DS-6030 Homework Module 7

Tom Lever

07/08/2023

DS 6030 | Spring 2023 | University of Virginia

8. In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable.

   Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

   (a) Split the data set into a training set and a test set.

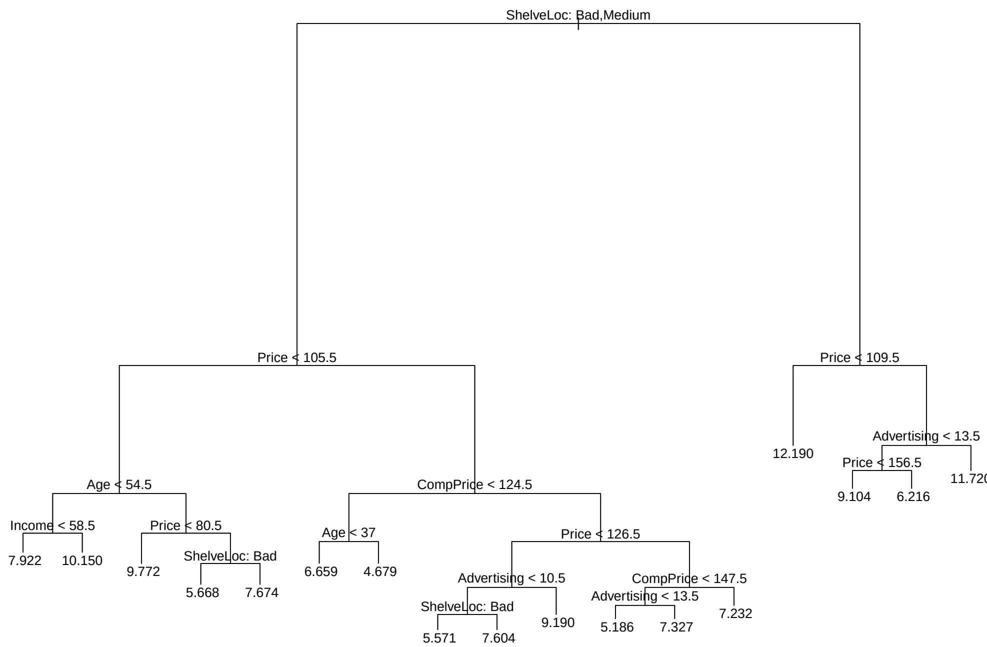
```
set.seed(1)
library(ISLR2)
library(TomLeversRPackage)
training_and_testing_data <- split_data_set_into_training_and_testing_data(
    Carseats,
    proportion_of_training_data = 0.9
)
training_data <- training_and_testing_data$training_data
testing_data <- training_and_testing_data$testing_data
```

   (b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```
library(tree)
full_tree <- tree(Sales ~ ., data = training_data)
summary(full_tree)
```

```
#
# Regression tree:
# tree(formula = Sales ~ ., data = training_data)
# Variables actually used in tree construction:
# [1] "ShelveLoc"   "Price"       "Age"         "Income"      "CompPrice"
# [6] "Advertising"
# Number of terminal nodes:  17
# Residual mean deviance:  2.653 = 910.1 / 343
# Distribution of residuals:
#     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
# -5.18600 -1.09000  0.05305  0.00000  1.08300  4.63100
```

