Insurance Case Study

Regression Tree using CART

EduPristine

Contents

Insurance Case Study – Estimation of Losses

Introduction

In the insurance industry, a lot of times an Insurance company would like to assess what kind of claims, in terms of monetary value, their consumers make. This usually helps insurance companies evaluate the premiums being offered and the claims being made by the consumers. Once the insurance company has these details, they would be able calculate the Losses they may incur from each of the consumer. This case study is going to help an insurance company, which is into Motor Insurance, build a statistical model that in turn would help them in assessing their consumer base.

The expectation out of this case study is to make use of various factors that affect/ influence a claim process and predict a monetary value, which is the claim amount, for each of the customer. Hence, the dependent variable is a continuous value, and independent variables comprise of continuous as well categorical variables. Unlike, Linear Regression, where the participants are expected to build a regression equation to arrive at the predicted values, the technique to be used in this case study will be to build a Regression Tree using CART algorithm.

Dataset Explained

The dataset contains the following variables:

Variable Name	Description
Policy Number	Unique Id to identify each customer
Gender	Male/ Female
Married	Married or Single
Age	Age of the Customer
Vehicle Age	Number of Years Vehicle has been in use
Years of Driving Experience	Number of Years customer has been driving
Number of Vehicles	Number of Vehicles owned by the Customer
Fuel Type	Diesel/ Petrol
Losses	(Dependent Variable) Loss Amount

CART Algorithm – Regression Tree

The following list high lights the key steps in constructing a regression tree using CART:

1. Starting with the root node, CART per forms all possible splits on each of the predictor variables, applies a predefined node impurity measure to each split, and deter mines the reduction in impurity that is achieved.
2. CART then selects the "best" split by applying the goodness- of- split criteria and partitions the data set into left- and right- child nodes.
3. Because CART is recursive, it repeats steps 1 and 2 for each of the nonterminal nodes and produces the largest possible tree.
4. Finally, CART applies its pruning algorithm to the largest tree and produces a sequence of subtrees of different sizes from which an optimal tree is selected.

CART Regression Tree Splitting Criteria

There are two split ting rules or impurity functions for a regression tree. These are (1) the Least Squares (LS) function and (2) the Least Absolute Deviation (LAD) function. Since the mechanism for both rules is the same, only the LS impurity measure will be described. Under the LS criterion, node impurity is measured by within- node sum of squares, SS(t), which is defined as

$\mathbf{SS(t) = \sum(y_{i(t)} - y_{(t)})^2}$, for $i = 1,2, . . .,N_t$,

Where, $y_{i(t)}$ = individual values of the dependent variable at node t, and

$y_{(t)}$ = the mean of the dependent variable at node t.

Given the impurity function, SS(t), and split s that sends cases to left (tL) and right (tR) nodes, the goodness of a split is measured by the function

$\mathbf{f(s,t) = SS(t) - SS(t_R) - SS(t_L)}$**,**

Where, $SS(t_R)$ is the sum of squares of the right child node, and

$SS(t_L)$ is the sum of squares of the left child node

<u>The best split is that split for which f(s,t) is the highest.</u> From the series of splits generated by a variable at a node, the rule is to choose that split that results in the maximum reduction in the impurity of the parent node.

Advantages of the CART Algorithm

Following are some of the advantages of using CART algorithm:

1. CART makes no distributional assumptions of any kind for dependent and independent variables. No variable in CART is assumed to follow any kind of statistical distribution.

2. The explanatory variables in CART can be a mixture of categorical and continuous.

3. CART has a built- in algorithm to deal with the missing values of a variable for a case, except when a linear combination of variables is used as a split ting rule.

4. CART is not at all affected by the outliers, collinearities, heteroskedasticity, or distributional error structures that affect parametric procedures. Outliers are isolated into a node and thus have no effect on splitting. Contrary to situations in parametric modelling, CART makes use of col linear variables in “surrogate” splits.

5. CART has the ability to detect and reveal variable interactions in the data set.

6. CART does not vary under a mono tone transformation of independent variables; that is, the transformation of explanatory variables to logarithms or squares or square roots has no effect on the tree produced.

