workshop11

February 3, 2019

```
In [54]: library(tidyverse)
         library(rpart)
         library(rpart.plot)
         library(caret)
         library(ISLR)
         library(Metrics)
         library(RPostgreSQL)



         # Plot size deppening on your screen resolution to 5 x 3
         options(repr.plot.width=6, repr.plot.height=6)
```

1 Welcome to Workshop 11

1.0.1 Exercise 1: Build a Regression Tree

For this workshop, we will build a tree based on a continuous response variable. In a regression tree, the prediction at each node is the mean value of the target variable for data points in that node. The measure of disorder is calculated for each node.

```
In [2]: str(Carseats)
```

```
'data.frame':        400 obs. of  11 variables:
 $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
 $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
 $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
 $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
 $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
 $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
 $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
 $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
 $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
 $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
 $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

```
In [3]: head(Carseats)
```

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

```
In [4]: #If you do not have an ID per row, use the following code to create an ID
        carseats.df <- Carseats %>%
                           mutate(id = row_number())


        #Check IDs
        head(carseats.df)

        #Create training set
        train.df <- carseats.df %>%
                       sample_frac(.75)


        #Create test set
        test.df  <- anti_join(carseats.df, train.df, by = 'id')


        train.df <- train.df %>%
            select(-id)

        test.df <- test.df %>%
            select(-id)
```

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

Let's now build a tree explaining Sales using the training set

```
In [5]: tree_model<-rpart(Sales ~ .,
                          data=train.df,
                          method="anova",
                          control=rpart.control( maxdepth = 2, # max depth
                                                  xval = 1)) # to use all samples for the first try
        tree_model
```

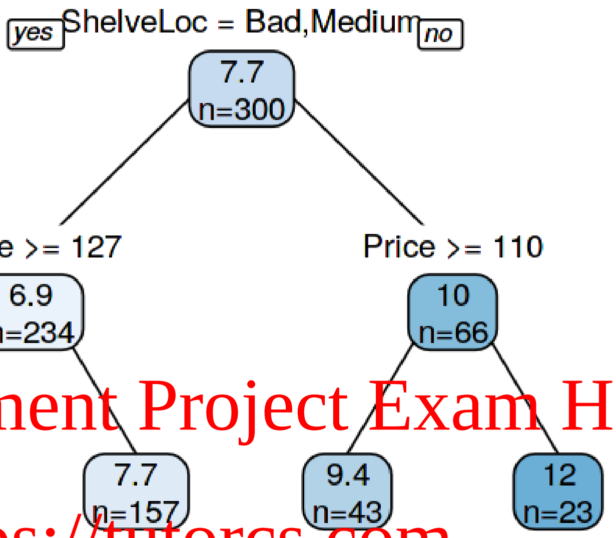
```
n= 300

node), split, n, deviance, yval
      * denotes terminal node

1) root 300 2393.40100  7.694233
  2) ShelveLoc=Bad,Medium 234 1355.81400  6.933504
    4) Price>=127 77  341.19520  5.471169 *
    5) Price< 127 157  769.20400  7.650701 *
  3) ShelveLoc=Good 66  422.05140 10.391360
    6) Price>=109.5 43  221.86590  9.357209 *
    7) Price< 109.5 23   68.22157 12.324780 *
```

```
In [6]: printcp(tree_model)
```

```
Regression tree:
rpart(formula = Sales ~ ., data = train.df, method = "anova",
    control = rpart.control(maxdepth = 2, xval = 1))

Variables actually used in tree construction:
[1] Price     ShelveLoc

Root node error: 2393.4/300 = 7.978

n= 300

        CP nsplit rel error
1 0.257180      0   1.00000
2 0.102538      1   0.74282
3 0.055137      2   0.64028
4 0.010000      3   0.58514
```

Let's plot the tree, this is one of the few times we make use of the R-base graphics

