Decision Trees Lab.

27 de marzo, 2023

Contents

```r
# In this chunk, you do the assignment of values to R variables
# This code must be adapted to any change of values of R variables

myDescription <- "The data are a simulated data set containing sales of child car seats
at different stores [@james2013introduction]"
mydataset <- Carseats
```

```r
n <- nrow(mydataset)
p <- ncol(mydataset)
```

1 Description data

In this section, you should be do a short explain about problem and the data set.

The data are a simulated data set containing sales of child car seats at different stores (James et al. 2013).

The data set has **400** observations on **11** variables. The variable names are: *Sales, CompPrice, Income, Advertising, Population, Price, ShelveLoc, Age, Education, Urban, US*.

The first step is 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.

```r
# as.factor() changes the type of variable to factor
mydataset$High=as.factor(ifelse(mydataset$Sales<=8,"No","Yes"))
```

The number of observations for each class is:

```r
kable(table(mydataset$High), caption= "Number of observations for each class",
      col.names = c('High','Freq'))
```

Table 1: Number of observations for each class

High	Freq
No	236
Yes	164

The aim is of this study is predict the `High` using all variables but `Sales` so it is a classification problem.

In this case, it is apply the classification tree model.

This is a short data set summary

```r
summary(mydataset)
```

```
##      Sales          CompPrice       Income        Advertising
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000
##    Population        Price        ShelveLoc        Age          Education
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00   Min.   :10.0
```

```
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75   1st Qu.:12.0
##  Median :272.0   Median :117.0   Medium:219   Median :54.50   Median :14.0
##  Mean   :264.8   Mean   :115.8                Mean   :53.32   Mean   :13.9
##  3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00   3rd Qu.:16.0
##  Max.   :509.0   Max.   :191.0                Max.   :80.00   Max.   :18.0
##  Urban       US       High
##  No :118   No :142   No :236
##  Yes:282   Yes:258   Yes:164
##
##
##
##
```

2 Preprocess

Sometimes it is should be do a data preprocess before to train the model

---

In this model is not need a data preprocess.

3 Partition of data

In order to properly evaluate the performance of a model, we must estimate the error rather than simply computing the training error. We split the observations into a training set and a test set, build the model using the training set, and evaluate its performance on the test data.

```r
set.seed(2)
pt <- 1/2
train <- sample(1:nrow(mydataset),pt*nrow(mydataset))
mydataset.test <- mydataset[-train,]
High.test <-  mydataset[-train,"High"]
```

The train set has 200 observations and the test set has 200.

In train data, the number of observations for each class is:

```r
kable(table(mydataset[train,"High"]), caption= "Train data: number of observations for each class",
      col.names = c('High','Freq'))
```

Table 2: Train data: number of observations for each class

High	Freq
No	119
Yes	81

4 Train model

We now use the `tree()` function to fit a classification tree in order to predict `High` using all variables but `Sales` using only de train set.

```r
tree.mydataset=tree(High~.-Sales, mydataset, subset=train, split="deviance")
```

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

```
summary(tree.mydataset)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = mydataset, subset = train,
##     split = "deviance")
## Variables actually used in tree construction:
## [1] "Price"       "Population"  "ShelveLoc"   "Age"         "Education"
## [6] "CompPrice"   "Advertising" "Income"      "US"
## Number of terminal nodes:  21
## Residual mean deviance:  0.5543 = 99.22 / 179
## Misclassification error rate: 0.115 = 23 / 200
```

For a classification reported in the output of `summary()` is given by

$$-2 \sum_m \sum_k n_{mk} log(\hat{p}_{mk}),$$

where n_{mk} is the number of observations in the `m`th terminal node that belong to the `k`th class. The *residual mean difference* reported is simply the deviance divided by $n - |T_0|$ where T_0 is the number of terminal nodes.

The next step is display the tree graphically. We use the `plot()` function to display the tree structure, and the `text()`function to display the node labels.

