Classification Trees

Maja Lie

14/01/2020

Classification Trees

Classification is a supervised learning method, meaning we know what categories we are working with before creating our model. It is used for qualitative responses. One method we will learn is decision trees. This is useful for analysing large data sets or data sets with many predictors. The idea is to partition our feature space into a set of rectangles and then assign each rectangle to a category. Our feature space is just the set of all our input variables, i.e. predictors.

We divide the space by binary recursive splitting. That means we will start by splitting our feature space by one predictor first, then continue to split by other predictors until we have a full grown tree.

If you are interested in how we choose to split our tree first, there is a lot of theory you can read. Basically, the idea is to choose the feature that has the biggest information gain or lowest conditional entropy. Entropy is a measure for uncertainty, so maximising information gain means decreasing uncertainty in our model.

In the end, we end up with a tree with many branches which end in terminal nodes. The nodes represent the label of the class. So, given a new data point, we should end up with a class label for it by following the direction of the branches. The label for terminal nodes are determined by majority vote; if most of the data points we are using to build our model are labeled A in a node, then the node gets labelled A.

To begin, we need to install the packages tree and ISLR. We will be using the data set Carseats built into R.

```
# Divide the response variable into qualitative categories

attach(Carseats)
High = ifelse(Sales > 8, "Yes", "No") # sales greater than 8 are high sales

# Append High to the data frame
Carseats = data.frame(Carseats, High)
head(Carseats)
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age
## 1  9.50       138     73          11        276   120       Bad  42
## 2 11.22       111     48          16        260    83      Good  65
## 3 10.06       113     35          10        269    80    Medium  59
## 4  7.40       117    100           4        466    97    Medium  55
## 5  4.15       141     64           3        340   128       Bad  38
## 6 10.81       124    113          13        501    72       Bad  78
##   Education Urban  US High
## 1        17   Yes Yes  Yes
## 2        10   Yes Yes  Yes
## 3        12   Yes Yes  Yes
## 4        14   Yes Yes   No
## 5        13   Yes  No   No
## 6        16    No Yes  Yes
```

Creating the Tree

We use the tree function from the tree package. The first argument is an R formula relating the response variable and the predictors. The .- means to include all predictors except for Sales. The second argument is the data source.

The summary gives us the number of variables in the tree, the number of terminal nodes, and the misclass error rate. It also gives us something called mean deviance; the smaller the mean deviance, the better our model is.

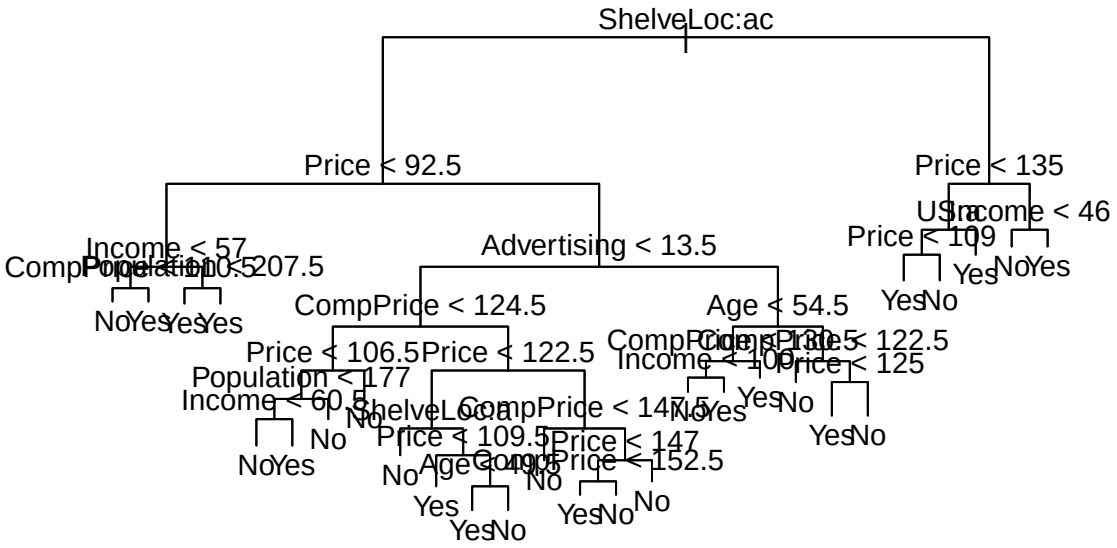
```
treeCarseats = tree(formula = High ~.-Sales, data = Carseats)
summary(treeCarseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

Visualising the Tree

We can plot the tree and add text labels to each branch to see how the tree was split. Traditionally when reading a decision tree, if the label applies to a data point we go down the branch to the left.

