TP 1_Classification and Regression Trees

Group 14

2/24/2021

Exercise 9: Using Tree Classification to Predict Orange Juice Sales

Exercise 9 uses tree classification to predict whether a customer will by Citrus Hill (CH) or Minute Maid (MM) orange juice. The exercise involves making the model, running cross validation, and then using a pruned model.

Load the Data

The OJ data contains 1070 purchases where the customer either purchased Citrus Hill or Minute Maid Orange Juice. There are 17 characteristics of the customer and product recorded in the data. The data set is already cleaned as shown in the 'str' function, so no data cleaning is required.

```
library(ISLR)
data(OJ)
str(OJ)
```

```
## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
##  $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : num  1 1 1 1 0 0 0 0 0 0 ...
```

Split the Data

Instead of using the book's method, the data is split using the caret package. The training data contains 80% of the data while the remaining 20% is saved for testing.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.3
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
set.seed(420)
d_divide<-createDataPartition(OJ$Purchase,p=.8,list = FALSE)
train<-OJ[d_divide,]
test<-OJ[-d_divide,]
```

Fit the Model

To fit the tree classification model, use the tree library. As seen in the model summary, there are 10 terminal nodes and a training error rate of 15.87%.

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.0.3
```

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

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "SalePriceMM" "PriceDiff"
## Number of terminal nodes:  10
## Residual mean deviance:  0.7509 = 636 / 847
## Misclassification error rate: 0.1587 = 136 / 857
```

Display Detailed Results

Printing the model shows the break down of each node in the tree. In order it displays the split criterion, the number of observations in that branch, the deviance, the overall prediction for the branch, and the fraction of observations that take on the prediction. It also denotes if a node is a terminal node or not.

```
model
```

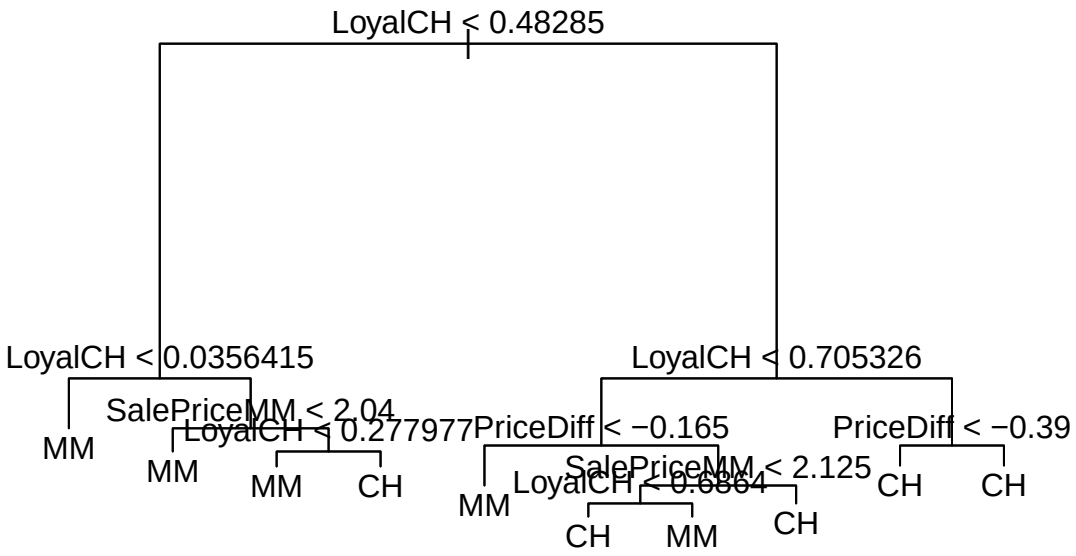
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 857 1146.00 CH ( 0.61027 0.38973 )
##    2) LoyalCH < 0.48285 331  377.10 MM ( 0.25680 0.74320 )
##      4) LoyalCH < 0.0356415 65    0.00 MM ( 0.00000 1.00000 ) *
##      5) LoyalCH > 0.0356415 266  333.30 MM ( 0.31955 0.68045 )
##       10) SalePriceMM < 2.04 144  144.70 MM ( 0.20139 0.79861 ) *
##       11) SalePriceMM > 2.04 122  168.30 MM ( 0.45902 0.54098 )
##         22) LoyalCH < 0.277977 54   63.81 MM ( 0.27778 0.72222 ) *
##         23) LoyalCH > 0.277977 68   91.36 CH ( 0.60294 0.39706 ) *
##    3) LoyalCH > 0.48285 526  475.10 CH ( 0.83270 0.16730 )
##      6) LoyalCH < 0.705326 214  267.60 CH ( 0.68224 0.31776 )
##       12) PriceDiff < -0.165 30   30.02 MM ( 0.20000 0.80000 ) *
##       13) PriceDiff > -0.165 184  202.40 CH ( 0.76087 0.23913 )
##         26) SalePriceMM < 2.125 115  145.90 CH ( 0.66957 0.33043 )
##           52) LoyalCH < 0.6864 110  134.40 CH ( 0.70000 0.30000 ) *
##           53) LoyalCH > 0.6864 5    0.00 MM ( 0.00000 1.00000 ) *
##         27) SalePriceMM > 2.125 69   40.77 CH ( 0.91304 0.08696 ) *
```

```
##      7) LoyalCH > 0.705326 312  148.60 CH ( 0.93590 0.06410 )
##       14) PriceDiff < -0.39 13   17.94 CH ( 0.53846 0.46154 ) *
##       15) PriceDiff > -0.39 299  113.10 CH ( 0.95318 0.04682 ) *
```

