# Tree-Based Methods
# Classification Trees

November 3, 2017

## Classification Trees

- A classification tree is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- Recall that for a regression tree, the predicted response for an observation is given by the mean response of the training observations that belong to the same terminal node.
- In contrast, for a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
- In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region.
- However, in the classification setting, RSS cannot be used as a criterion for making the binary splits.
- A natural alternative to RSS is the *classification error rate*.
- The classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class.

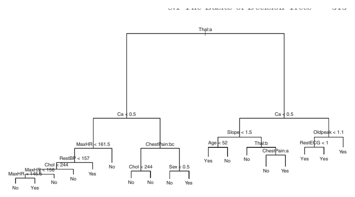

Figure 1: The unpruned tree.

Figure 1 shows an example on the **Heart data** set. These data contain a binary outcome HD for 303 patients who presented with chest pain. An outcome value of Yes indicates the presence of heart disease based on an angiographic test, while No means no heart disease. There are 13 predictors including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.

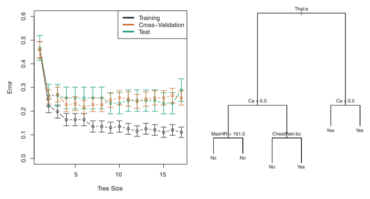

Figure 2: Left: Cross -validation error, training, and test error, for different sizes of the pruned tree. Bottom Right: The pruned tree corresponding to the minimal cross-validation error.

## Fitting Classification Trees

The **tree library** is used to construct classification and regression trees.

### Creat the data set

We first use classification trees to analyze the Carseats data set. In these data, Sales is a continuous variable, and so we begin by recoding it as a binary variable. We use the **ifelse()** function to create a variable, called High, which takes on a value of Yes if the Sales variable exceeds 8, and takes on a value of No otherwise.

```
> library (ISLR)
> attach (Carseats )
> High=ifelse (Sales <=8," No"," Yes ")
> Carseats =data.frame(Carseats ,High)
```

Finally, we use the data.frame() function to merge High with the rest of the Carseats data.
We now use the **tree() function** to fit a classification tree in order to predict High using all variables but Sales.

```
> tree.carseats =tree(High.-Sales ,Carseats ) <---- Using All Data
> summary (tree.carseats )
Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "US"          "Income"      "CompPrice"   "Population"  "Adver
```

```
[8] "Age"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

The summary() function lists the variables that are used as internal nodes in the tree, the number of terminal nodes, and the (training) error rate. We see that the training error rate is 9 %.
A **small deviance** indicates a tree that provides a good fit to the (training) data.
One of the most attractive properties of trees is that they can be graphically displayed. We use the plot() function to display the tree structure, and the text() function to display the node labels. The argument pretty=0 instructs R to include the category names for any qualitative predictors, rather than simply displaying a letter for each category.

```
> plot(tree.carseats )
> text(tree.carseats ,pretty =0)
```

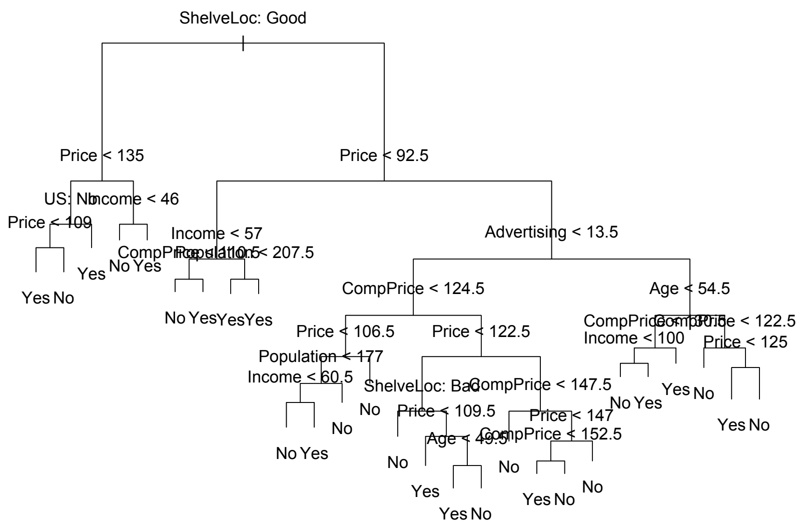

Figure 3: Decision Tree

If we just type the name of the tree object, R prints output corresponding to each branch of the tree. R displays the split criterion (e.g. Price < 92.5), the number of observations in that branch, the deviance, the overall prediction for the branch (Yes or No), and the fraction of observations in that branch that take on values of Yes and No. Branches that lead to terminal nodes are indicated using asterisks.