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

It is also possible to show a `R` prints output corresponding to each branch of the tree.

```
tree.mydataset
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 200 270.000 No ( 0.59500 0.40500 )
##     2) Price < 96.5 40  47.050 Yes ( 0.27500 0.72500 )
##       4) Population < 414 31  40.320 Yes ( 0.35484 0.64516 )
##         8) ShelveLoc: Bad,Medium 25  34.300 Yes ( 0.44000 0.56000 )
##          16) Age < 64.5 17  20.600 Yes ( 0.29412 0.70588 )
##            32) Education < 13.5 7   0.000 Yes ( 0.00000 1.00000 ) *
##            33) Education > 13.5 10  13.860 Yes ( 0.50000 0.50000 )
##              66) Education < 16.5 5   5.004 No ( 0.80000 0.20000 ) *
##              67) Education > 16.5 5   5.004 Yes ( 0.20000 0.80000 ) *
##          17) Age > 64.5 8   8.997 No ( 0.75000 0.25000 ) *
##         9) ShelveLoc: Good 6   0.000 Yes ( 0.00000 1.00000 ) *
##       5) Population > 414 9   0.000 Yes ( 0.00000 1.00000 ) *
##     3) Price > 96.5 160 201.800 No ( 0.67500 0.32500 )
##       6) ShelveLoc: Bad,Medium 135 154.500 No ( 0.74074 0.25926 )
##        12) Price < 124.5 82 107.700 No ( 0.63415 0.36585 )
##          24) Age < 49.5 34  45.230 Yes ( 0.38235 0.61765 )
##            48) CompPrice < 130.5 21  28.680 No ( 0.57143 0.42857 )
##              96) Population < 134.5 6   0.000 No ( 1.00000 0.00000 ) *
##              97) Population > 134.5 15  20.190 Yes ( 0.40000 0.60000 )
##               194) Population < 343 7   5.742 Yes ( 0.14286 0.85714 ) *
```