```
In [7]: rpart.plot(tree_model,
                   fallen.leaves = FALSE, # to position the leaf nodes at the bottom of the grap
                   type  = 1, # 1 Label all nodes, not just leaves.
                   extra = 1, # 1 Display the number of observations that fall in the node
                   split.font = 1, # Font for the split labels. 1=normal 2=bold
                   varlen = -10) # Length of variable names in text at the splits (and, for clas
```



Evaluate its performance on the test-set:

```
In [8]: treePred.reg <- predict(tree_model, newdata = test.df)

        head(treePred.reg)
```

1 5.47116883116883 **2** 7.65070063694267 **3** 9.35720930232558 **4** 9.35720930232558 **5**
9.35720930232558 **6** 12.3247826086957

Lets calculate the RMSE:

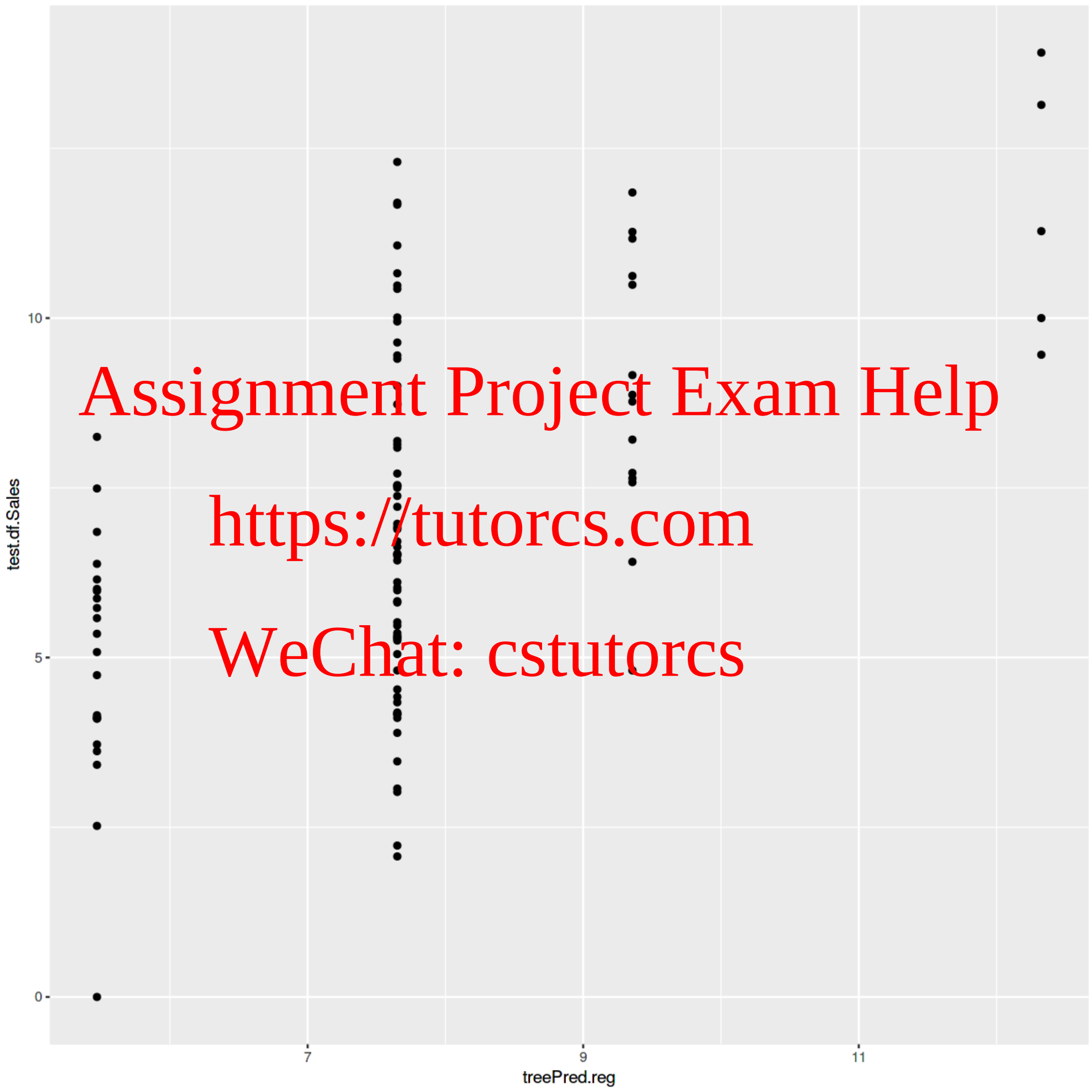
```
In [9]: rmse(test.df$Sales, treePred.reg)
```

2.35049935871488