```
> tree.carseats
node), split, n, deviance, yval, (yprob)
* denotes terminal node

1) root 400 541.500 No ( 0.41000 0.59000 )
2) ShelveLoc: Good 85  90.330  Yes  ( 0.77647 0.22353 )
4) Price < 135 68  49.260  Yes  ( 0.88235 0.11765 )
```

```
8) US: No 17  22.070  Yes  ( 0.64706 0.35294 )
16) Price < 109 8   0.000  Yes  ( 1.00000 0.00000 ) *
```

**Properly evaluate the performance of a classification tree**

In order to properly evaluate the performance of a classification tree on these data, we must estimate the test error rather than simply computing the training error. We split the observations into a training set and a test set, build the tree using the training set, and evaluate its performance on the test data. The predict() function can be used for this purpose. In the case of a classification tree, the argument type=”class” instructs R to return the actual class prediction.

```
> set.seed (2)
> train=sample (1: nrow(Carseats ), 200)
> Carseats .test=Carseats [-train ,]
> High.test=High[-train ]
> tree.carseats =tree(High.-Sales ,Carseats ,subset =train )
> tree.pred=predict (tree.carseats ,Carseats .test ,type =" class ")
> table(tree.pred ,High.test)
          High.test
tree.pred No  Yes
      No  86  27
      Yes 30 57
> (86+57) /200
[1] 0.715
```

This approach leads to correct predictions for around 71.5% of the locations in the test data set. Next, we consider whether pruning the tree might lead to improved results. The function cv.tree() performs cross-validation in order to cv.tree() determine the optimal level of tree complexity; cost complexity pruning is used in order to select a sequence of trees for consideration. We use the argument FUN=prune.misclass in order to indicate that we want the classification error rate to guide the cross-validation and pruning process, rather than the default for the cv.tree() function, which is deviance. The cv.tree() function reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate and the value of the cost-complexity parameter used.

```
> set.seed (3)
> cv.carseats =cv.tree(tree.carseats ,FUN=prune.misclass )
> names(cv.carseats )
[1] "size" "dev " "k" "method "
> cv.carseats
$size
[1] 19 17 14 13 9 7 3 2 1
$dev
[1] 55 55 53 52 50 56 69 65 80
$k
[1] -Inf 0.0000000 0.6666667 1.0000000 1.7500000
2.0000000 4.2500000
[8] 5.0000000 23.0000000
$method
[1] "misclass "
```

```
attr(," class ")
[1] "prune" "tree.sequence "
```

Note that, despite the name, **dev** corresponds to the **cross-validation error** rate in this instance. The tree with 9 terminal nodes results in the lowest cross-validation error rate, with 50 cross-validation errors.We plot the error rate as a function of both *size* and$k$.