7. CART’s major advantage is that it deals effectively with large datasets and the issues of higher dimensionality; that is, it can produce useful results from a large number of variables submitted for analysis by using only a few important variables.

Case Study Implementation Workflow

1. Load Libraries, Set working directory and Read in the Dataset
2. Exploratory Data Analysis
3. Sampling of Data – Train and Test
4. Model Training
5. Model Validation and Model Finalization

R Code Explained

```
# Edupristine | CART - Regression Tree | Insurance Case Study

# Load required R Packages

library(rpart)

library(rpart.plot)

library(forecast)

library(rattle)

# Set the working directory to folder where you have placed the Input Data

setwd(dir = "")

InsData = read.csv(file = "Insurance_Dataset.csv", header = TRUE)

View(x = InsData)

############################## Exploratory Data Analysis ##############################

# Summarize the dataset

summary(object = InsData) # Look for any missing values and aberrations in the variables

# Look at the average Losses()

for(i in 2:ncol(InsData))

{

 if(length(unique(InsData[,i])) <= 5)

 {

  AverageLoss = aggregate(x = InsData$Losses, by = list(InsData[,i]), FUN = mean)

  print(colnames(InsData)[i])

  print(AverageLoss)

  print("*************************************************************")

 }

}

# Output

# [1] "Number.of.Vehicles"

# Group.1       x

# 1       1 397.3377
```

```
# 2       2 389.9086
# 3       3 386.9976
# 4       4 388.0227
# [1] "************************************************************"
# [1] "Gender"
# Group.1        x
# 1       F 343.7089
# 2       M 437.2587
# [1] "************************************************************"
# [1] "Married"
# Group.1        x
# 1 Married 323.7442
# 2  Single 458.4060
# [1] "************************************************************"
# [1] "Fuel.Type"
# Group.1        x
# 1       D 720.0174
# 2       P 287.4458
# [1] "************************************************************"
```

\# Observation 1: Variable "Fuel.Type" having Diesel and Petrol have significant difference in the average losses claimed

\# Observation 1: Variable "Number.of.Vehicles" does not have a clear bifurcation of average losses based on the number of vehicles owned by customers

```
# Boxplot/ Plot
InsuranceBoxPlot = boxplot(x = InsData[,-1]) # Look for outliers

# Print Five-Number summary for each variable obtained from boxplot
colnames(InsuranceBoxPlot$stats) = colnames(InsData[,-1])
InsuranceBoxPlot$stats
```

```
# Output

# Age Years.of.Driving.Experience Number.of.Vehicles Gender Married Vehicle.Age Fuel.Type   Losses | Five-Number-Summary

# [1,]  16      0      1   1   1     0     2  12.53452  | Min
# [2,]  24      6      2   1   1     6     2 226.42319  | 25th Percentile
# [3,]  42     23      2   1   1     9     2 354.93787  | 50th Percentile
# [4,]  61     42      3   2   2    12     2 488.68240  | 75th Percentile
# [5,]  70     53      4   2   2    15     2 881.88665  | Max
```

The Boxplot figure shows that there are significant number of outliers in Losses column and as a result, we will approach the problem statement by breaking it down into two parts

Divide the analysis into 2 parts. Part A will include building a model with the Dependent Variable (Losses) as is, and on the other hand, Part B will include building a model after "capping" the Dependent Variable to a certain upper limit value

```
############################## PART A Starts ##############################

# Make a copy of the Original Dataset
InsDataUncapped = InsData

# Random Sampling
set.seed(777) # To ensure reproducibility
Index = sample(x = 1:nrow(InsDataUncapped), size = 0.7*nrow(InsDataUncapped))

# Create Train dataset
InsDataTrainUncapped = InsDataUncapped[Index, ]
nrow(InsDataTrainUncapped)

# Create Test dataset
InsDataTestUncapped = InsDataUncapped[-Index, ]
nrow(InsDataTestUncapped)
```

############################## Model Iterations ##############################

We will try to build different CART Regression Tree by tweaking some of the parameters/ settings

Build a full model with default settings

set.seed(123) # To ensure reproducibility of xerrors (cross validated errors while estimating complexity paramter for tree pruning)

CartFullModel = rpart(formula = Losses ~ . , data = InsDataTrainUncapped[,-1], method = "anova")

CartFullModel

Gives the splitting values of the variables considered in the model.