Plot the Tree

Plotting the tree shows a visualization of the different branches in the tree. It does not go as in-depth as printing the model, but it does show the split criteria.

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



Use the Test Data and Create a Confusion Matrix

Using the test data to make a prediction yields a test error rate of 16.9%. This is slightly higher than the training error rate, which is generally expected. There were 36 misclassifications and 177 valid classifications.

```
prediction<-predict(model,test,type = 'class')
table(prediction, test$Purchase)
```

```
##
## prediction  CH  MM
##         CH 120  26
##         MM  10  57
```

```
cat("Test Error Rate:",toString((26+10)/length(test$Purchase)))
```

```
## Test Error Rate: 0.169014084507042
```

Use Cross Validation to Find Optimal Tree Size

To cross validate the tree model, the cv.tree function is used. Setting FUN to prune.misclass ensures that the training error rate is used as the criteria for model improvement. Displaying the results of the model shows the deviation at each number of terminal nodes.

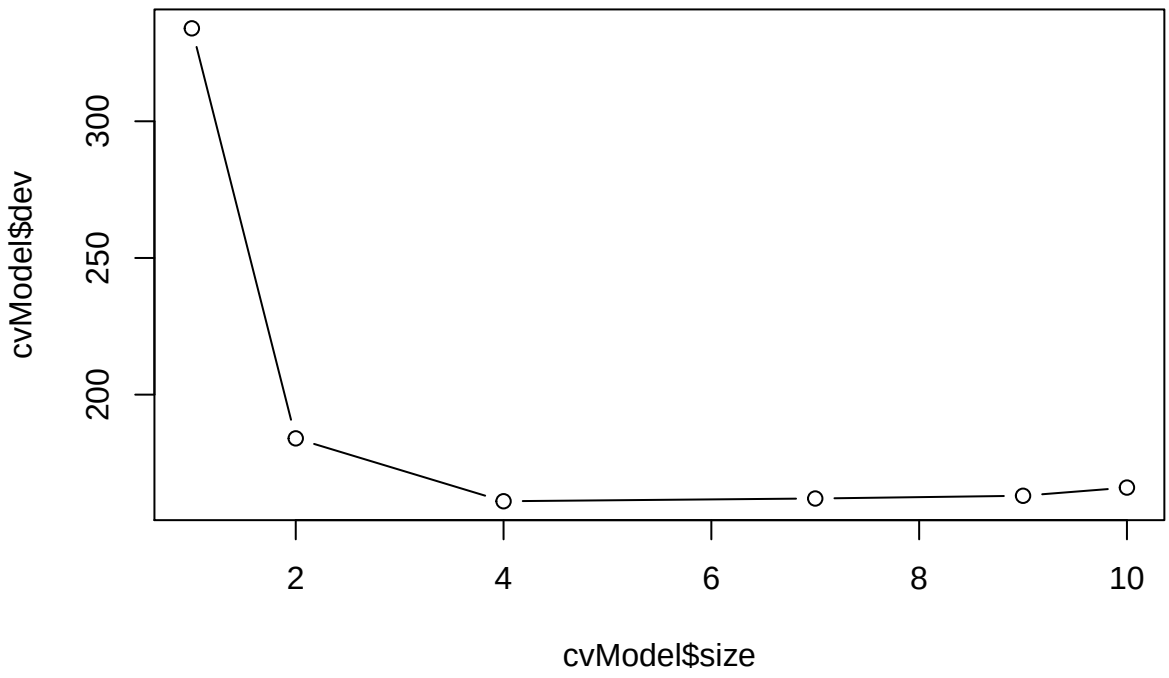
```
cvModel<-cv.tree(model, FUN = prune.misclass)
cvModel
```

```
## $size
## [1] 10  9  7  4  2  1
##
## $dev
## [1] 166 163 162 161 184 334
##
## $k
## [1]       -Inf   0.000000   2.500000   4.666667   9.000000 161.