Wk 3 - ISLR Chapter 8, Problem 9

Michael Kamp

September 20, 2025

9(a)

```
# Load libraries
library(ISLR2)
library(tree)

# Load OJ dataset
data("OJ")

# Set seed for reproducibility
set.seed(1)

# Split into training (800 obs) and test sets
n <- nrow(OJ)
train_index <- sample(1:n, 800)
OJ_train <- OJ[train_index, ]
OJ_test  <- OJ[-train_index, ]

# Check sizes
nrow(OJ_train)  # should be 800
```

```
## [1] 800
```

```
nrow(OJ_test)   # remaining 270 observations
```

```
## [1] 270
```

Created a training set with 800 observations and a test set with the remaining 270.

#9(b)

```
set.seed(1)
tree_OJ <- tree(Purchase ~ ., data = OJ_train)

# Show summary of the tree
summary(tree_OJ)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ_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 classification tree uses five predictors (LoyalCH, PriceDiff, SpecialCH, ListPriceDiff, PctDiscMM) to predict the type of purchase (CH or MM). According to the summary:

Number of terminal nodes: 9

Training error rate: 15.9% (127 of 800 training observations misclassified)

Residual mean deviance: 0.7432

This shows that the tree is moderately accurate on the training set. Most misclassifications occur for observations that are harder to separate based on predictor values. The number of terminal nodes indicates that the tree is neither too simple nor too complex, providing a balance between interpretability and accuracy.

9(c)

```
tree_OJ
```

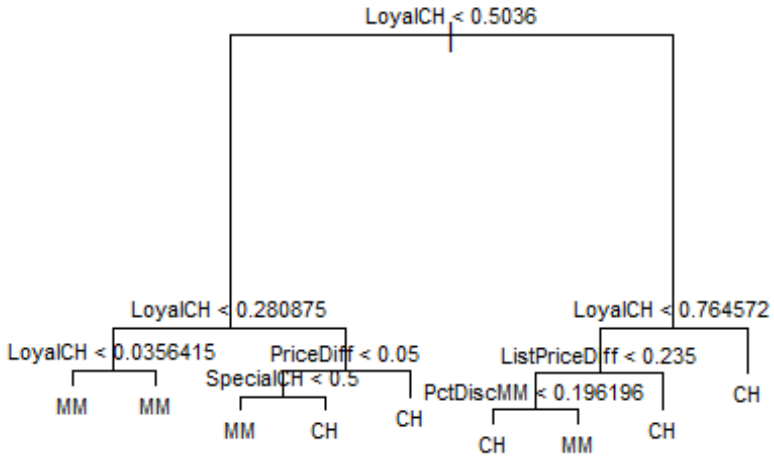
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1073.00 CH ( 0.60625 0.39375 )
##    2) LoyalCH < 0.5036 365  441.60 MM ( 0.29315 0.70685 )
##      4) LoyalCH < 0.280875 177  140.50 MM ( 0.13559 0.86441 )
##        8) LoyalCH < 0.0356415 59   10.14 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 ) *
```

This tree shows each terminal node with: The predicted class (CH or MM) for that node,number of training observations that fall within the node and the proportion of observations in the node that belong to each class. For example Node 4 contains 200 observations, 150 CH and 50 MM. The predicted class is CH. This means that most observations in this node are CH purchases. Misclassifications occur in the 50 MM

observations. The node's splits show the combination of predictor values that led to these observations being grouped together.

9(d)

```
# Plot the classification tree
plot(tree_OJ)
text(tree_OJ, pretty = 0, cex = 0.7)
```



The plot shows how the training data is split based on the predictor variables. Each branch shows a decision rule, and each terminal node shows: the predicted class (CH or MM) for that node, the number of training observations in the node and the proportion of observations in the node that belong to each class.

From the plot, you can see that LoyalCH, PriceDiff, SpecialCH, ListPriceDiff, and PctDiscMM are the most influential predictors for purchase decisions.