Also, show the predicted values at each of the leaf nodes (the ones with *)

summary(object = CartFullModel)

Some important apsects to be noted from the output of the above command

=> Shows the Variable Importance Table, which will give an idea about the variables that have been considered for regression tree building

=> Shows Complexity Parameter Table (CP Table) - Explained further down

=> Mean or Predicted values at each of the node (same as from the command "CartFullModel")

=> Number of observations at each node

=> MSE at each node (Lower the better)

Plot the Regression Tree

rpart.plot() function to plot the model

Expand the plot window in R Studio to see a presentable output

rpart.plot(x = CartFullModel, type = 4,fallen.leaves = T, cex = 0.6)

title("CartFullModel") # Enlarge the plot by clicking on Zoom button in Plots Tab on R Studio

fancyRpartPlot() function to plot the same model

Expand the plot window in R Studio to see a presentable output

fancyRpartPlot(model = CartFullModel, main = "CartFullModel", cex = 0.6)

The following code also produces the same output, but in a windowed form

windows()

fancyRpartPlot(model = CartFullModel, main = "CartFullModel", cex = 0.6)

printcp(x = CartFullModel)

Output

Variables actually used in tree construction:

[1] Age Fuel.Type Gender Married Vehicle.Age

Shows the variables that have been actually used for building the regression tree

```
# CP        nsplit rel error  xerror    xstd
# 0.529185      0   1.00000 1.00025 0.041798
# 0.062940      1   0.47082 0.47104 0.035081
# 0.021788      2   0.40787 0.40814 0.035068
# 0.018227      4   0.36430 0.36511 0.032545
# 0.015568      5   0.34607 0.34612 0.033006
# 0.010000      7   0.31494 0.31670 0.030775
```

The CP table also shows us valueable information in terms of pruning the tree (which is explained later in the code)

=> The complexity parameter "CP" specifies how the cost of a tree C(T) is penalized by the number of terminal nodes |T|.

Hence, Small "CP" results in larger trees and potential overfitting, large CP" in small trees and potential underfitting.

=> The "rel error" is 1- RSquare, similar to linear regression. This is the error on the observations used to estimate the model. The last node value of rel error suggests the R-Square of the model, if this happens to be the model

=> The "xerror" is is the error on the observations from cross validation data, which happens to be a 10-Fold Cross Validation. Particpants need to note that, in order to reproduce the "xerror" values, they must use the same set.seed() number each time

=> Root Node Error is given by sum((Dependent - Mean(Dependent))^2), i.e.

sum((InsDataTrainUncapped$Losses-mean(InsDataTrainUncapped$Losses))^2

```
rsq.rpart(x = CartFullModel)
```

This produces a plot which may help particpants to look for a model depending on R-Square values produced at various splits

Lets change rpart.control() to specify certain attributes for tree building

```
RpartControl = rpart.control(cp = 0.005)
set.seed(123)
CartModel_1 = rpart(formula = Losses ~ . , data = InsDataTrainUncapped[,-1], method = "anova", control = RpartControl)
```

Display Model related output and Model evaluation/ optimization results

```
CartModel_1
summary(CartModel_1)
rpart.plot(x = CartModel_1, type = 4,fallen.leaves = T, cex = 0.6)
printcp(x = CartModel_1)
rsq.rpart(x = CartModel_1)
```

CartModel_2

```
RpartControl = rpart.control(cp = 0.015) # Increase cp to 0.015
set.seed(123)
CartModel_2 = rpart(formula = Losses ~ . , data = InsDataTrainUncapped[,-1], method = "anova", control = RpartControl)
```

