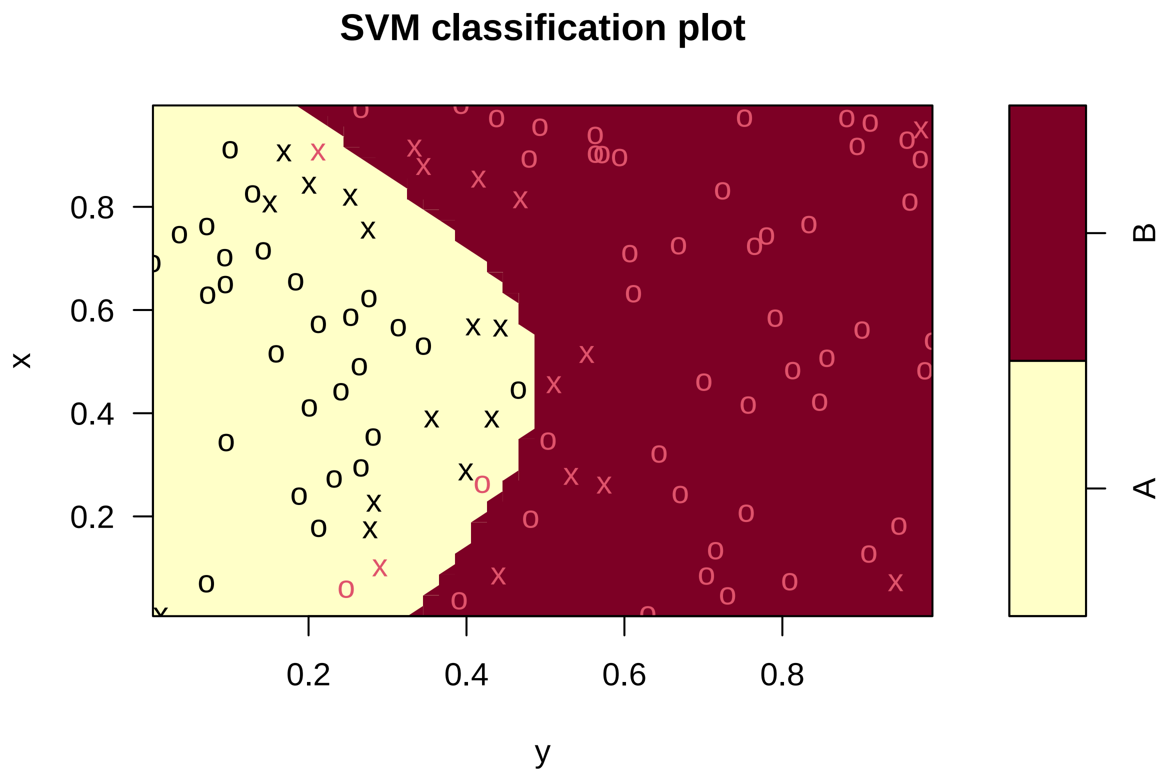
```r
# Plot the decision boundaries
# Linear SVM (Support Vector Classifier)
plot(lin_fit, data)
```




```r
# Polynomial SVM
plot(poly_fit, data)
```




```
# Radial SVM
plot(rad_fit, data)
```



Looking at the decision boundaries for all three svm models, it is clear that the radial kernel SVM is closest to the actual decision boundary. As the actual decision boundary is non-linear, it outperforms the support vector classifier.

Part 2

Now let's look at the OJ data set from the ISLR2 package. I'll first create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```r
library(ISLR2)
library(tree)
## Warning: package 'tree' was built under R version 4.3.3
data <- ISLR2::OJ
#attach(data)
set.seed(1)

training_indices <- sample(1:nrow(data), 800, replace = FALSE)
train_data <- data[training_indices, ]
test_data <- data[-training_indices, ]
# train_data
```

Now let's fit a tree to the training data, with Purchase as the response and the other variables as predictors. I'll use the summary() function to produce summary statistics about the tree, and describe the results obtained.

```r
tree_fit <- tree(Purchase ~ ., data = train_data)
summary(tree_fit)
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train_data)
## 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 recursive binary splitting algorithm ends up using five features listed above for the tree. The tree training error rate is 0.1588, and the tree has 9 terminal nodes.

