Practical 03: Decision Trees

Alex Mounsey

Classification (Decision) Trees

Classification trees involve sequential binary splitting of predictors' values, one predictor at a time. After each split, the training error is decreased. A new observation x_0 is assigned to the most commonly occuring class of training observations in the region to which it belongs.

Splitting Criteria

Classification error is usually not sufficiently sensitive for tree-growing. Thus, in practice other measures are used: either the Gini Index or Cross-Entropy.

Gini Index

Gini-Index measures the 'diversity' of classes in a given tree node:

$$G = p_1(1 - p_1) + p_2(1 - p_2) + ... + p_g(1 - p_g)$$

where G is the number of classes and $p_1...p_g$ are the proportions of observations for each class. A small value of G indicates that a node contains predominantly observations from a single class.

Cross-Entropy

$$D = -p_1 log p_1 - p_2 log p_2 - ... - p_g log p_g$$

D has similar properties and interpretation as G.

Tree Pruning

Classification trees typically overfit the data. A tree can be 'pruned' - that is we can restrict the number of leaves (terminal nodes) to a certain number (k). This number k is chosen to minimise the test error rate. The test error rate is usually estimated using the cross-validation approach.

Advantages and Disadvantages of Decision Trees

Advantages

- Classification trees can be easily understood and applied by non-experts
- Trees can be easily constructed in the presence of qualitative (non-numerical) predictors, such as colour, nationality, etc.

Disadvantages

- Trees are often less accurate in predicting classes than other classification methods
- As a non-parametric method, trees suffer from high variance, which means that if we split the training data into two parts at random and fit a decision tree to both halves, the results could be quite different.

Families of Trees

Families of trees are used to improve the predictive power of a decision tree. They are based on the idea that the collective wisdom of many people is greater than the wisdom of one person, only here this idea is applied to classifiers and not people.

For a given classification task, we construct many trees. Then, to classify a new observation x_0, we apply each tree and obtain a predicted class from each of them. The final predicted class for x_0 is the majority vote, that is the most commonly occurring class among the predictions.

Bagging

How to construct many trees for one classification problem and one data set?

1. We take repeated samples from the (single) training data. Each sample is of size n and is taken with replacement, which means that the same observation can occur more than once in the sample. This sampling procedure is called bootstrapping.
2. We obtain a certain number, say B, of different training sets
3. Then, we construct a classification tree for each of the bootstrapped training sets.

This method is called bagging.

Random Forests

Random forests provide an improvement over bagged trees by way of a small tweak that results in a more 'diverse' family of trees.

1. As in bagging, we build a number of decision trees on B bootstrapped training samples
2. When building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
3. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ - that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

Bagging is a special case of a random forest where $m = p$

Task A

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

```
set.seed(1)

df <- OJ

df.subset <- sample(1:nrow(df), size = 800)  # randomly select 800 values

df.train <- df[df.subset, ]  # training set
df.test <- df[-df.subset, ]  # test set
```

Task B

1. Fit a tree to the training data, with `Purchase` as the response *(class label)* and the other variables as predictors
2. Use the `summary()` function to produce summary statistics for the tree. What is the training error rate? How many terminal nodes does the tree have?

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

```
summary(df.tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = df.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 training error is 15.1% and the tree consists of 9 terminal nodes.

The deviance reported in the output above is calculated using the following formula:

$$-2\sum_{m}\sum_{k} n_{mk} log\hat{p}_{mk}$$

where n_{mk} is the number of observations in the m^{th} terminal node that belong to the k^{th} class and $\hat{p}_{mk}$ is the fraction of observations in the m^{th} terminal node that belong to the k^{th} class.

A small deviance indicates a tree that provides a good fit to the (training) data. The residual mean deviance reported is simply the deviance divided by n minus the number of leaves.

Task C

1. Type in the name of the tree to get a detailed text output. Pick one of the terminal nodes and interpret the information displayed