```
> par(mfrow =c(1,2))
> plot(cv.carseats$size ,cv.carseats$dev ,type="b")
> plot(cv.carseats$k ,cv.carseats$dev ,type="b")
```

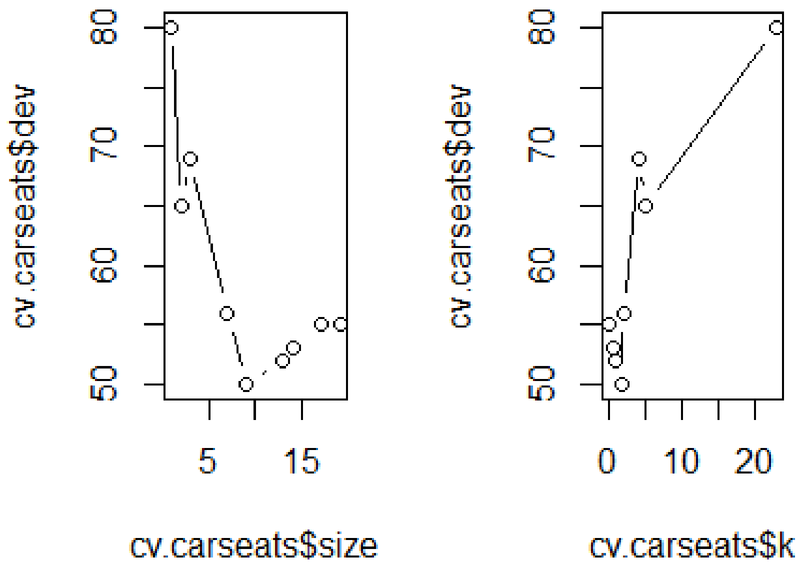

Figure 4

We now apply the **prune.misclass()** function in order to prune the tree to prune. obtain the nine-node tree.

```
> prune.carseats =prune.misclass (tree.carseats ,best =9)
> plot(prune.carseats )
> text(prune.carseats ,pretty =0)
```

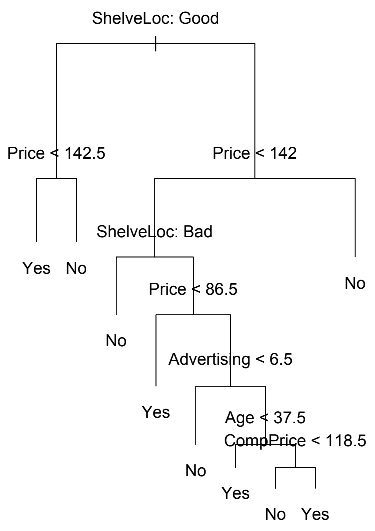

Figure 5

How well does this pruned tree perform on the test data set? Once again, we apply the **predict() function**.

```
> tree.pred=predict (prune.carseats , Carseats .test ,type=" class ")
> table(tree.pred ,High.test)
         High.test
tree.pred No   Yes
      No  94   24
     Yes  22   60
> (94+60) /200
[1] 0.77
```

Now 77% of the test observations are correctly classified, so not only has the pruning process produced a more interpretable tree, but it has also improved the classification accuracy.

## R Code

```
library (tree)
library (MASS)
library (ISLR)
?Carseats
View(Carseats)

############################################################ Creat the proper data
attach (Carseats )
High=ifelse (Sales <=8,"No"," Yes ")
Carseats =data.frame(Carseats ,High)
#################################################### Fit the Tree
```

```
tree.carseats =tree(High~.-Sales ,Carseats )
summary (tree.carseats )
################################# plot
plot(tree.carseats )
text(tree.carseats ,pretty =0)
###############
tree.carseats
####################################################### Using Training and Test Data
set.seed(2)
train=sample (1: nrow(Carseats ), 200)
Carseats.test=Carseats [-train ,]
High.test=High[-train ]
##########################################
tree.carseats =tree(High~.-Sales ,Carseats ,subset =train )  ###<--- Using Training data to bui
tree.pred=predict (tree.carseats ,Carseats.test ,type ="class")##<-- Using Testing data to pred
table(tree.pred ,High.test)
(86+57) /200
##########################################Prunning the Tree
set.seed(3)
cv.carseats =cv.tree(tree.carseats ,FUN=prune.misclass )
names(cv.carseats )
cv.carseats
#################
par(mfrow =c(1,2))
plot(cv.carseats$size ,cv.carseats$dev ,type="b")
plot(cv.carseats$k ,cv.carseats$dev ,type="b")
#################################################################Plot the Pruned Tree withn nir
prune.carseats =prune.misclass (tree.carseats ,best=9)
plot(prune.carseats )
text(prune.carseats ,pretty =0)
#################################################How well this mosel predict?
tree.pred=predict (prune.carseats , Carseats.test ,type="class")
table(tree.pred ,High.test)
(94+60) /200
```