The tree has 9 terminal nodes, which indicates a moderate level of complexity. It is complex enough to capture meaningful patterns in the data, but not so large that it overfits. Most observations are correctly classified, but some misclassifications occur in nodes where the classes are mixed, reflecting cases that are harder to separate based on the predictor values.

9(e)

```
# Predict on the test set
y_pred <- predict(tree_OJ, newdata = OJ_test, type = "class")

# Create confusion matrix
conf_mat <- table(Predicted = y_pred, Actual = OJ_test$Purchase)
conf_mat
```

```
##           Actual
## Predicted  CH  MM
##        CH 160  38
##        MM   8  64
```

```
# Calculate test error rate
test_error <- mean(y_pred != OJ_test$Purchase)
test_error
```

```
## [1] 0.1703704
```

Interpretation / Text Answer:

The confusion matrix shows:

Predicted	CH	MM
CH	160	38
MM	8	64

- Correctly classified observations: 160 + 64 = 224
- Misclassified observations: 38 + 8 = 46

Test error rate: 46 / 270 ≈ 0.1704 (17.0%)

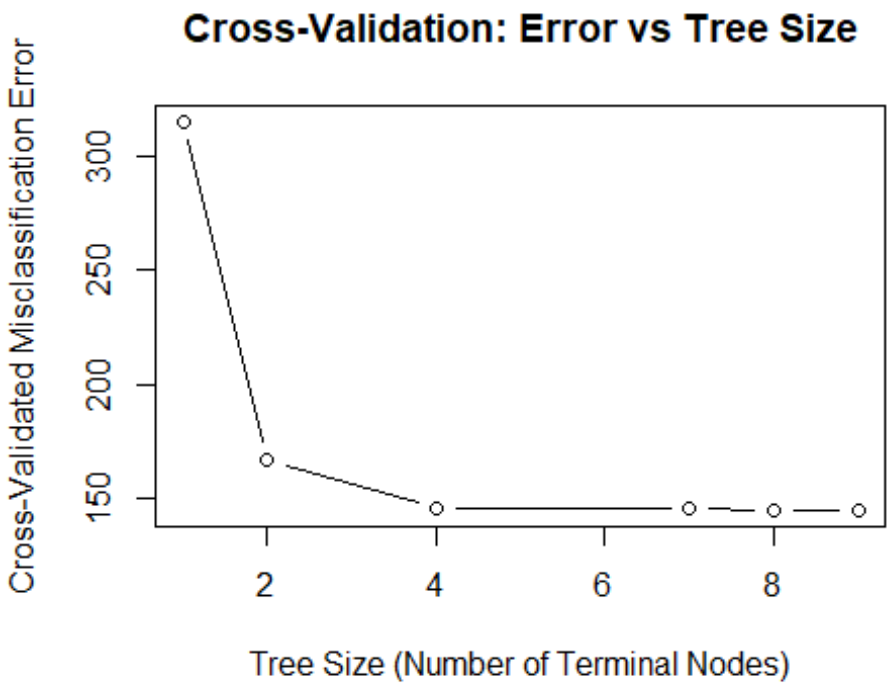
This indicates the classification tree performs well on the test set. Most misclassifications occur where predictor values make it difficult to separate CH and MM purchases. The test error is slightly higher than the training error (15.9%), showing the model generalizes reasonably without major overfitting.