```
df.tree
```

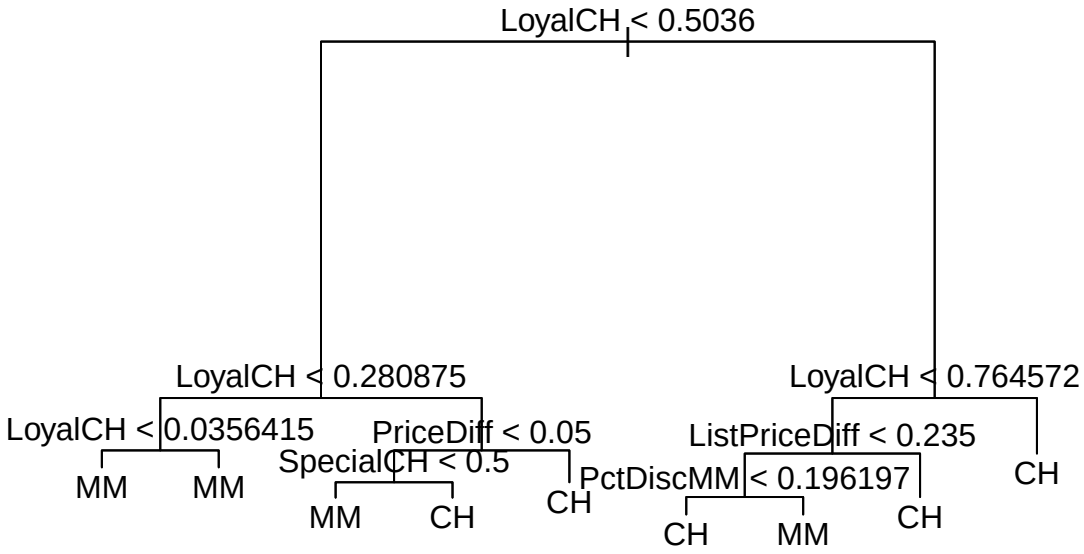
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1073.00 CH ( 0.60625 0.39375 )
##    2) LoyalCH < 0.5036 365  441.60 MM ( 0.29315 0.70685 )
##      4) LoyalCH < 0.280875 177  140.50 MM ( 0.13559 0.86441 )
##        8) LoyalCH < 0.0356415 59   10.14 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 < 0.196197 55   73.14 CH ( 0.61818 0.38182 ) *
##         25) PctDiscMM > 0.196197 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 ) *
```

When we enter the name of the tree object, R prints the output corresponding to each node of the tree. It displays the split criterion (e.g. $LoyalCH < 0.280875$), the number of observations in that node, the deviance, the overall prediction for the node (`CH` and `MM`), and the fraction of observations in that node that take on values of `CH` and `MM`, respectively. Terminal nodes (leaves) are indicated with asterisks.

Task D

1. Display the tree as a graph

```
plot(df.tree)
text(df.tree, pretty = 0)
```



The most important indicator of `Purchase` appears to be `LoyalCH`, since the first branch differentiates high and low customer loyalty.

Task E

1. 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?

In order to evaluate the performance of a classification tree, we estimate the test error. The `predict()` function can be used for this purpose. The argument `type = 'class'` instructs R to return class predictions (predicted categories).

```
df.test.y <- df.test$Purchase  # class labels from test data

# make predictions on the test data using the tree
df.tree.pred <- predict(df.tree, df.test, type = 'class')

# create confusion matrix of predictions
results <- table(df.tree.pred, df.test.y)
results
```

```
##             df.test.y
## df.tree.pred  CH  MM
##           CH 160  38
##           MM   8  64
```

```
# calculate the test error
(results[1,2] + results[2,1]) / sum(results)
```