```
plot(full_tree)
text(full_tree, pretty = 0)
```




```r
vector_of_predicted_sales <- predict(full_tree, newdata = testing_data)
vector_of_actual_sales <- testing_data$Sales
calculate_mean_squared_error(vector_of_predicted_sales, vector_of_actual_sales)
```

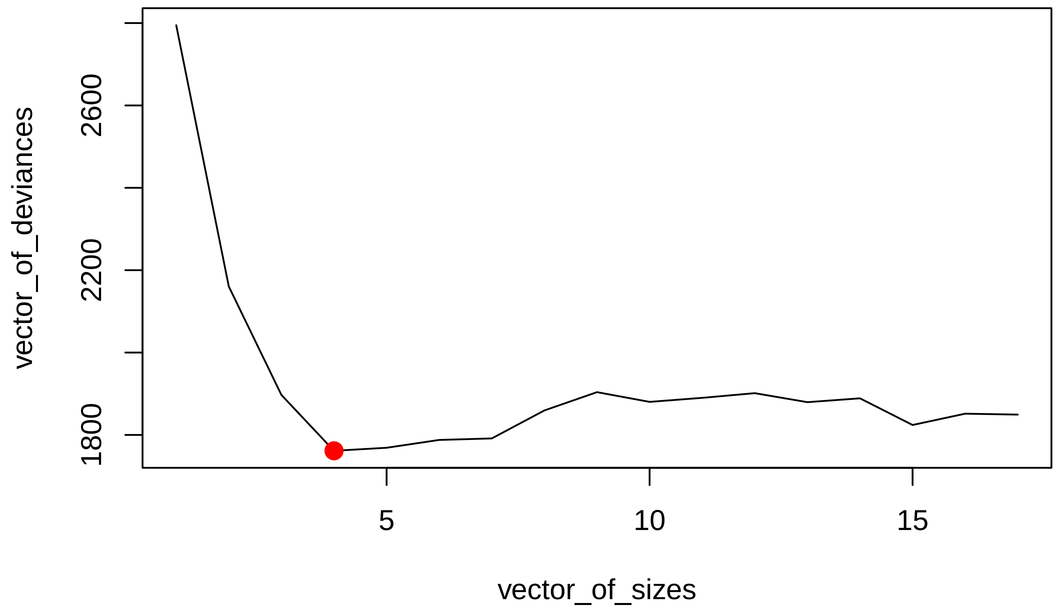
```
# [1] 4.896065
```

When shelf location is good and price is less than 109.5 monetary units, our tree predicts that 12.190 thousand child car seats will be sold at each location in each time period.

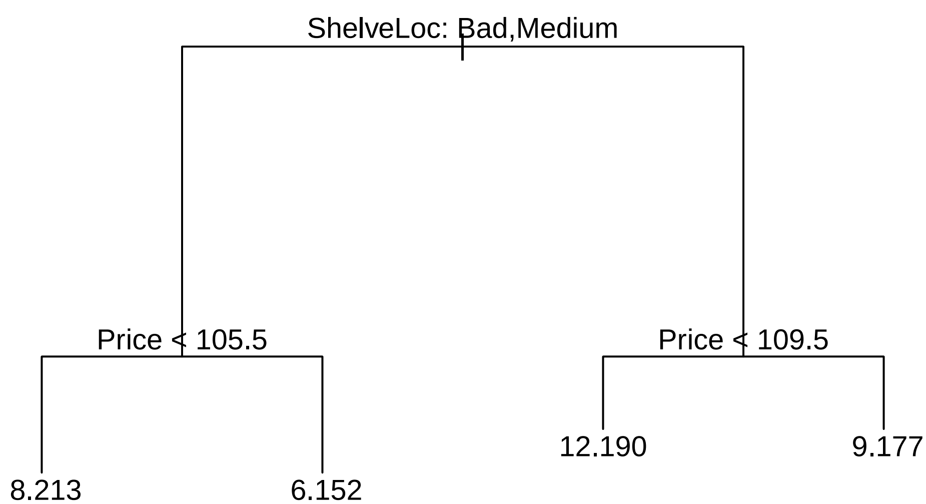
The test Mean Squared Error of our tree when predicting sales is 4.896 $thousand^2$.

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```r
object_of_types_prune_and_tree_sequence <- cv.tree(full_tree)
vector_of_sizes <- object_of_types_prune_and_tree_sequence$size
vector_of_deviances <- object_of_types_prune_and_tree_sequence$dev
plot(vector_of_sizes, vector_of_deviances, type = "l")
index_of_minimum_deviance <- which.min(vector_of_deviances)
optimal_size <-
    vector_of_sizes[index_of_minimum_deviance]
minimum_deviance <- min(vector_of_deviances)
points(
    optimal_size,
    minimum_deviance,
    col = "red",
    cex = 2,
    pch = 20
)
```




```r
pruned_tree <- prune.tree(full_tree, best = optimal_size)
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```



```r
vector_of_predicted_sales <- predict(pruned_tree, newdata = testing_data)
calculate_mean_squared_error(vector_of_predicted_sales, vector_of_actual_sales)
```

```
# [1] 6.785063
```

The test Mean Squared Error for the pruned tree is greater and less desirable than the Mean Squared Error for the full tree.

(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important.

Per *An Introduction to Statistical Learning* (Second Edition), bagging "is simply a special case of a random forest with [the number of variables randomly sampled as candidates at each split] $m = p$ the number of predictors."