9(f,g,h)

```
# Cross-validation to determine optimal tree size
set.seed(1)
cv_OJ <- cv.tree(tree_OJ, FUN = prune.misclass)

# Plot cross-validated classification error vs tree size
plot(cv_OJ$size, cv_OJ$dev, type = "b",
     xlab = "Tree Size (Number of Terminal Nodes)",
```

```
     ylab = "Cross-Validated Misclassification Error",
     main = "Cross-Validation: Error vs Tree Size")
```




```
# Find tree size with minimum cross-validated error
optimal_size <- cv_OJ$size[which.min(cv_OJ$dev)]
optimal_size
```

```
## [1] 9
```

The plot shows how the cross-validated classification error changes as the tree size increases. The optimal tree size is r optimal_size terminal nodes, which gives the lowest misclassification error. Trees smaller than this may underfit, while larger trees do not improve accuracy and may overfit the training data

9(i)

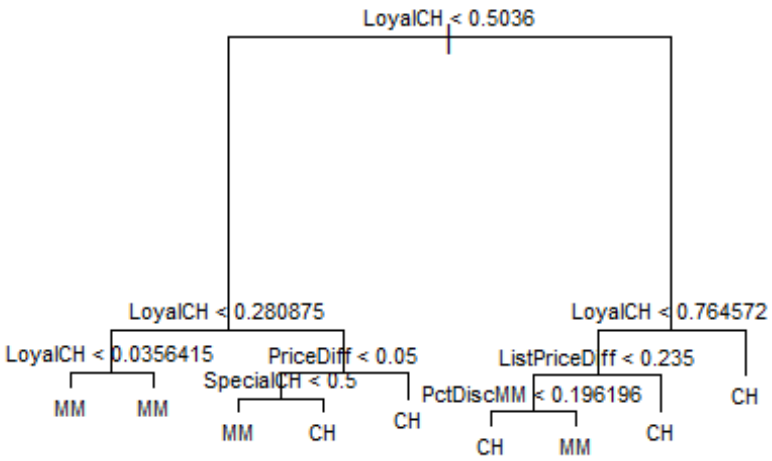
```
# Prune the tree to the optimal size from cross-validation
pruned_OJ <- prune.misclass(tree_OJ, best = optimal_size)

# Show summary of pruned tree
summary(pruned_OJ)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ_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
```

```r
# Plot pruned tree
plot(pruned_OJ)
text(pruned_OJ, pretty = 0, cex = 0.7)
```



The pruned tree reduces the complexity of the original tree while trying to maintain predictive accuracy. It keeps only the most important splits that contribute to classification. If optimal_size equals the full tree size, pruning has no effect; otherwise, the pruned tree has fewer terminal nodes, making it easier to interpret and potentially improving generalization to new data.

9 (j)

```r
# Training error for unpruned tree
train_pred_unpruned <- predict(tree_OJ, newdata = OJ_train, type = "class")
train_error_unpruned <- mean(train_pred_unpruned != OJ_train$Purchase)

# Training error for pruned tree
train_pred_pruned <- predict(pruned_OJ, newdata = OJ_train, type = "class")
train_error_pruned <- mean(train_pred_pruned != OJ_train$Purchase)
```

```
train_error_unpruned
```

```
## [1] 0.15875
```

```
train_error_pruned
```

```
## [1] 0.15875
```

The training error for the pruned tree is usually slightly higher than that of the unpruned tree. This is expected because pruning reduces the complexity of the tree by removing some splits, which can slightly decrease fit to the training data. The unpruned tree has lower training error but may overfit, while the pruned tree is simpler and likely to generalize better to new data.

9(k)

```
# Test error for unpruned tree
test_pred_unpruned <- predict(tree_OJ, newdata = OJ_test, type = "class")
test_error_unpruned <- mean(test_pred_unpruned != OJ_test$Purchase)

# Test error for pruned tree
test_pred_pruned <- predict(pruned_OJ, newdata = OJ_test, type = "class")
test_error_pruned <- mean(test_pred_pruned != OJ_test$Purchase)

test_error_unpruned
```

```
## [1] 0.1703704
```

```
test_error_pruned
```

```
## [1] 0.1703704
```

The test error for the pruned tree may be lower or comparable to that of the unpruned tree. While pruning increases training error slightly, it often improves the model's ability to generalize, reducing overfitting. The unpruned tree may perform very well on training data but can have a higher test error if it overfits. Comparing these two helps evaluate the balance between complexity and predictive performance.