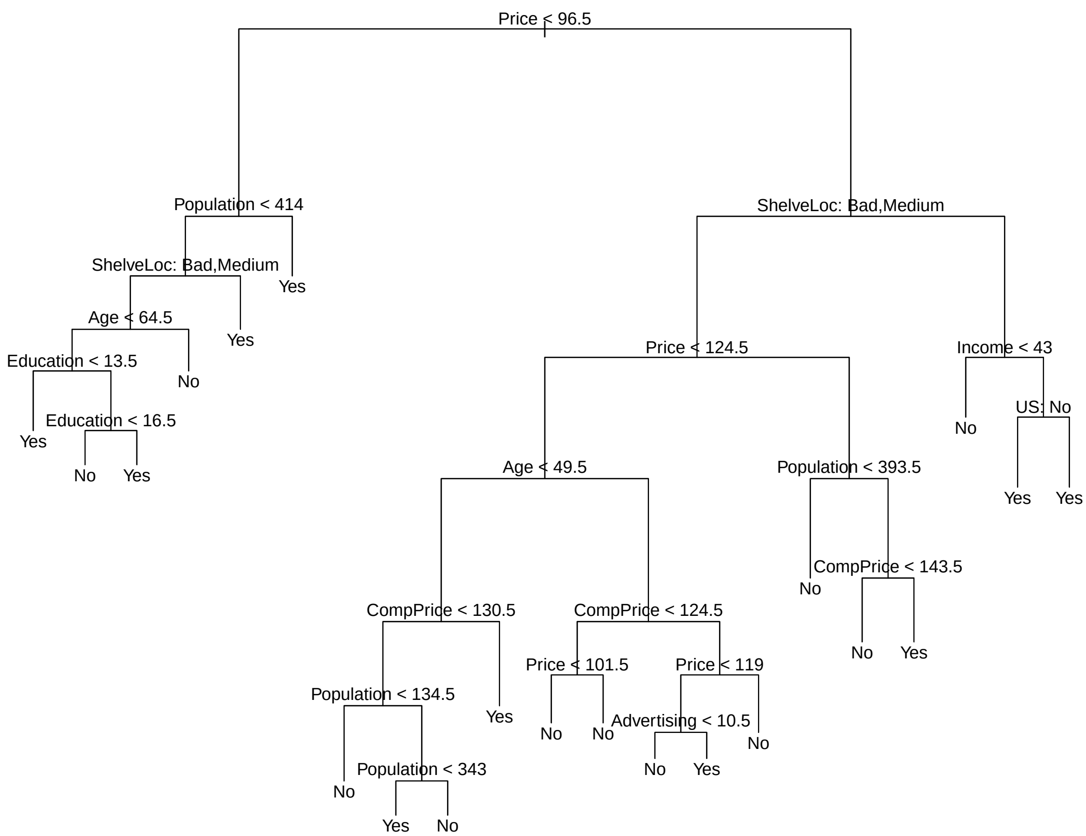


Figure 1: Classification tree

```
##               195) Population > 343 8  10.590 No ( 0.62500 0.37500 ) *
##            49) CompPrice > 130.5 13   7.051 Yes ( 0.07692 0.92308 ) *
##          25) Age > 49.5 48  46.330 No ( 0.81250 0.18750 )
##            50) CompPrice < 124.5 28  14.410 No ( 0.92857 0.07143 )
##             100) Price < 101.5 8   8.997 No ( 0.75000 0.25000 ) *
##             101) Price > 101.5 20   0.000 No ( 1.00000 0.00000 ) *
##            51) CompPrice > 124.5 20  25.900 No ( 0.65000 0.35000 )
##             102) Price < 119 14  19.410 No ( 0.50000 0.50000 )
##               204) Advertising < 10.5 9  11.460 No ( 0.66667 0.33333 ) *
##               205) Advertising > 10.5 5   5.004 Yes ( 0.20000 0.80000 ) *
##             103) Price > 119 6   0.000 No ( 1.00000 0.00000 ) *
##        13) Price > 124.5 53  33.120 No ( 0.90566 0.09434 )
##          26) Population < 393.5 34   0.000 No ( 1.00000 0.00000 ) *
##          27) Population > 393.5 19  21.900 No ( 0.73684 0.26316 )
##            54) CompPrice < 143.5 13   7.051 No ( 0.92308 0.07692 ) *
##            55) CompPrice > 143.5 6   7.638 Yes ( 0.33333 0.66667 ) *
##       7) ShelveLoc: Good 25  31.340 Yes ( 0.32000 0.68000 )
##        14) Income < 43 7   8.376 No ( 0.71429 0.28571 ) *
##        15) Income > 43 18  16.220 Yes ( 0.16667 0.83333 )
##          30) US: No 6   8.318 Yes ( 0.50000 0.50000 ) *
##          31) US: Yes 12   0.000 Yes ( 0.00000 1.00000 ) *
```

5 Prediction

We now evaluate the performance of the classification tree on the test data. The `predict()` function can be used for this purpose.

```
tree.pred=predict(tree.mydataset,mydataset.test,type="class")
res <- table(tree.pred,High.test)
res
```

```
##          High.test
## tree.pred  No Yes
##       No  104  33
##       Yes  13  50
```

```
accrcy <- sum(diag(res)/sum(res))
```

The accuracy is **0.77** or misclassification error rate is **0.23**.

6 Prune the decision tree (Tunning model) with prediction

We consider whether pruning the tree lead to improved results.

```
set.seed(3)
cv.mydataset=cv.tree(tree.mydataset,FUN=prune.misclass)
names(cv.mydataset)
```

```
## [1] "size"   "dev"    "k"      "method"
```

```
cv.mydataset
```

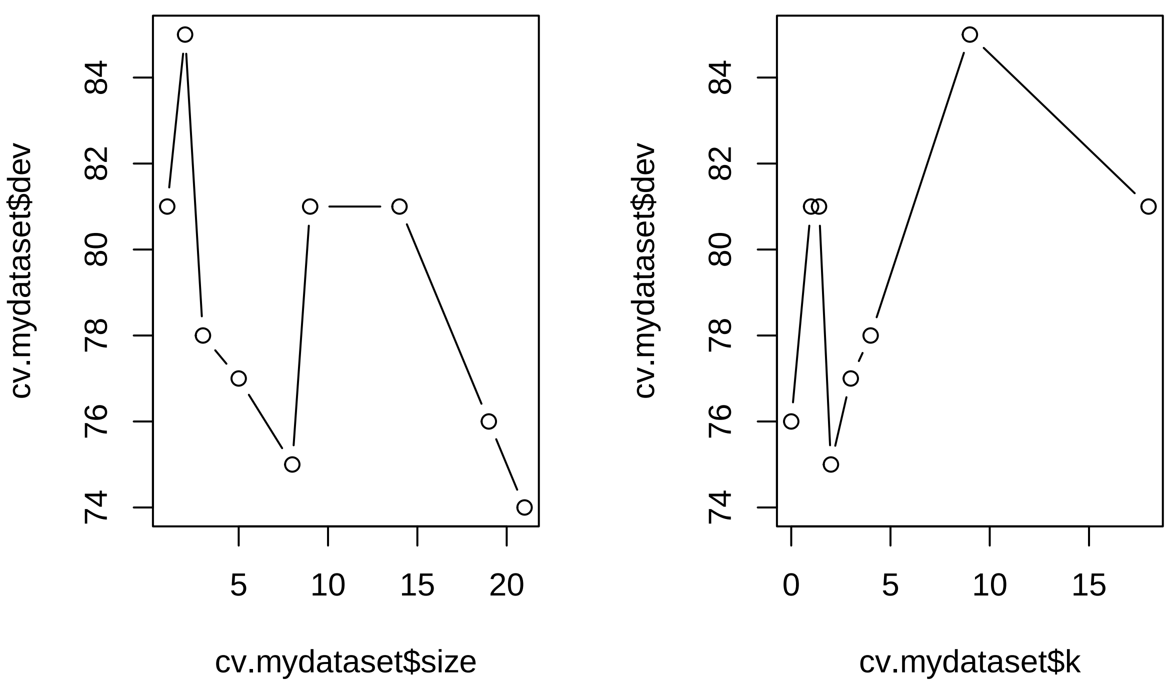
```
## $size
## [1] 21 19 14  9  8  5  3  2  1
##
## $dev
```

```
## [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"
```