```r
library(randomForest)
```

```
# randomForest 4.7-1.1

# Type rfNews() to see new features/changes/bug fixes.
```

```r
index_of_column_Sales <-
    get_index_of_column_of_data_frame(training_data, "Sales")
data_frame_of_predictors <- training_data[, -index_of_column_Sales]
data_frame_of_sales <- training_data[, index_of_column_Sales]
number_of_predictors <- ncol(data_frame_of_predictors)
get_test_MSE_and_vector_of_ordered_percents_increase_in_MSE_for_random_forest <-
    function(mtry) {
    the_randomForest <- randomForest(
        formula = Sales ~ .,
        data = training_data,
        mtry = mtry,
        importance = TRUE
    )
    vector_of_predicted_sales <-
        predict(the_randomForest, newdata = testing_data)
    test_MSE <- calculate_mean_squared_error(
        vector_of_predicted_sales,
        vector_of_actual_sales
    )
    matrix_of_importance_metrics <- importance(the_randomForest)
    vector_of_percents_increase_in_MSE <-
       matrix_of_importance_metrics[, "%IncMSE"]
    vector_of_indices_of_ordered_percents_increase_in_MSE <-
        order(vector_of_percents_increase_in_MSE, decreasing = TRUE)
    vector_of_ordered_percents_increase_in_MSE <-
        vector_of_percents_increase_in_MSE[
            vector_of_indices_of_ordered_percents_increase_in_MSE
        ]
    list_of_test_MSE_and_vector_of_ordered_percents_increase_in_MSE_for_random_forest <-
        list(
            test_MSE = test_MSE,
            vector_of_ordered_percents_increase_in_MSE =
                vector_of_ordered_percents_increase_in_MSE
        )
    return(
```

```
        list_of_test_MSE_and_vector_of_ordered_percents_increase_in_MSE_for_random_forest
    )
}
get_test_MSE_and_vector_of_ordered_percents_increase_in_MSE_for_random_forest(
    mtry = number_of_predictors
)
```

```
# $test_MSE
# [1] 2.912954
#
# $vector_of_ordered_percents_increase_in_MSE
#   ShelveLoc       Price   CompPrice Advertising         Age      Income
#   81.267672   79.323197   38.593171   25.822124   25.716498   14.313945
#   Education          US       Urban  Population
#    2.885808    2.270115   -1.691179   -2.211183
```

The test Mean Squared Error for our bootstrap aggregation (BAg) is 2.913, which is 0.595 of the MSE for our full tree and 0.429 of the MSE for our pruned tree.

According to In a random forest, is larger %IncMSE better or worse?, "%IncMSE is the most robust and informative measure. IT is the increase in mse of predictions(estimated with out-of-bag-CV) as a result of variable j being permuted(values randomly shuffled)... the higher the number, the more important."

%IncMSE is highest for *ShelveLoc* followed by *Price*; *ShelveLoc* and *Price* are the two most important variables.

(e) Use random forests to analyze this data. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
data_frame_of_values_of_mtry_and_test_MSEs <- data.frame(
    matrix(NA, nrow = number_of_predictors, ncol = 2)
)
colnames(data_frame_of_values_of_mtry_and_test_MSEs) <- c("mtry", "test_MSE")
for (mtry in 1:number_of_predictors) {
    print(paste("mtry: ", mtry, sep = ""))
    data_frame_of_values_of_mtry_and_test_MSEs[mtry, "mtry"] <- mtry
    test_MSE_and_vector_of_ordered_percents_increase_in_MSE <-
        get_test_MSE_and_vector_of_ordered_percents_increase_in_MSE_for_random_forest(
            mtry = mtry
        )
    test_MSE <- test_MSE_and_vector_of_ordered_percents_increase_in_MSE$test_MSE
    vector_of_ordered_percents_increase_in_MSE <-
        test_MSE_and_vector_of_ordered_percents_increase_in_MSE$
            vector_of_ordered_percents_increase_in_MSE
    print(vector_of_ordered_percents_increase_in_MSE)
    data_frame_of_values_of_mtry_and_test_MSEs[mtry, "test_MSE"] <- test_MSE
}
```

```
# [1] "mtry: 1"
#   ShelveLoc       Price         Age Advertising   CompPrice          US
#  27.4145011  22.6376857  12.1022390  11.8362136   9.1428466   6.3777495
#      Income   Education       Urban  Population
#   5.7156306   2.2635817  -0.2072851  -0.9091637
# [1] "mtry: 2"
#   ShelveLoc       Price Advertising         Age   CompPrice      Income
```