Display Model related output and Model evaluation/ optimization results

```
CartModel_2
summary(CartModel_2)
rpart.plot(x = CartModel_2, type = 4,fallen.leaves = T, cex = 0.6)
printcp(x = CartModel_2)
rsq.rpart(x = CartModel_2)
```

CartModel_3

```
RpartControl = rpart.control(cp = 0.02) # Increase cp to 0.2

set.seed(123)

CartModel_3 = rpart(formula = Losses ~ . , data = InsDataTrainUncapped[,-1], method = "anova", control = RpartControl)
```

Display Model related output and Model evaluation/ optimization results

```
CartModel_3

summary(CartModel_3)

rpart.plot(x = CartModel_3, type = 4,fallen.leaves = T, cex = 0.6)

printcp(x = CartModel_3)

rsq.rpart(x = CartModel_3)
```

CartModel_4

```
RpartControl = rpart.control(cp = 0.02) # Increase cp to 0.015

set.seed(123)

CartModel_4 = rpart(formula = Losses ~ Age + Fuel.Type + Vehicle.Age,
                data = InsDataTrainUncapped[,-1], method = "anova", control = RpartControl)
```

Display Model related output and Model evaluation/ optimization results

```
CartModel_4

summary(CartModel_4)

rpart.plot(x = CartModel_4, type = 4,fallen.leaves = T, cex = 0.6)

printcp(x = CartModel_4)

rsq.rpart(x = CartModel_4)
```

#################################### Tree Pruning ###################################

The following code for pruning a tree is only required to be run for the purpose of understanding how pruning is done

Lets change rpart.control() to specify certain attributes for tree building

```
RpartControl = rpart.control(cp = 0.0005)

set.seed(123)

CartModel_5 = rpart(formula = Losses ~ . , data = InsDataTrainUncapped[,-1], method = "anova", control = RpartControl)
```

Display Model related output and Model evaluation/ optimization results

```
CartModel_5
summary(CartModel_5)
rpart.plot(x = CartModel_5, type = 4,fallen.leaves = T, cex = 0.6)
printcp(x = CartModel_5)
rsq.rpart(x = CartModel_5)
```

Pruning of Tree using prune() function using the largest tree created above CartModel_1 which has 9 splits. According to One Standard Error Rule to prune a tree, we need to find a cutoff Complexity Parameter (cp) value from the printcp() output (recorded above for CartModel_1). Any splits below that value will be removed from the tree. In order for us to arrive at that value, we will make use of xerror (cross validation error) and xstd (standard deviation) columns from printcp() of CartModel_1

Re-printing the output for ease of reference

```
printcp(x = CartModel_5)
```

```
#         CP        nsplit    rel error  xerror     xstd
# 1  0.52918486      0   1.00000 1.00025 0.041798
# 2  0.06294029      1   0.47082 0.47104 0.035081
# 3  0.02178841      2   0.40787 0.40814 0.035068
# 4  0.01822673      4   0.36430 0.36511 0.032545
# 5  0.01556764      5   0.34607 0.34612 0.033006
# 6  0.00828604      7   0.31494 0.31670 0.030775
# 7  0.00455075      9   0.29836 0.30019 0.030689
# 8  0.00410217     10   0.29381 0.29957 0.031008
# 9  0.00409550     11   0.28971 0.29639 0.031034
# 10 0.00351720     12   0.28562 0.29161 0.030993
# 11 0.00295471     13   0.28210 0.29182 0.030737
# 12 0.00223291     14   0.27914 0.28887 0.030739
# 13 0.00183829     16   0.27468 0.28370 0.030809
# 14 0.00175323     17   0.27284 0.28284 0.030806
# 15 0.00136989     18   0.27109 0.27985 0.030925
```

16 0.00116193 19 0.26972 0.27852 0.030950

17 0.00109397 21 0.26739 0.27891 0.031011

18 0.00084070 22 0.26630 0.27809 0.030873

19 0.00081822 26 0.26294 0.27896 0.030880

20 0.00077355 28 0.26130 0.27852 0.030879

21 0.00076458 30 0.25975 0.27846 0.030880

22 0.00073571 31 0.25899 0.27838 0.030880

23 0.00064577 34 0.25678 0.28067 0.030921

24 0.00062720 36 0.25549 0.28070 0.030890

25 0.00056929 37 0.25486 0.28116 0.030899

26 0.00050000 38 0.25429 0.28097 0.030902

Typically, you will want to select a tree size that minimizes the cross-validated error, the xerror column printed by printcp() OR

