INFX 573 Lab: Tree-Based Method

Pierre Augustamar

November 23rd, 2016

Collaborators:

Don't forget to list the full names of your collaborators!

Instructions:

Before beginning this assignment, please ensure you have access to R and/or RStudio.

1. Download the `week9a_lab_james_trees.pdf` file from Canvas. Open `week9a_lab_james_trees.pdf` in RStudio (or your favorite editor) and supply your solutions to the assignment by editing `week9a_lab_james_trees.pdf`.

2. Replace the "Insert Your Name Here" text in the `author:` field with your own full name.

3. Be sure to include code chucks, figures and written explanations as necessary. Any collaborators must be listed on the top of your assignment. Any figures should be clearly labeled and appropriately referenced within the text.

4. When you have completed the assignment and have **checked** that your code both runs in the Console and knits correctly when you click `Knit`, rename the R Markdown file to `YourLastName_YourFirstName_lab6a.Rmd`, and knit it into a PDF. Submit the compiled PDF on Canvas.

   In this lab, you will need access to the following R packages:

```
# Load some helpful libraries
library(tree)
```

```
## Warning: package 'tree' was built under R
## version 3.3.2
```

Fitting Classification tree using the carseats data

```
library(ISLR)  # Download and load data
attach(Carseats)  #making carseats available
# set High to no if sales is less than equal
# to 8, otherwise set High to yes
High = ifelse(Sales <= 8, "No", "Yes")
```

```
Carseats = data.frame(Carseats, High)  #merge high with the rest of the carseats
```

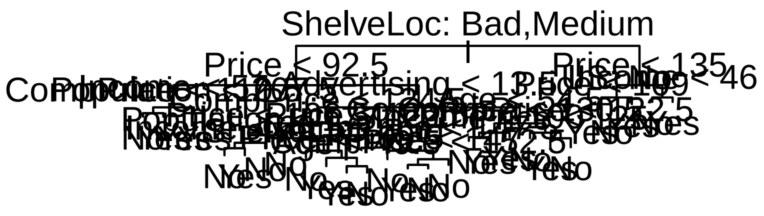
```
# fit a classfication tree in order to predict
# high using all variables excluding sales
tree.carseats = tree(High ~ . - Sales, Carseats)
```

```
summary(tree.carseats)  #list the variables that are used as internal nodes in the tree
```

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

```
plot(tree.carseats)  #display the tree structure

# display the node labels and include the
# category names for any qualitative
# predictors
text(tree.carseats, pretty = 0)
```



```
tree.carseats  #show output corresponding to each branch of the tree.
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500 No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680 Yes ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
##               170) Price < 109.5 16   7.481 Yes ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
##            43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##              86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##              87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##               174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##                 348) CompPrice < 152.5 7   5.742 Yes ( 0.14286 0.85714 ) *
##                 349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
##               175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
##        11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
##            44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##              88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##              89) Income > 100 5   0.000 Yes ( 0.00000 1.00000 ) *
##            45) CompPrice > 130.5 11   0.000 Yes ( 0.00000 1.00000 ) *
##          23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##            46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
##            47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##              94) Price < 125 5   0.000 Yes ( 0.00000 1.00000 ) *
##              95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
##     3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##       6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##        12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##          24) Price < 109 8   0.000 Yes ( 0.00000 1.00000 ) *
##          25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##        13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##       7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##        14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
##        15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *
```

Split the observations into a training set and a test set

```
set.seed(2)  #set seed to ensure that results and figures are reproducible.
# generate 200 values ranging from 1 to the
# number of rows in carseats for the train
# data
train = sample(1:nrow(Carseats), 200)
Carseats.test = Carseats[-train, ]  #create test data for carseats
High.test = High[-train]  #select all but the the train data
```

Tree based methods

```
tree.carseats = tree(High ~ . - Sales, Carseats,
    subset = train)  #create a decision tree
# generate an estimation using type = class
tree.pred = predict(tree.carseats, Carseats.test,
    type = "class")
# table of prediction for high test data
table(tree.pred, High.test)
```

```
##          High.test
## tree.pred No Yes
##       No  86  27
##       Yes 30  57
```

```
# set seed to ensure that results and figures
# are reproducible.
set.seed(3)
# perform cross valdation through prunning
# process
```