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

Task F

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

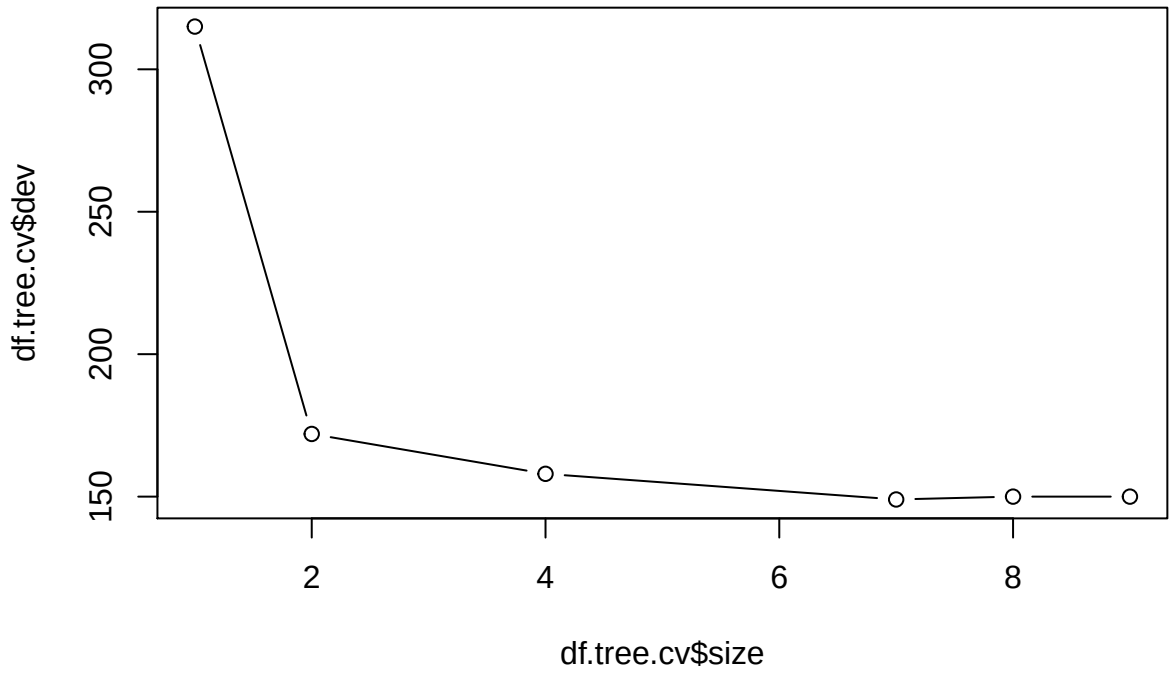
```
df.tree.cv <- cv.tree(df.tree, FUN = prune.misclass)
df.tree.cv
```

```
## $size
## [1] 9 8 7 4 2 1
##
## $dev
## [1] 150 150 149 158 172 315
##
## $k
## [1]       -Inf   0.000000   3.000000   4.333333  10.500000 151.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Task G

1. Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis
2. Which tree size corresponds to the lowest cross-validated classification error rate?

```
plot(df.tree.cv$size, df.tree.cv$dev, type = 'b')
```



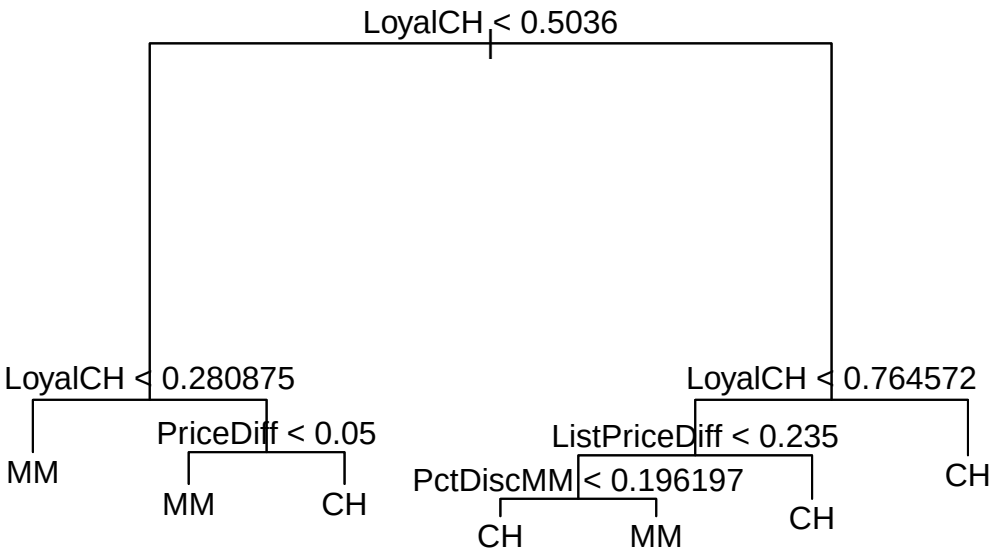
A tree of size 7 appears to produce the lowest cross-validated classification error rate.

Task H

1. Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to the selection of a pruned tree, then create a pruned tree with five terminal nodes

```
df.tree.pruned <- prune.misclass(df.tree, best = 7)

plot(df.tree.pruned)
text(df.tree.pruned, pretty = 0)
```



Task I

1. Compare the training error rates between the pruned and non-pruned trees, which is higher?

```
summary(df.tree.pruned)
```

```
##
## Classification tree:
## snip.tree(tree = df.tree, 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
```

Task J

1. Compare the test error rates between the pruned and non-pruned trees, which is higher?

```
df.tree.pruned.pred <- predict(df.tree.pruned, df.test, type = 'class')
results <- table(df.tree.pruned.pred, df.test.y)
results
```

```
##                     df.test.y
## df.tree.pruned.pred  CH  MM
##                  CH 160  36
##                  MM   8  66
```

```r
(results[1,2] + results[2,1]) / sum(results)
```

```
## [1] 0.162963
```