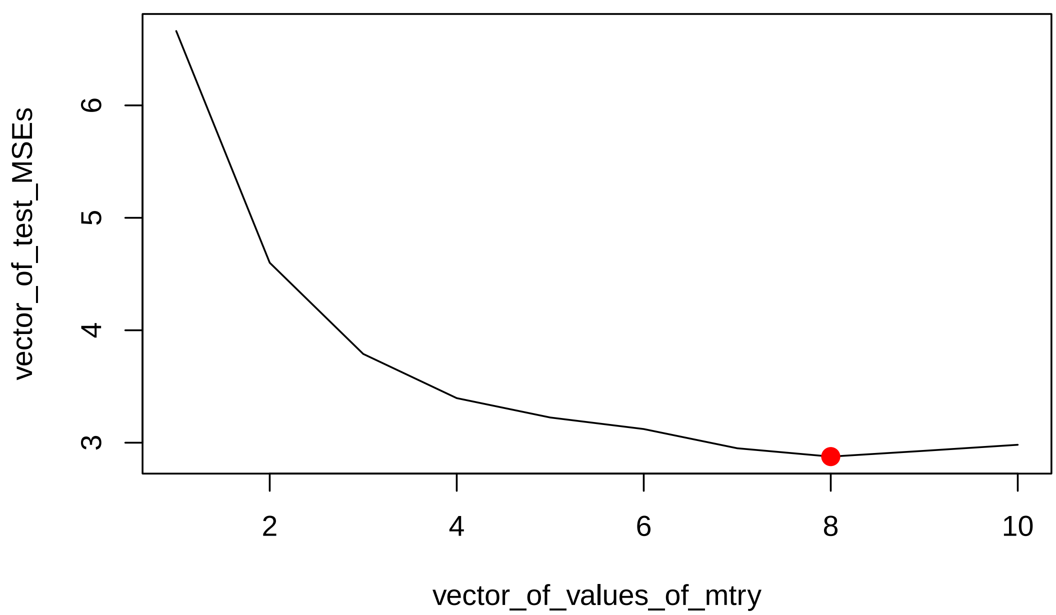
```
#   44.880078   37.860838   17.212010   16.137100   14.665327    6.413410
#          US   Education       Urban  Population
#    6.210358    1.257871   -1.022202   -1.798896
# [1] "mtry: 3"
#   ShelveLoc       Price Advertising         Age   CompPrice      Income
#   57.917795   48.708554   19.677702   19.528156   18.673141    7.957699
#          US   Education  Population       Urban
#    5.868734    3.956120   -2.433051   -2.642151
# [1] "mtry: 4"
#   ShelveLoc       Price   CompPrice         Age Advertising      Income
#   60.737660   56.391682   23.604401   21.854985   18.236436    8.220434
#          US   Education       Urban  Population
#    6.106552    1.320007   -1.574042   -1.839350
# [1] "mtry: 5"
#   ShelveLoc       Price   CompPrice         Age Advertising      Income
#   68.562969   61.860220   25.200712   23.425230   20.589795   10.621514
#          US   Education  Population       Urban
#    5.444433    3.662041   -1.106284   -2.907471
# [1] "mtry: 6"
#   ShelveLoc       Price   CompPrice         Age Advertising      Income
#   76.681110   67.104963   29.750847   24.706524   21.958387   11.035165
#          US   Education       Urban  Population
#    3.985181    2.732473   -1.483772   -2.223225
# [1] "mtry: 7"
#   ShelveLoc       Price   CompPrice         Age Advertising      Income
#  79.7220186  72.1876642  34.2439225  24.6451061  23.4062353  11.4780558
#          US   Education  Population       Urban
#   4.3765943   2.7168918  -0.4954921  -1.9197438
# [1] "mtry: 8"
#   ShelveLoc       Price   CompPrice         Age Advertising      Income
#  79.5222605  74.7577736  33.7322095  24.6121401  24.4295814  14.3477921
#          US   Education  Population       Urban
#   4.9868483   2.4070209  -0.2734359  -1.8060475
# [1] "mtry: 9"
#   ShelveLoc       Price   CompPrice         Age Advertising      Income
#   83.989493   78.130492   38.157897   26.980376   23.366049   13.876704
#   Education          US  Population       Urban
#    3.103712    2.183155   -1.673313   -2.381346
# [1] "mtry: 10"
#   ShelveLoc       Price   CompPrice Advertising         Age      Income
#   81.363209   77.647089   38.547174   27.190982   22.778387   14.981041
#          US   Education       Urban  Population
#    4.123539    3.332881   -1.602569   -2.302125
```

```r
print(data_frame_of_values_of_mtry_and_test_MSEs)
```

```
#    mtry test_MSE
# 1     1 6.661409
# 2     2 4.600594
# 3     3 3.789455
# 4     4 3.396240
# 5     5 3.224155
# 6     6 3.121334
# 7     7 2.950383
```

```
# 8     8 2.876444
# 9     9 2.927813
# 10   10 2.981307
```

```
vector_of_values_of_mtry <- data_frame_of_values_of_mtry_and_test_MSEs$mtry
vector_of_test_MSEs <- data_frame_of_values_of_mtry_and_test_MSEs$test_MSE
plot(
    x = vector_of_values_of_mtry,
    y = vector_of_test_MSEs,
    type = "l"
)
index_of_minimum_test_MSE <- which.min(vector_of_test_MSEs)
optimal_value_of_mtry <-
    vector_of_values_of_mtry[index_of_minimum_test_MSE]
minimum_test_MSE <- min(vector_of_test_MSEs)
points(
    optimal_value_of_mtry,
    minimum_test_MSE,
    col = "red",
    cex = 2,
    pch = 20
)
```