```
plot(treeCarseats); text(treeCarseats, cex = 0.9)
```



Looking at our tree, the first split is done by location. This indicates that location is an important predictor of whether or not there are high sales.

Evaluating Performance

To evaluate the performance of the tree, we split the data into training and testing data. We're going to recreate the tree with just the training data and then evaluate the model with the testing data.

```
set.seed(2)
train = sample(1:nrow(Carseats), 200) # selects half of the data for training
CarseatsTest = Carseats[-train,]
HighTest = High[-train]

treeCarseats = tree(High~.-Sales, Carseats, subset = train)
treePred = predict(treeCarseats, CarseatsTest, type="class")

table(treePred, HighTest)
```

```
##         HighTest
## treePred  No Yes
##      No  104  33
##      Yes  13  50
```

```
# tabulates which individuals are correctly predicted, HighTest = true labels
(104+50)/200 # correct predictions %
```

```
## [1] 0.77
```

Tree Pruning

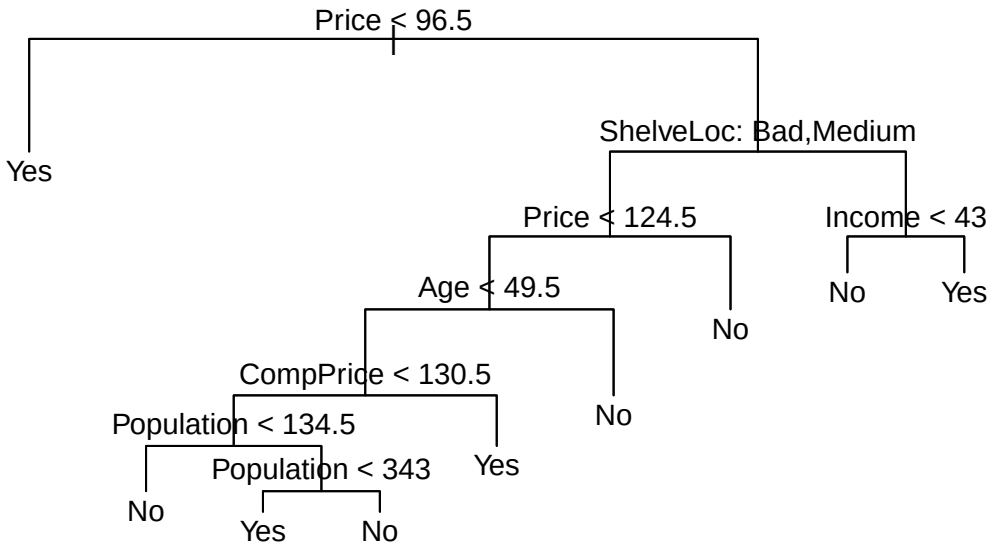
Once we have a full grown tree, we may want to prune back some branches so that our tree is able to generalise well to new data. Otherwise, we run the risk of overfitting our model.

One way of pruning is by k-fold cross validation.

```
set.seed(3) # set randomization seed for k-fold validation
cvCarseats = cv.tree(treeCarseats, FUN = prune.misclass)
cvCarseats
```

```
## $size
## [1] 21 19 14  9  8  5  3  2  1
##
## $dev
## [1] 74 76 81 81 75 77 78 85 81
##
## $k
## [1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
pruneCarseats = prune.misclass(treeCarseats, best = 9) # choose terminal nodes that minimize deviance
plot(pruneCarseats)
text(pruneCarseats, cex = 0.9, pretty = 0)
```



Testing the Pruned Tree

```
treePred = predict(pruneCarseats, CarseatsTest, type="class")
table(treePred, HighTest)
```

```
##         HighTest
## treePred No Yes
##      No  97  25
##      Yes 20  58
```

```
(97+58)/200
```

```
## [1] 0.775
```