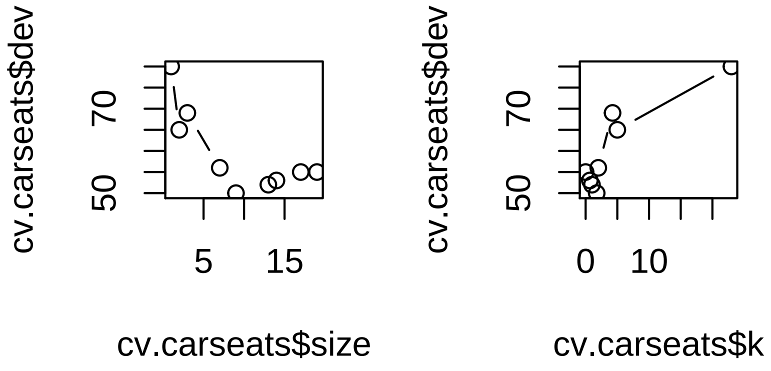
```
cv.carseats = cv.tree(tree.carseats, FUN = prune.misclass)
names(cv.carseats)  #get the names of the cross validation for
```

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

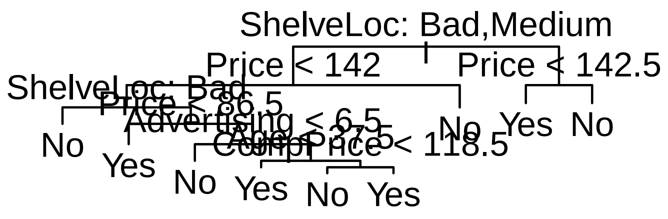
```
cv.carseats  #show output corresponding to the cross validation carseats
```

```
## $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
## [4]  1.0000000  1.7500000  2.0000000
## [7]  4.2500000  5.0000000 23.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
# create a multi panel with 1 row and 2
# columns
par(mfrow = c(1, 2))
# display the error rate as a function of the
# size
plot(cv.carseats$size, cv.carseats$dev, type = "b")
# display the error rate as a function of k
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```



```
# prune the tree to obtain the nine-node tree
prune.carseats = prune.misclass(tree.carseats,
    best = 9)
plot(prune.carseats)  #display the pruned carseats
# adding test to the display pruned with a
# nicer settings
text(prune.carseats, pretty = 0)
```



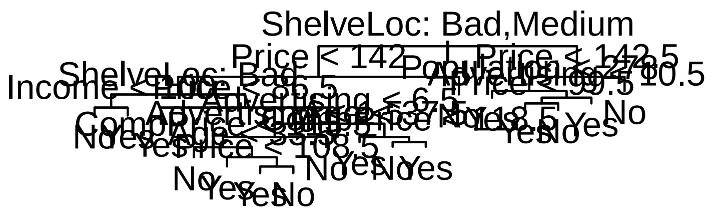
```
# unpruned the tree to make prediction on the
# test data
tree.pred = predict(prune.carseats, Carseats.test,
    type = "class")
table(tree.pred, High.test)  #table of prediction for high test data
```

```
##           High.test
## tree.pred No Yes
##       No  94  24
##       Yes 22  60
```

```
# figure out percentage of the observations
# that are correctly classified
(94 + 60)/200
```

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

```
# prune the tree to obtain the fifteen noded
# tree
prune.carseats = prune.misclass(tree.carseats,
    best = 15)
plot(prune.carseats)  #display the pruned carseats
text(prune.carseats, pretty = 0)  #adding test to the display pruned with a nicer settings
```



```
# apply predict to find out how the pruned
# performed against the test data
tree.pred = predict(prune.carseats, Carseats.test,
    type = "class")
table(tree.pred, High.test)  #table of prediction for high test data
```

```
##          High.test
## tree.pred No Yes
##       No  86  22
##       Yes 30  62
```