See above plot for test Mean Squared Errors for different values of the number of variables randomly sampled as candidates at each split m. Test MSE decreases parabolically with number of variables to a minimum for $m = 8$. In all cases *ShelveLoc* and *Price* are the most important predictors.

(f) Now analyze the data using BART, and report your results. (skip this exercise)

9. This problem involves the OJ data set which is part of the ISLR package.

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

```
training_and_testing_data <- split_data_set_into_training_and_testing_data(
    OJ,
    number_of_training_data = 800
)
training_data <- training_and_testing_data$training_data
testing_data <- training_and_testing_data$testing_data
```

(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?

```
full_tree <- tree(Purchase ~ ., data = training_data)
summary(full_tree)
```

```
#
# Classification tree:
# tree(formula = Purchase ~ ., data = training_data)
# Variables actually used in tree construction:
# [1] "LoyalCH"     "SalePriceMM" "SpecialCH"   "PriceDiff"   "STORE"
# Number of terminal nodes:  9
# Residual mean deviance:  0.7139 = 564.7 / 791
# Misclassification error rate: 0.1575 = 126 / 800
```

Our full tree is a classification tree that predicts whether a customer will purchase Citrus Hill or Minute Maid orange juice. A tree is grown by binary recursive partitioning using the response in the specified formula, $Purchase$, and choosing splits from the terms of the right-hand-side, which in our case are all terms besides $Purchase$. The predictors actually used in tree construction are $LoyalCH$, $SalePriceMM$, $SpecialCH$, $PriceDiff$, and $STORE$. $Purchase$ is a factor with levels CH and MM indicating whether the customer purchased Citrus Hill or Minute Maid Orange Juice. $LoyalCH$ seems to be a rate of customer brand loyalty for CH between 0 and 1. $SalePriceMM$ seems to be the net sale price of Minute Maid orange juice in