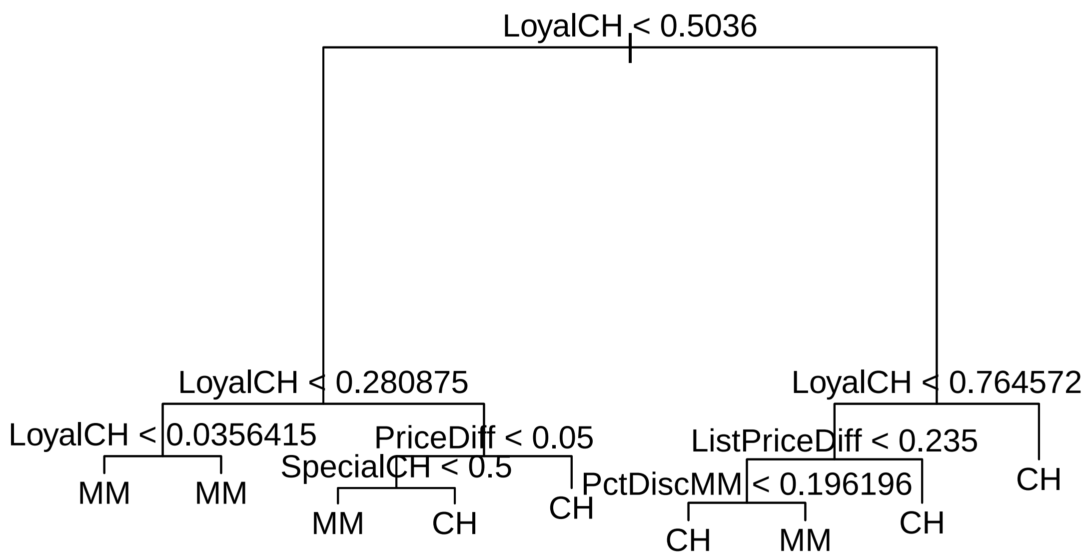
Now let's focus on one of the terminal nodes and interpret the information displayed.

```r
tree_fit
## 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 ) *
```

Let's examine the terminal node 8) within the tree: The overall split criterion for this class is LoyalCH less than 0.0356415. There are 59 observations within this branch, and the deviance of the branch is 10.14. The overall prediction for data points that fall within this class is MM.

Let's create a plot of the tree and interpret the results.

```r
plot(tree_fit)
text(tree_fit)
```



The tree will determine the class of different data points first by considering whether the LoyalCH value for that point is less than 0.5036, then moves down the decision path according to whether the split condition printed on each split is true or false. We see that of the terminal nodes, four are classified with Purchase = MM, and five are classified with Purchase = CH.

Now we can predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels.

```
pred <- predict(tree_fit, test_data, type="class")
#pred
table(pred, test_data$Purchase)
##
## pred  CH  MM
##   CH 160  38
##   MM   8  64
error_rate <- mean(pred != test_data$Purchase)
error_rate
## [1] 0.1703704
```

The test error rate of this tree is 0.1703704.