Use the best tree (lowest cross-validated error) or the smallest (simplest) tree within one standard error of the best tree. This may be regarded as One Standard Error rule

One Standard Error Rule Steps:

Step 1. The best model/ regression tree happens to be Row 18 in the above output. Minimum xerror, i.e. 0.27809

Step 2. With One Standard Error Rule, you would want to select the best tree which is within (xerror + xstd), i.e. (0.27809+0.030873) = 0.308963

Step 3. Scanning through the CP Table tells us that Row 7 happens to have a xerror of 0.30019, which is the best model within 0.308963

Step 4. Note down the CP value associated with the model at Row 7, i.e. 0.00455075

Step 5. Prune the tree using prune() function to remove all the rows belwow that CP value

```
set.seed(123)

CartPrunedModel = prune(tree = CartModel_5, cp = 0.0045508)

printcp(CartPrunedModel) # Validate pruned tree by seeing the printcp result

##########################################################################################
```

Model Evaluation Measures on test dataset using the finalized (pruned model)

Use predict() the get the predicted values for the testset using the finalized model

Intermediate Model: Finalize CartFullModel (Based on Tree size i.e. Depth, Variables included as well as the R-Square produced)

Predict on testset

```
CartFullModelPredictTest = predict(object = CartFullModel, newdata = InsDataTestUncapped, type = "vector")
```

Calculate RMSE and MAPE manually.

Participants can calculate RMSE and MAPE using various available functions in R, but that may not

communicate effectively the mathematical aspect behind the calculations

RMSE

```
Act_vs_Pred = CartFullModelPredictTest - InsDataTestUncapped$Losses # Differnce
Act_vs_Pred_Square = Act_vs_Pred^2 # Square
Act_vs_Pred_Square_Mean = mean(Act_vs_Pred_Square) # Mean
Act_vs_Pred_Square_Mean_SqRoot = sqrt(Act_vs_Pred_Square_Mean) # Square Root
Act_vs_Pred_Square_Mean_SqRoot
```

MAPE

```
Act_vs_Pred_Abs = abs(CartFullModelPredictTest - InsDataTestUncapped$Losses) # Absolute Differnce
Act_vs_Pred_Abs_Percent = Act_vs_Pred_Abs/InsDataTestUncapped$Losses # Percent Error
Act_vs_Pred_Abs_Percent_Mean = mean(Act_vs_Pred_Abs_Percent)*100 # Mean
Act_vs_Pred_Abs_Percent_Mean
```

Validate RMSE and MAPE calculation with a function in R

```
UncappedModelAccuaracy = accuracy(f = CartFullModelPredictTest, x = InsDataTestUncapped$Losses)
```

```
################################### PART A Ends ###################################



################################### PART B Starts ###################################


# Look at quantiles to cap the Losses Column
quantile(x = InsData$Losses, probs = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.85, 0.95, 0.99, 1))
# 10%      20%      30%      40%      50%      60%      70%      80%      85%      95%      99%      100%
# 121.9355   194.8197   259.7261   313.7939   354.9379   398.6677   453.5502   538.0226   603.3140   820.8124 1203.9556 3500.0000
# Lets try capping the Losses value to 1200, since that covers about 99% of the data


# Make a copy
InsDataCapped = InsData


# Capping the Losses column
InsDataCapped$CappedLosses = ifelse(test = InsData$Losses > 1200, yes = 1200, no = InsData$Losses)


# Confirm that the CappedLosses has been added the dataset
summary(object = InsDataCapped)


# Check Column Names to find the column number of Losses column
colnames(x = InsDataCapped)
# [1] "Policy.Number"            "Age"
# [3] "Years.of.Driving.Experience" "Number.of.Vehicles"
# [5] "Gender"                   "Married"
# [7] "Vehicle.Age"              "Fuel.Type"
# [9] "Losses"                   "CappedLosses"
```

```
# Remove Losses column
InsDataCapped = InsDataCapped[,-9]

# Boxplot/ Plot
boxplot(x = InsDataCapped[,-1]) # Look for outliers

# Average Capped Losses()
for(i in 2:ncol(InsDataCapped))
{
 if(length(unique(InsDataCapped[,i])) <= 5)
 {
   AverageLoss = aggregate(x = InsDataCapped$CappedLosses, by = list(InsDataCapped[,i]), FUN = mean)
   print(colnames(InsDataCapped)[i])
   print(AverageLoss)
   print("************************************************************")
 }
}

# Random Sampling
set.seed(777) # To ensure reproducibility
Index = sample(x = 1:nrow(InsDataCapped), size = 0.7*nrow(InsDataCapped))

# Create Train dataset
InsDataTrainCapped = InsDataCapped[Index, ]
nrow(InsDataTrainCapped)

# Create Test dataset
InsDataTestCapped = InsDataCapped[-Index, ]
nrow(InsDataTestCapped)
```