dollars. $SpecialCH$ seems to be an indicator of whether or not there is a special on Citrus Hill orange juice. $PriceDiff$ seems to be net sale price of Minute Maid orange juice less net sale price of Citrus Hill orange juice in dollars. $STORE$ seems to be a categorical variable indicating at which of 5 possible stores the purchase occurred. In our full tree there are 9 terminal nodes / leaves. The deviance of our full tree is 564.7. A small deviance indicates a tree that provides a good fit to the training data. The residual mean deviance for our full tree is $564.7/(800-9)$. The training error rate / misclassification error rate for our full tree is $126/800$.

(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.

```
full_tree
```

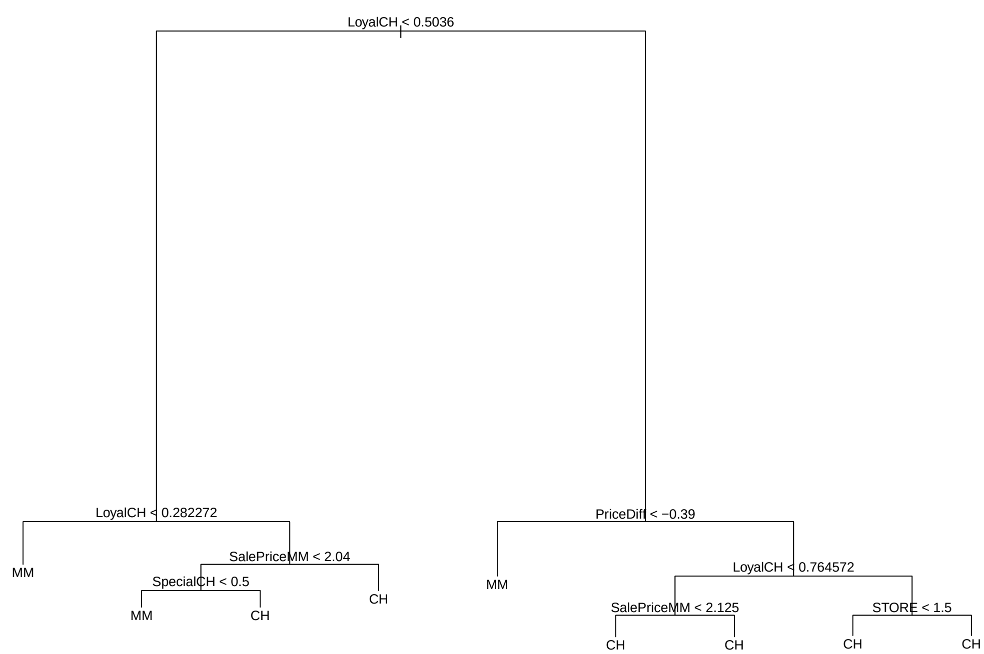
```
# node), split, n, deviance, yval, (yprob)
#       * denotes terminal node
#
#  1) root 800 1077.00 CH ( 0.60000 0.40000 )
#    2) LoyalCH < 0.5036 341  371.50 MM ( 0.23460 0.76540 )
#      4) LoyalCH < 0.282272 167  114.20 MM ( 0.10778 0.89222 ) *
#      5) LoyalCH > 0.282272 174  226.60 MM ( 0.35632 0.64368 )
#       10) SalePriceMM < 2.04 97  101.40 MM ( 0.21649 0.78351 )
#         20) SpecialCH < 0.5 75   58.90 MM ( 0.13333 0.86667 ) *
```

```
#         21) SpecialCH > 0.5 22   30.50 CH ( 0.50000 0.50000 ) *
#       11) SalePriceMM > 2.04 77  106.40 CH ( 0.53247 0.46753 ) *
#    3) LoyalCH > 0.5036 459  352.10 CH ( 0.87146 0.12854 )
#      6) PriceDiff < -0.39 22   27.52 MM ( 0.31818 0.68182 ) *
#      7) PriceDiff > -0.39 437  285.40 CH ( 0.89931 0.10069 )
#       14) LoyalCH < 0.764572 175  172.30 CH ( 0.80571 0.19429 )
#         28) SalePriceMM < 2.125 106  126.30 CH ( 0.71698 0.28302 ) *
#         29) SalePriceMM > 2.125 69   30.55 CH ( 0.94203 0.05797 ) *
#       15) LoyalCH > 0.764572 262   84.93 CH ( 0.96183 0.03817 )
#         30) STORE < 1.5 133    0.00 CH ( 1.00000 0.00000 ) *
#         31) STORE > 1.5 129   70.35 CH ( 0.92248 0.07752 ) *
```