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Plot the CV Results

Plotting the cross validation results shows that using 4 terminal nodes yields the smallest deviation/error rate for the model. This differs from our original model that used 10 terminal nodes. Not only is the error rate lower, less nodes and branches means there is a lower chance of overfitting.

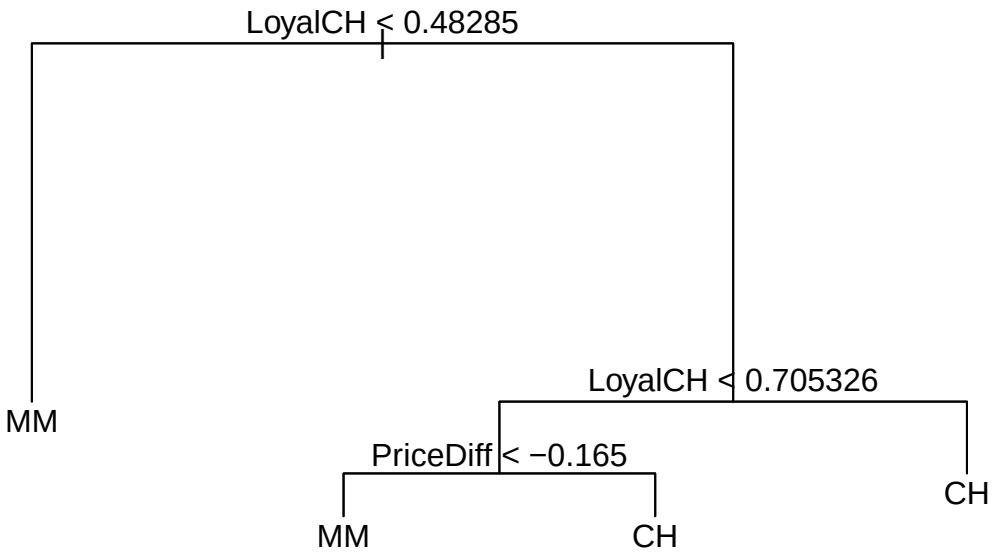
```
plot(cvModel$size,cvModel$dev, type = 'b')
```



Make a Model Using the Best Pruned Tree

Using the number of terminal nodes found above, a pruned model is created using the prune.misclass function. The plot shows the branches for a 4 terminal node model.

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



Find the New Test Error Rate

Using the pruned model yields different results from the original model. The training error rate was slightly higher at 18.09% as opposed to the original 15.87%. The test error rate fell by 2.82% from the original test error rate of 16.9% to the pruned test error rate of 14.08%.

```
summary(pruneModel)
```

```
##
## Classification tree:
## snip.tree(tree = model, nodes = c(7L, 13L, 2L))
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  4
## Residual mean deviance:  0.8888 = 758.2 / 853
## Misclassification error rate: 0.1809 = 155 / 857
```

```
prunePred<-predict(pruneModel,test,type = 'class')
table(prunePred, test$Purchase)
```

```
##
## prunePred  CH  MM
##        CH 117  17
##        MM  13  66
```

```
cat("Test Error Rate:",toString((13+17)/length(test$Purchase)))
```

```
## Test Error Rate: 0.140845070422535
```