Build a full model with default settings

set.seed(123) # To ensure reproducibility of xerrors (cross validated errors while estimating complexity paramter)

CartCappedFullModel = rpart(formula = CappedLosses ~ . , data = InsDataTrainCapped[,-1], method = "anova")

Display Model related output and Model evaluation/ optimization results

CartCappedFullModel

summary(object = CartCappedFullModel)

rpart.plot(x = CartCappedFullModel, type = 4,fallen.leaves = T, cex = 0.6)

printcp(x = CartCappedFullModel)

rsq.rpart(x = CartCappedFullModel)

Lets change rpart.control() to specify certain attributes for tree building

CartModel_11

RpartControl = rpart.control(cp = 0.005)

set.seed(123)

CartModel_11 = rpart(formula = CappedLosses ~ . , data = InsDataTrainCapped[,-1], method = "anova", control = RpartControl)

Display Model related output and Model evaluation/ optimization results

CartModel_11

summary(CartModel_11)

rpart.plot(x = CartModel_11, type = 4,fallen.leaves = T, cex = 0.6)

printcp(x = CartModel_11)

rsq.rpart(x = CartModel_11)

CartModel_12

RpartControl = rpart.control(cp = 0.015) # Increase cp to 0.015

set.seed(123)

CartModel_12 = rpart(formula = CappedLosses ~ . , data = InsDataTrainCapped[,-1], method = "anova", control = RpartControl)

Display Model related output and Model evaluation/ optimization results

CartModel_12

summary(CartModel_12)

rpart.plot(x = CartModel_12, type = 4,fallen.leaves = T, cex = 0.6)

printcp(x = CartModel_12)

rsq.rpart(x = CartModel_12)

CartModel_13

RpartControl = rpart.control(cp = 0.02) # Increase cp to 0.2

set.seed(123)

CartModel_13 = rpart(formula = CappedLosses ~ . , data = InsDataTrainCapped[,-1], method = "anova", control = RpartControl)

Display Model related output and Model evaluation/ optimization results

CartModel_13

summary(CartModel_13)

rpart.plot(x = CartModel_13, type = 4,fallen.leaves = T, cex = 0.6)

printcp(x = CartModel_13)

rsq.rpart(x = CartModel_13)

Finalize CartModel_12 (Based on Tree size - Depth, Variables included as well as the R-Square produced)

Predict on testset

CartModel_12PredictTest = predict(object = CartModel_12, newdata = InsDataTestCapped, type = "vector")

Calculate RMSE and MAPE manually

RMSE

CappedModel_Act_vs_Pred = CartModel_12PredictTest - InsDataTestCapped$CappedLosses # Differnce

CappedModel_Act_vs_Pred_Square = CappedModel_Act_vs_Pred^2 # Square

CappedModel_Act_vs_Pred_Square_Mean = mean(CappedModel_Act_vs_Pred_Square) # Mean