A terminal node / leaf / branch to leaf is indicated by an asterisk. Because `full_tree` outputs information for each of two branches for each internal node / node other than the root node / trunk and the leaves, we speak in terms of branches. Let us consider the terminal node / leaf / branch to leaf labeled 4. The prediction of the full tree associated with this branch to leaf is MM. We arrive at this branch when the split criterion $LoyalCH$ is less than 0.504 and less than 0.282. The split criterion associated with this branch is $LoyalCH < 0.282$. The number of observations / purchases in our training data set is 800. Of those purchases, 341 purchases are by customers with loyalty to Citrus Hill less than 0.504. Of those purchases, 167 purchases are by customers with loyalty to Citrus Hill less than 0.282. The number of purchases associated with our branch is 167 with a deviance of 114.2. 0.108 of purchases associated with our branch were of Citrus Hill orange juice. 0.892 of purchases associated with our branch were of Minute Maid orange juice.

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

```
plot(full_tree)
text(full_tree, pretty = 0)
```



Per our full tree, the most important predictor of whether a customer will purchase Citrus Hill or Minute Maid orange juice is loyalty to Citrus Hill. The split criterion for the first non-root / internal node / the first pair of branches is $LoyalCH$. The split criteria for the second internal node in the second echelon is also $LoyalCH$.

(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?

```
vector_of_predicted_purchases <- predict(full_tree, testing_data, type = "class")
vector_of_actual_purchases <- testing_data$Purchase
confusion_matrix <- table(vector_of_predicted_purchases, vector_of_actual_purchases)
confusion_matrix
```

```
#                                vector_of_actual_purchases
# vector_of_predicted_purchases  CH  MM
#                             CH 144  27
#                             MM  29  70
```

```
number_of_purchases_of_Citrus_Hill_orange_juice_predicted_correctly <- 144
number_of_purchases_of_Minute_Maid_orange_juice_predicted_correctly <- 70
number_of_purchases_predicted_correctly <-
    number_of_purchases_of_Citrus_Hill_orange_juice_predicted_correctly +
    number_of_purchases_of_Minute_Maid_orange_juice_predicted_correctly
number_of_purchases <- nrow(testing_data)
test_accuracy <- number_of_purchases_predicted_correctly / number_of_purchases
test_error_rate <- 1 - test_accuracy
test_error_rate
```

```
# [1] 0.2074074
```

The test error rate is about 0.207.

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

```
object_of_types_prune_and_tree_sequence <-
    cv.tree(full_tree, FUN = prune.misclass)
vector_of_sizes <- object_of_types_prune_and_tree_sequence$size
vector_of_numbers_of_errors <- object_of_types_prune_and_tree_sequence$dev
minimum_number_of_errors <- min(vector_of_numbers_of_errors)
index_of_minimum_number_of_errors <- which.min(vector_of_numbers_of_errors)
optimal_size <-
    vector_of_sizes[index_of_minimum_number_of_errors]
optimal_size
```