```
# figure out percentage of the observations
# that are correctly classified
(86 + 62)/200  #
```

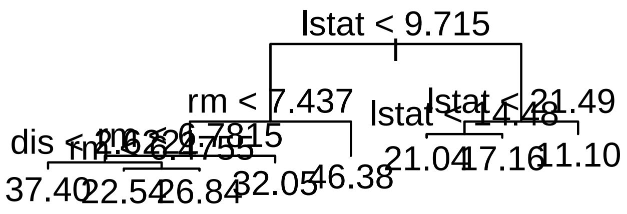
```
## [1] 0.74
```

Fitting Regression Trees using Boston data set

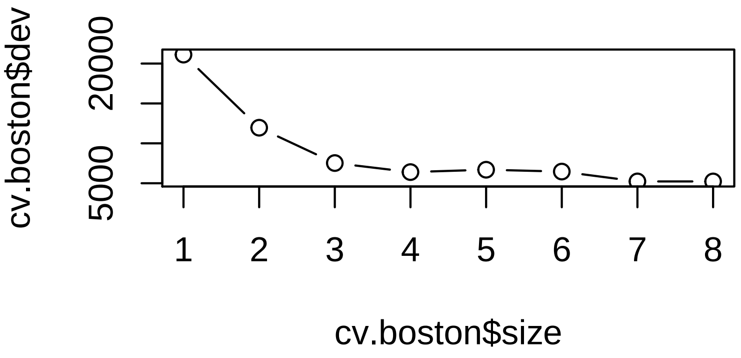
```
library(MASS)  #load the library that contains Boston data
set.seed(1)  #set seed to ensure that results and figures are reproducible.
train = sample(1:nrow(Boston), nrow(Boston)/2)  #create a training set
tree.boston = tree(medv ~ ., Boston, subset = train)  #fit the tree to the training data
summary(tree.boston)  #list the variables that are used as internal nodes in the tree
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean
## -14.10000  -2.04200  -0.05357   0.00000
##   3rd Qu.      Max.
##   1.96000  12.60000
```

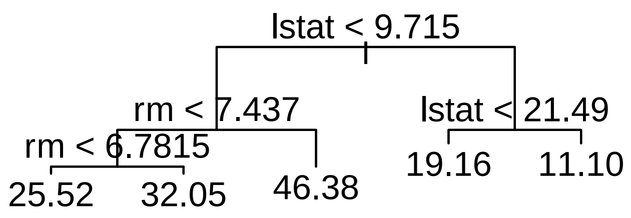
```
# display the boston tree node
plot(tree.boston)
# adding test to the display pruned with a
# nicer settings
text(tree.boston, pretty = 0)
```



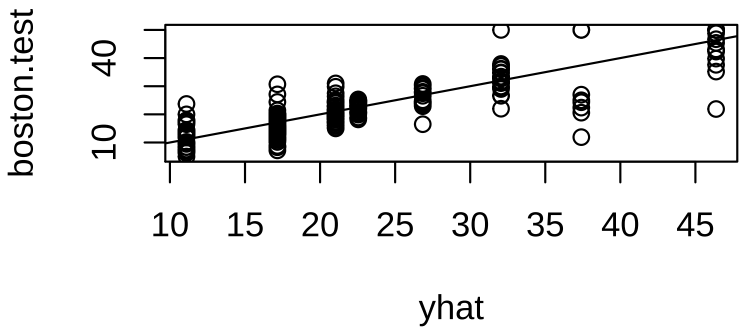
```
# Check if pruning will improve the
# performance
cv.boston = cv.tree(tree.boston)
# display the error rate as a function of the
# size
plot(cv.boston$size, cv.boston$dev, type = "b")
```



```
# prun the boston tree to obtain the 5 nodes
# tree
prune.boston = prune.tree(tree.boston, best = 5)
plot(prune.boston)  #display the prune tree
# adding test to the display pruned with a
# nicer settings
text(prune.boston, pretty = 0)
```



```r
# unpruned the tree to make predictions on the
# test set
yhat = predict(tree.boston, newdata = Boston[-train,
    ])
boston.test = Boston[-train, "medv"]
# display the unpruned test data
plot(yhat, boston.test)
abline(0, 1)  # add a straight line to the plot
```



```r
mean((yhat - boston.test)^2)  #calculate the mean value of test data
```

```
## [1] 25.04559
```