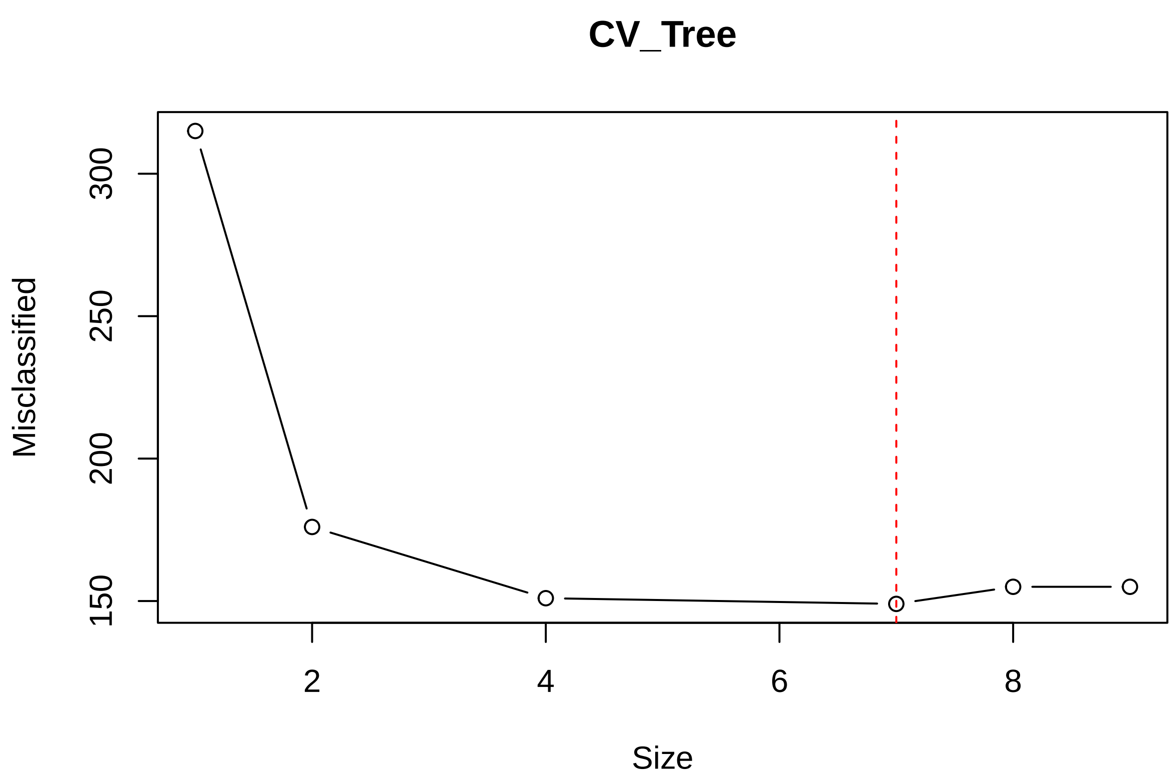
Let's apply the cv.tree() function to the training set in order to determine the optimal tree size.

```
set.seed(134)
cv_tree <- cv.tree(tree_fit, FUN = prune.misclass)
cv_tree
```

```
## $size
## [1] 9 8 7 4 2 1
##
## $dev
## [1] 155 155 149 151 176 315
##
## $k
## [1]       -Inf   0.000000   3.000000   4.333333  10.500000 151.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

For visual reference, I will produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

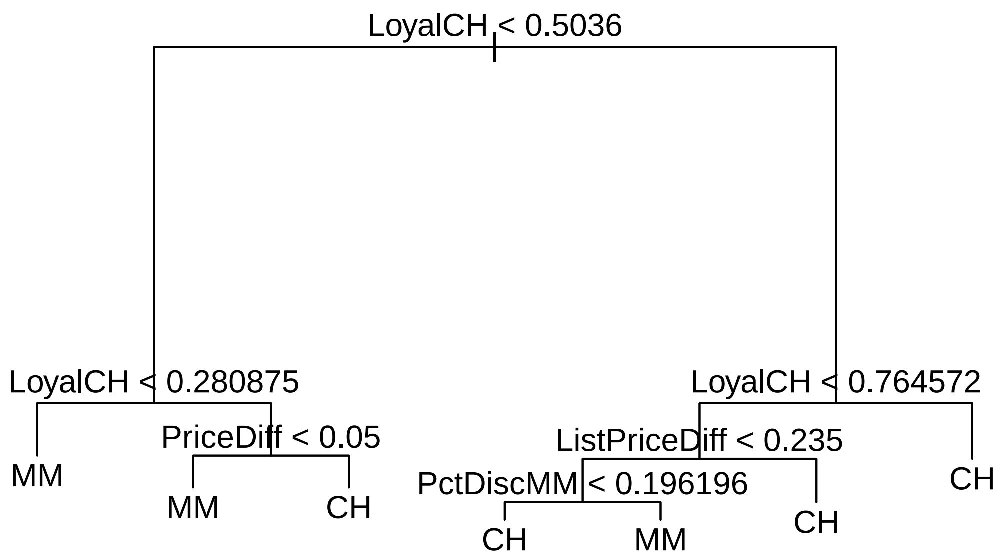
```
plot(cv_tree$size, cv_tree$dev, type = "b", xlab = "Size", ylab = "Misclassified", main = "CV_Tree")
min_dev <- which.min(cv_tree$dev)
abline(v = cv_tree$size[min_dev], lty = 2, col = "red")
```



The tree with size 7 (7 terminal nodes) has the lowest cross-validated classification error rate.

Let's prune the tree corresponding to the optimal tree size obtained using cross-validation:

```r
pruned_tree <- prune.misclass(tree_fit, best = cv_tree$size[min_dev])
plot(pruned_tree)
text(pruned_tree, pretty=0)
```



Now let's compare the training error rates between the pruned and un pruned trees to see which is higher.

```r
summary(pruned_tree)
##
## Classification tree:
## snip.tree(tree = tree_fit, nodes = c(4L, 10L))
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7748 = 614.4 / 793
## Misclassification error rate: 0.1625 = 130 / 800
summary(tree_fit)
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train_data)
## 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 training error rate for the pruned tree is 0.1625, which is higher than the the training error of the unpruned tree.

Now let's do the same for the test error rates:

```
pred_pruned <- predict(pruned_tree, test_data, type = "class")
table(pred_pruned, test_data$Purchase)
##
## pred_pruned  CH  MM
##          CH 160  36
##          MM   8  66
table(pred, test_data$Purchase)
##
## pred  CH  MM
##   CH 160  38
##   MM   8  64
mean(pred != test_data$Purchase)
## [1] 0.1703704
mean(pred_pruned != test_data$Purchase)
## [1] 0.162963
```

The test error rate for the unpruned tree is 0.1703704, which is higher than the test error rate for the pruned tree.