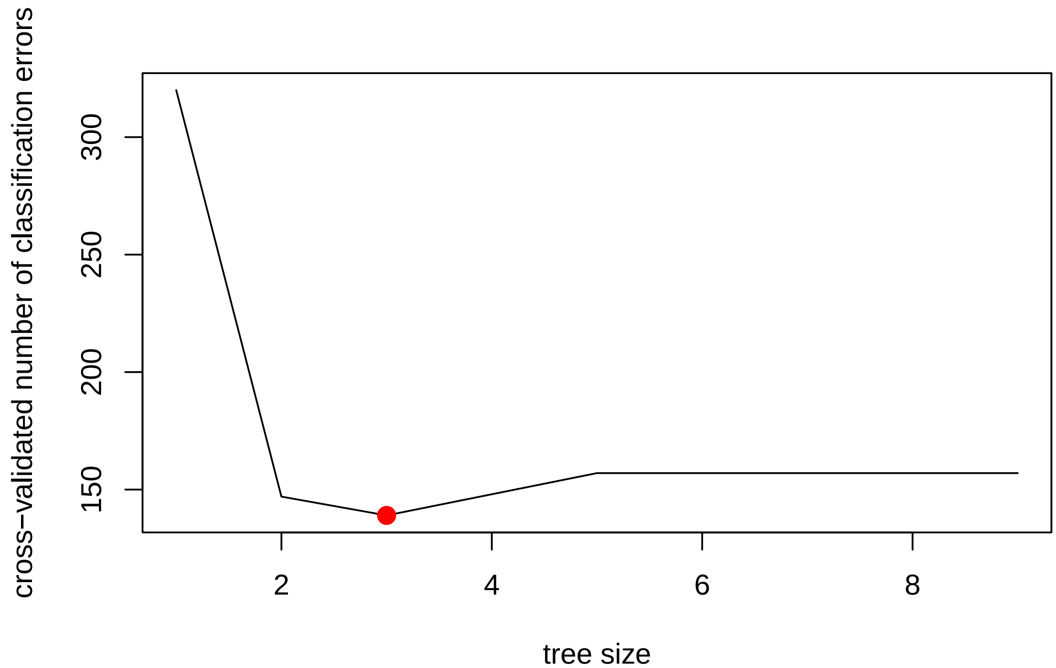
```
# [1] 3
```

The optimal tree size is 3.

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(
    vector_of_sizes,
    vector_of_numbers_of_errors,
    type = "l",
    xlab = "tree size",
    ylab = "cross-validated number of classification errors"
)
points(
```

```
    optimal_size,
    minimum_number_of_errors,
    col = "red",
    cex = 2,
    pch = 20
)
```

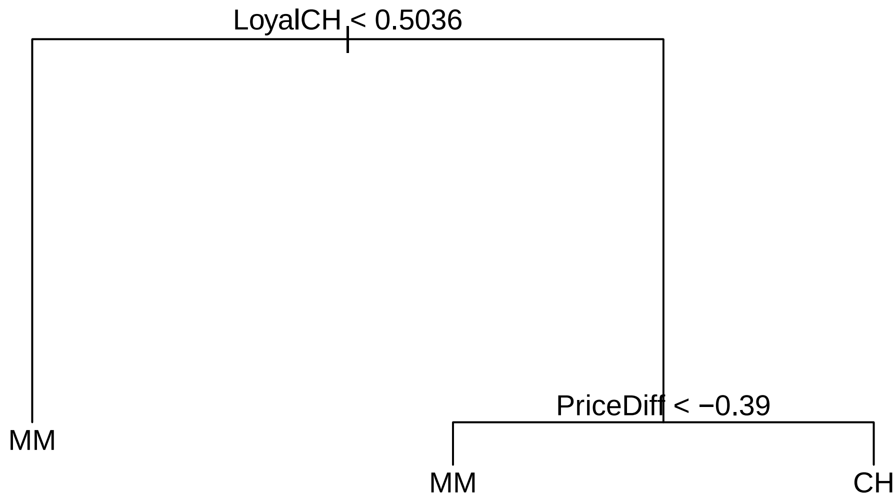


(h) Which tree size corresponds to the lowest cross-validated classification error rate?

A tree size of 3 corresponds to the lowest cross-validated classification number of errors.

(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.

```
pruned_tree <- prune.tree(full_tree, best = optimal_size)
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```



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

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