Note that, despiste the name, `dev` corresponds to the cross-validation error rate in this instance.

We plot the error rate as a function of both `size`and `k`.

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




```
par(mfrow=c(1,1))
```

We now apply the `prune.misclass()` function in order to prune the tree to obtain the a best tree. The best tree is the tree with …

```
prune.mydataset=prune.misclass(tree.mydataset,
                               best=cv.mydataset$size[which.min(cv.mydataset$dev)])
plot(prune.mydataset)
text(prune.mydataset,pretty=0)
```

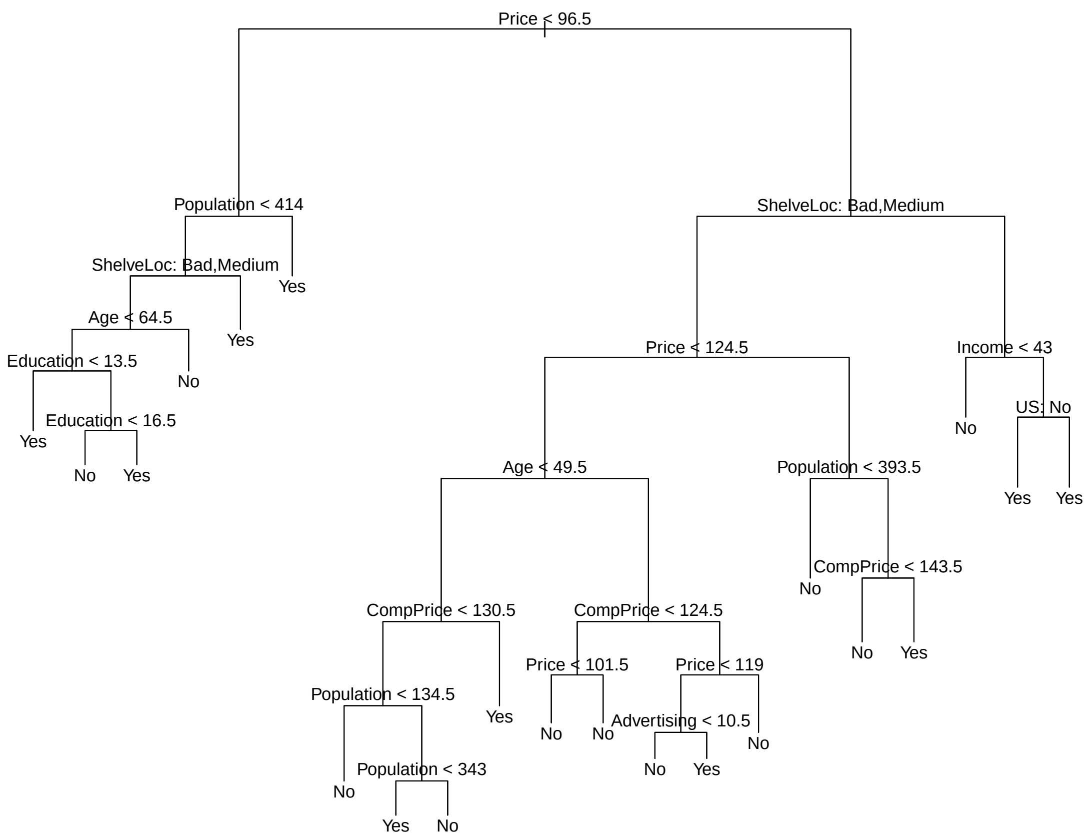


Figure 2: The best classification pruned tree

How well does this pruned tree perform on the test data set?

```
tree.pred=predict(prune.mydataset,mydataset.test,type="class")
res <- table(tree.pred,High.test)
res
```

```
##          High.test
## tree.pred  No Yes
##       No  104  32
##       Yes  13  51
```

```
accrcy <- sum(diag(res)/sum(res))
```

The accuracy is **0.775**.

If we increase the value of `best`, for example 21 terminal nodes, we obtain a larger pruned tree with lower classification accuracy:

```
prune.mydataset=prune.misclass(tree.mydataset,
                               best = cv.mydataset$size[1])
plot(prune.mydataset)
text(prune.mydataset, pretty=0)
```

```
tree.pred=predict(prune.mydataset, mydataset.test, type="class")
res <- table(tree.pred, High.test)
res
```

```
##          High.test
## tree.pred  No Yes
##       No  105  31
##       Yes  12  52
```

```
accrcy <- sum(diag(res)/sum(res))
```

The accuracy is **0.785**.

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

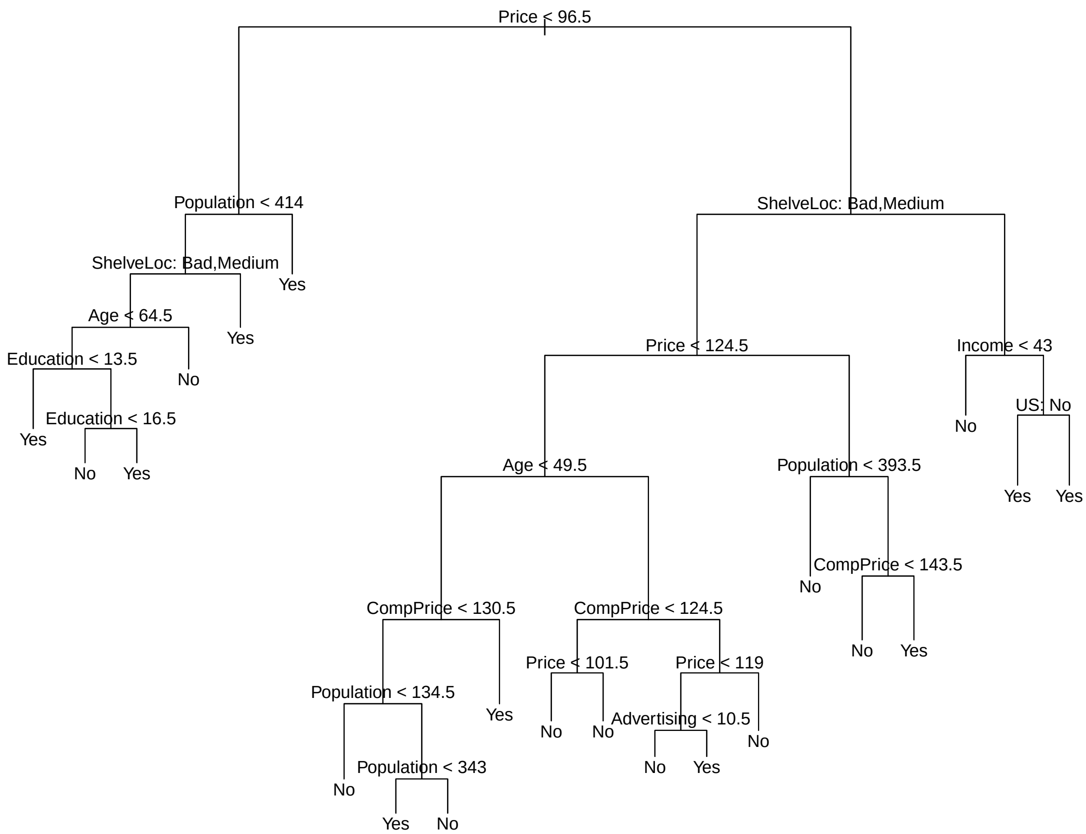


Figure 3: Other classification pruned tree