CappedModel_Act_vs_Pred_Square_Mean_SqRoot = sqrt(CappedModel_Act_vs_Pred_Square_Mean) # Square Root

CappedModel_Act_vs_Pred_Square_Mean_SqRoot

MAPE

CappedModel_Act_vs_Pred_Abs = abs(CartModel_12PredictTest - InsDataTestCapped$CappedLosses) # Absolute Differnce

CappedModel_Act_vs_Pred_Abs_Percent = CappedModel_Act_vs_Pred_Abs/InsDataTestCapped$CappedLosses # Percent Error

CappedModel_Act_vs_Pred_Abs_Percent_Mean = mean(CappedModel_Act_vs_Pred_Abs_Percent)*100 # Mean

CappedModel_Act_vs_Pred_Abs_Percent_Mean

Validate the same with a function in R

CappedModelAccuarcy = accuracy(f = CartModel_12PredictTest, x = InsDataTestCapped$CappedLosses)

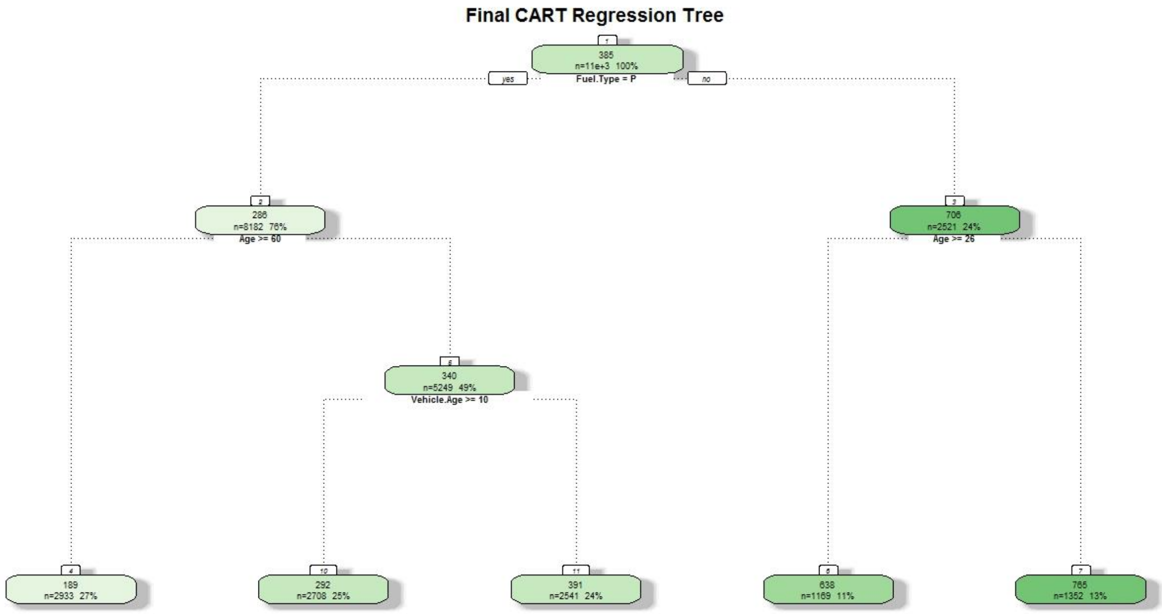
Select one final model from two intermediate finalized models - With the help of RMSE and MAPE

Although, MAPE for the two finalized models, namely, CartFullModel and CartModel_12 happen to be very close,

but there is quite a difference in RMSE of the two models. Based on RMSE, CartModel_12 is the finalized model

windows()

fancyRpartPlot(model = CartModel_12, main = "CartModel_12", cex = 0.6)


Final CART Regression Tree
1
385
n=11e+3 100%
Fuel.Type = P
yes
no
2
286
n=8182 76%
Age >= 60
3
706
n=2521 24%
Age >= 26
5
340
n=5249 49%
Vehicle.Age >= 10
4
189
n=2933 27%
10
292
n=2708 25%
11
391
n=2541 24%
6
638
n=1169 11%
7
765
n=1352 13%

Model 12