```
In [38]: options(repr.plot.width=10, repr.plot.height=10)
```

Create a new data frame with predicted and actual values and create a scatter plot. Fix the axis to have the same dimensions:

```
In [39]: p<- data.frame(test.df$Sales,treePred.reg)
         ggplot(data=p,
               aes(x=treePred.reg,
                  y=test.df.Sales))+
         geom_point()
```



If you look at this graph, you will see that there is not much variation in the predicted values. Change the parameters to decrease RMSE and make the graph look better. For this write nested for loops to perform a grid search across parameters on the training set.

```
In [68]: minsplit <- 1:2
         minbucket <- 1:2
```

```
maxdepth <- 1:5

n<- length(minsplit)*length(minbucket)*length(maxdepth)

gridsearch.df <- data.frame(minsplit = numeric(n),
                            minbucket = numeric(n),
                            maxdepth = numeric(n),
                            rmse = numeric(n))


f=1
for( i in minsplit)
{
    for (j in minbucket)
        {
            for(k in maxdepth)
                {
                   tree_model<-rpart(Sales ~ .,
                 data=train.df,
                 method="anova",
                 control=rpart.control( minsplit =i, minbucket=j,maxdepth=k,
                                      xval = 1))
                treePred.reg <- predict(tree_model, newdata=test.df)
                gridsearch.df$minsplit[f]<-i
                gridsearch.df$minbucket[f]<-j
                gridsearch.df$maxdepth[f]<-k
                gridsearch.df$rmse[f]<-rmse(test.df$Sales, treePred.reg)
                f<-f+1
            }
    }
}
```

```
In [ ]: gridsearch.df %>%
            arrange(rmse)
```

Pick the best configuration, re-run the build of a tree and provide a scatter plot. What do you see now ?

```
In [ ]: # your code here
        fail() # No Answer - remove if you provide an answer
```

1.0.2 Exercise 2: Flights from JFK

Connect to the database as in workshop02 and load the table `public.flight_delay_workshop`. You will see that the first columns represent the arrival delay in minutes. Build a predictive model that will predict the delay as accurate as possible for December, with using training data up to November. What RMSE can you achieve ? What could you do to achieve a higher value ?

```
In [24]: drv <- dbDriver('PostgreSQL')
         con <- dbConnect(
           drv,
           host = "118.138.234.161",
           dbname = "summer2019",
           user =  "student",
           password = "4XcxqUo6AHPn"
         )
```

```
In [50]: query <- "
         SELECT
             *
         FROM
             public.flight_delay_workshop;
         "

         flight.df <- dbGetQuery(con, query)
         head(flight.df)
```

arrdelay	carrier	dayofweek	month	departurehour	destination	origin	weatherdelay	airtime
14	MQ	5	2	16	RDU	JFK	0	82
35	MQ	1	2	17	RDU	JFK	0	74
56	MQ	7	2	17	RDU	JFK	0	79
92	MQ	3	2	18	RDU	JFK	0	88
4	MQ	6	2	17	RDU	JFK	0	70
36	MQ	6	2	17	RDU	JFK	0	70